Amazon SageMaker

Developer Guide

aws

Amazon SageMaker: Developer Guide

Table of Contents

What Is Amazon SageMaker?

Amazon SageMaker is a fully managed machine learning service. With Amazon SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. With native support for bring-your-own-algorithms and frameworks, Amazon SageMaker offers flexible distributed training options that adjust to your specific workflows. Deploy a model into a secure and scalable environment by launching it with a single click from the Amazon SageMaker console. Training and hosting are billed by minutes of usage, with no minimum fees and no upfront commitments.

This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see HIPAA Overview.

Are You a First-time User of Amazon SageMaker?

If you are a first-time user of Amazon SageMaker, we recommend that you do the following:

1. **Read How Amazon SageMaker Works (p. 2)** – This section provides an overview of Amazon SageMaker, explains key concepts, and describes the core components involved in building AI solutions with Amazon SageMaker. We recommend that you read this topic in the order presented.
2. **Read Get Started (p. 16)** – This section explains how to set up your account and create your first Amazon SageMaker notebook instance.
3. **Try a model training exercise** – This exercise walks you through training your first model. You use training algorithms provided by Amazon SageMaker. For more information, see Get Started (p. 16).
4. **Explore other topics** – Depending on your needs, do the following:
   - **Submit Python code to train with deep learning frameworks** – In Amazon SageMaker, you can use your own training scripts to train models. For information, see Use Machine Learning Frameworks with Amazon SageMaker (p. 440).
   - **Use Amazon SageMaker directly from Apache Spark** – For information, see Use Apache Spark with Amazon SageMaker (p. 440).
   - **Use Amazon AI to train and/or deploy your own custom algorithms** – Package your custom algorithms with Docker so you can train and/or deploy them in Amazon SageMaker. See Use Your Own Algorithms or Models with Amazon SageMaker (p. 384) to learn how Amazon SageMaker interacts with Docker containers, and for the Amazon SageMaker requirements for Docker images.
5. **See the API Reference (p. 616)** – This section describes the Amazon SageMaker API operations.

How Amazon SageMaker Works

Amazon SageMaker is a fully managed service that enables you to quickly and easily integrate machine learning-based models into your applications. This section provides an overview of machine learning and explains how Amazon SageMaker works. If you are a first-time user of Amazon SageMaker, we recommend that you read the following sections in order:

Topics

How It Works: Next Topic

Machine Learning with Amazon SageMaker

This section describes a typical machine learning workflow and summarizes how you accomplish those tasks with Amazon SageMaker.

In machine learning, you "teach" a computer to make predictions, or inferences. First, you use an algorithm and example data to train a model. Then you integrate your model into your application to generate inferences in real time and at scale. In a production environment, a model typically learns from millions of example data items and produces inferences in hundreds to less than 20 milliseconds.

The following diagram illustrates the typical workflow for creating a machine learning model:
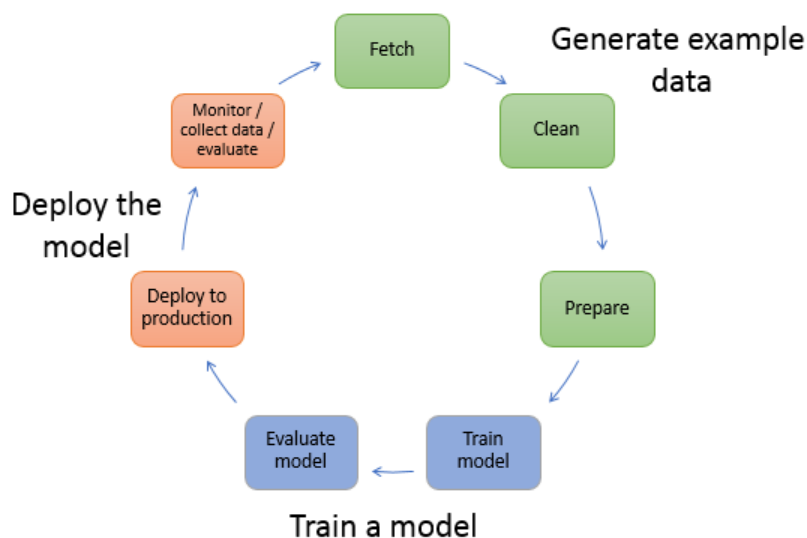
As the diagram illustrates, you typically perform the following activities:

1. **Generate example data**—To train a model, you need example data. The type of data that you need depends on the business problem that you want the model to solve (the inferences that you want

the model to generate). For example, suppose that you want to create a model to predict a number given an input image of a handwritten digit. To train such a model, you need example images of handwritten numbers.

Data scientists often spend a lot of time exploring and preprocessing, or "wrangling," example data before using it for model training. To preprocess data, you typically do the following:

a. **Fetch the data**— You might have in-house example data repositories, or you might use datasets that are publicly available. Typically, you pull the dataset or datasets into a single repository.

b. **Clean the data**—To improve model training, inspect the data and clean it as needed. For example, if your data has a `country name` attribute with values `United States` and `US`, you might want to edit the data to be consistent.

c. **Prepare or transform the data**—To improve performance, you might perform additional data transformations. For example, you might choose to combine attributes. If your model predicts the conditions that require de-icing an aircraft, instead of using temperature and humidity attributes separately, you might combine those attributes into a new attribute to get a better model.

In Amazon SageMaker, you preprocess example data in a Jupyter notebook on your notebook instance. You use your notebook to fetch your dataset, explore it, and prepare it for model training. For more information, see Explore and Preprocess Data (p. 4). For more information about preparing data in AWS Marketplace, see data preparation.

2. **Train a model**—Model training includes both training and evaluating the model, as follows:

- **Training the model**— To train a model, you need an algorithm. The algorithm you choose depends on a number of factors. For a quick, out-of-the-box solution, you might be able to use one of the algorithms that Amazon SageMaker provides. For a list of algorithms provided by Amazon SageMaker and related considerations, see Use Amazon SageMaker Built-in Algorithms (p. 56).

  You also need compute resources for training. Depending on the size of your training dataset and how quickly you need the results, you can use resources ranging from a single general-purpose instance to a distributed cluster of GPU instances. For more information, see Train a Model with Amazon SageMaker (p. 4).

- **Evaluating the model**—After you've trained your model, you evaluate it to determine whether the accuracy of the inferences is acceptable. In Amazon SageMaker, you use either the AWS SDK for Python (Boto) or the high-level Python library that Amazon SageMaker provides to send requests to the model for inferences.

  You use a Jupyter notebook in your Amazon SageMaker notebook instance to train and evaluate your model.

3. **Deploy the model**— You traditionally re-engineer a model before you integrate it with your application and deploy it. With Amazon SageMaker hosting services, you can deploy your model independently, decoupling it from your application code. For more information, see Deploy a Model on Amazon SageMaker Hosting Services (p. 7).

Machine learning is a continuous cycle. After deploying a model, you monitor the inferences, collect "ground truth," and evaluate the model to identify drift. You then increase the accuracy of your inferences by updating your training data to include the newly collected ground truth. You do this by retraining the model with the new dataset. As more and more example data becomes available, you continue retraining your model to increase accuracy.

How It Works: Next Topic

Explore and Preprocess Data (p. 4)

Explore and Preprocess Data

Before using a dataset to train a model, data scientists typically explore and preprocess it. For example, in one of the exercises in this guide, you use the MNIST dataset, a commonly available dataset of handwritten numbers, for model training. Before you begin training, you transform the data into a format that is more efficient for training. For more information, see Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 21).

To preprocess data use one of the following methods:

- Use a Jupyter notebook on an Amazon SageMaker notebook instance. You can also use the notebook instance to do the following:
  - Write code to create model training jobs
  - Deploy models to Amazon SageMaker hosting
  - Test or validate your models

  For more information, see Use Notebook Instances (p. 36)
- You can use a model to transform data by using Amazon SageMaker batch transform. For more information, see Step 6.2: Deploy the Model with Batch Transform (p. 28).

How It Works: Next Topic

Train a Model with Amazon SageMaker

The following diagram shows how you train and deploy a model with Amazon SageMaker:
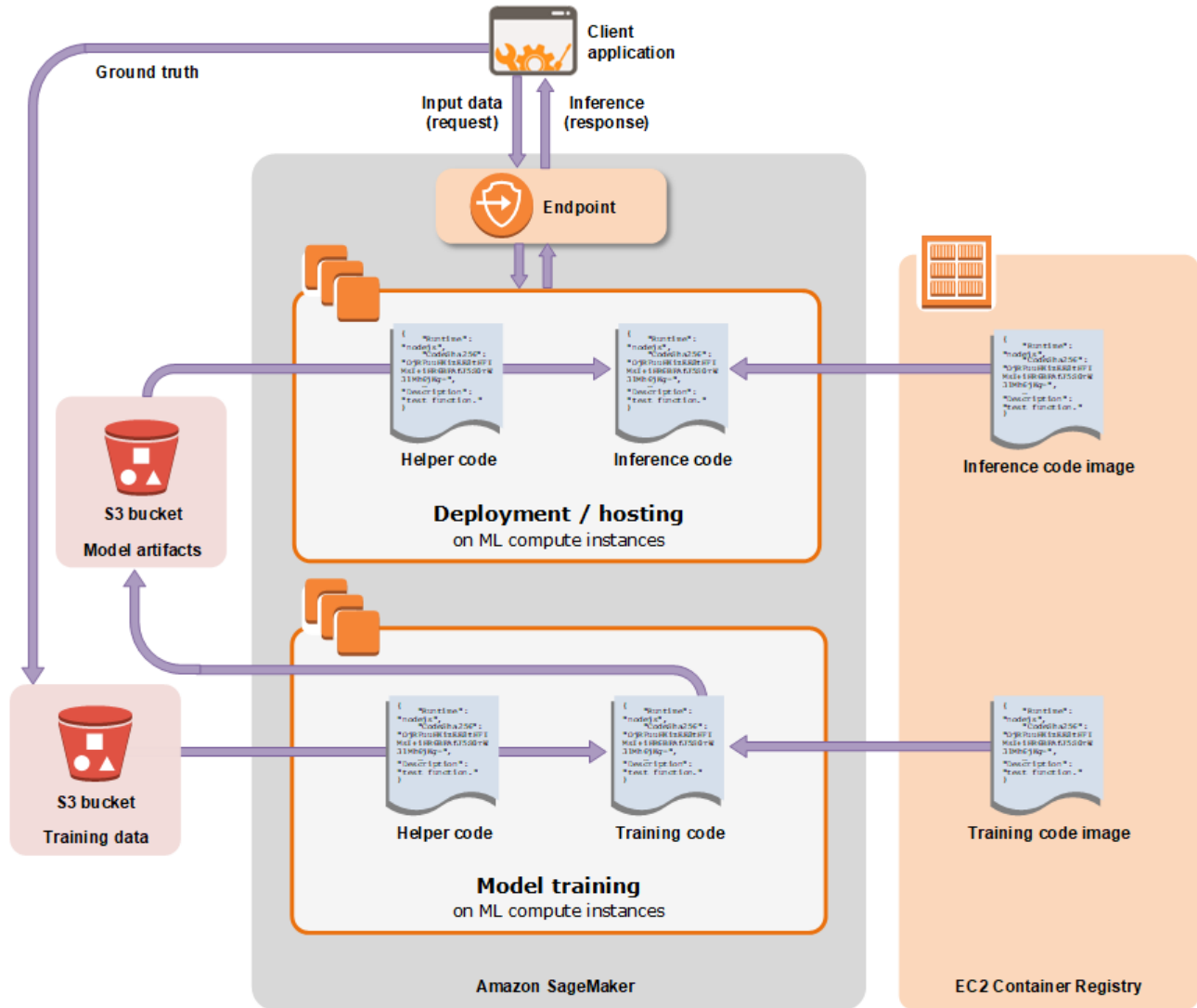
The area labeled Amazon SageMaker highlights the two components of Amazon SageMaker: model training and model deployment.

To train a model in Amazon SageMaker, you create a training job. The training job includes the following information:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The compute resources that you want Amazon SageMaker to use for model training. Compute resources are ML compute instances that are managed by Amazon SageMaker.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see Common Parameters for Built-In Algorithms (p. 58).

You have the following options for a training algorithm:

- **Use an algorithm provided by Amazon SageMaker**—Amazon SageMaker provides training algorithms. If one of these meets your needs, it's a great out-of-the-box solution for quick model

training. For a list of algorithms provided by Amazon SageMaker, see Use Amazon SageMaker Built-in Algorithms (p. 56). To try an exercise that uses an algorithm provided by Amazon SageMaker, see Get Started (p. 16).

- **Use Apache Spark with Amazon SageMaker**—Amazon SageMaker provides a library that you can use in Apache Spark to train models with Amazon SageMaker. Using the library provided by Amazon SageMaker is similar to using Apache Spark MLLib. For more information, see Use Apache Spark with Amazon SageMaker (p. 440).

- **Submit custom code to train with deep learning frameworks**—You can submit custom Python code that uses TensorFlow or Apache MXNet for model training. For more information, see Use TensorFlow with Amazon SageMaker (p. 449) and Use Apache MXNet with Amazon SageMaker (p. 450).

- **Use your own custom algorithms**—Put your code together as a Docker image and specify the registry path of the image in an Amazon SageMaker `CreateTrainingJob` API call. For more information, see Use Your Own Algorithms or Models with Amazon SageMaker (p. 384).

- **Use an algorithm that you subscribe to from AWS Marketplace**—For information, see Find and Subscribe to Algorithms and Model Packages on AWS Marketplace (p. 431).

After you create the training job, Amazon SageMaker launches the ML compute instances and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket you specified for that purpose.

You can create a training job with the Amazon SageMaker console or the API. For information about creating a training job with the API, see the CreateTrainingJob (p. 667) API.

When you create a training job with the API, Amazon SageMaker replicates the entire dataset on ML compute instances by default. To make Amazon SageMaker replicate a subset of the data on each ML compute instance, you must set the `S3DataDistributionType` field to `ShardedByS3Key`. You can set this field using the low-level SDK. For more information, see `S3DataDistributionType` in S3DataSource (p. 994).

> **Important**
> To prevent your algorithm container from contending for memory, you should reserve some memory for Amazon SageMaker critical system processes on your ML compute instances. If the algorithm container is allowed to use memory needed for system processes, it can trigger a system failure.
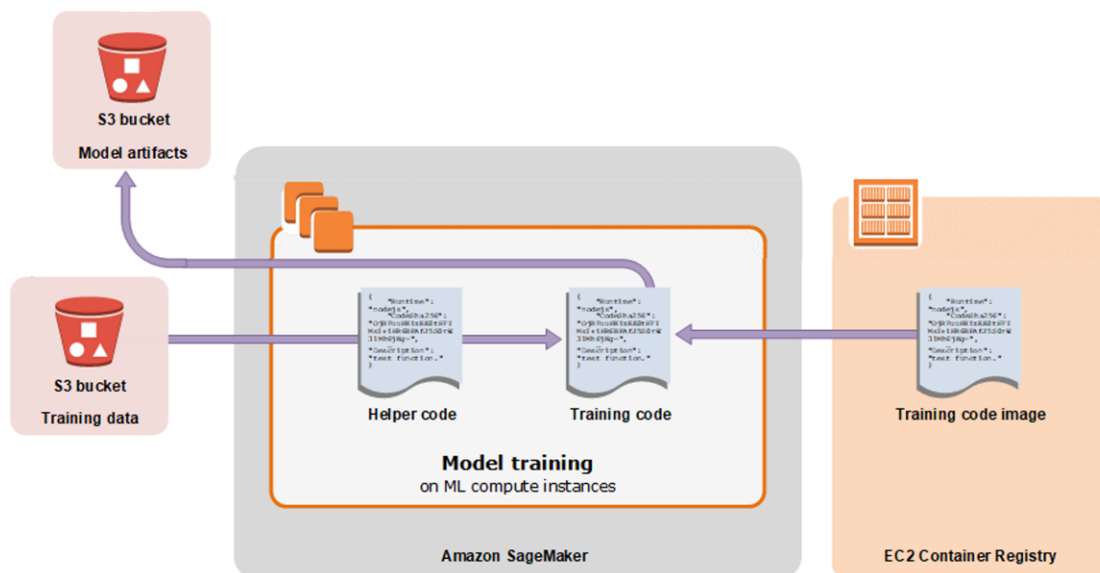
How It Works: Next Topic

Deploy a Model in Amazon SageMaker (p. 7)

Deploy a Model in Amazon SageMaker

After you train your model, you can deploy it to get predictions in one of two ways:

- To set up a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
- To get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics

Deploy a Model on Amazon SageMaker Hosting Services

Amazon SageMaker also provides model hosting services for model deployment, as shown in the following diagram. Amazon SageMaker provides an HTTPS endpoint where your machine learning model is available to provide inferences.
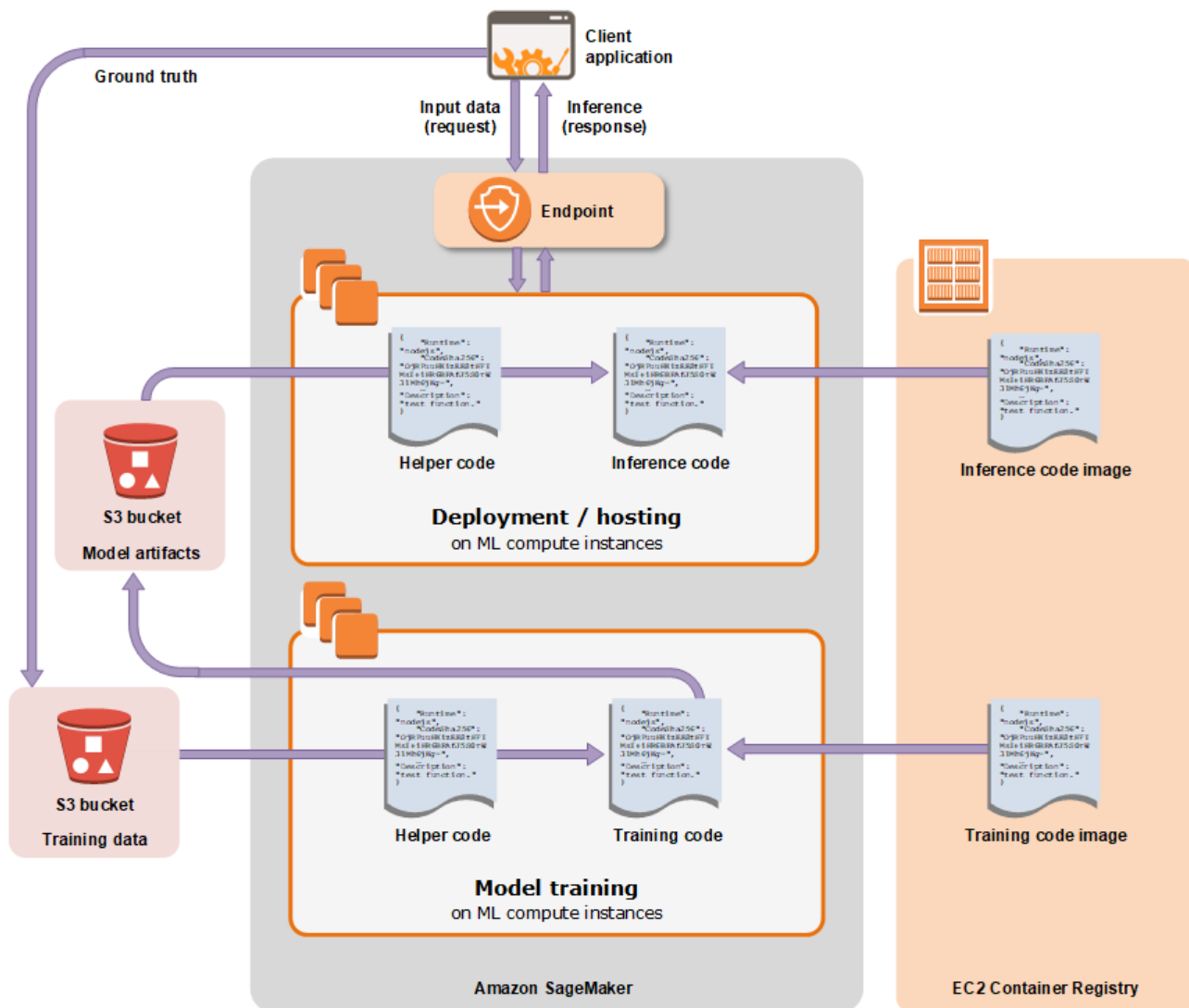
Deploying a model using Amazon SageMaker hosting services is a three-step process:

1. **Create a model in Amazon SageMaker**—By creating a model, you tell Amazon SageMaker where it can find the model components. This includes the S3 path where the model artifacts are stored and the Docker registry path for the image that contains the inference code. In subsequent deployment steps, you specify the model by name. For more information, see the CreateModel (p. 648) API.
2. **Create an endpoint configuration for an HTTPS endpoint**—You specify the name of one or more models in production variants and the ML compute instances that you want Amazon SageMaker to launch to host each production variant.

    When hosting models in production, you can configure the endpoint to elastically scale the deployed ML compute instances. For each production variant, you specify the number of ML compute instances that you want to deploy. When you specify two or more instances, Amazon SageMaker launches them in multiple Availability Zones. This ensures continuous availability. Amazon SageMaker manages deploying the instances. For more information, see the CreateEndpointConfig (p. 635) API.
3. **Create an HTTPS endpoint**—Provide the endpoint configuration to Amazon SageMaker. The service launches the ML compute instances and deploys the model or models as specified in the configuration. For more information, see the CreateEndpoint (p. 632) API. To get inferences from

the model, client applications send requests to the Amazon SageMaker Runtime HTTPS endpoint. For more information about the API, see the InvokeEndpoint (p. 853) API.

> **Note**
> When you create an endpoint, Amazon SageMaker attaches an Amazon EBS storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type. For a list of instance types that Amazon SageMaker hosting service supports, see AWS Service Limits. For a list of the sizes of the storage volumes that Amazon SageMaker attaches to each instance, see Hosting Instance Storage Volumes (p. 381).

To increase a model's accuracy, you might choose to save the user's input data and ground truth, if available, as part of the training data. You can then retrain the model periodically with a larger, improved training dataset.

Best Practices for Deploying Models on Amazon SageMaker Hosting Services

When hosting models using Amazon SageMaker hosting services, consider the following:

- Typically, a client application sends requests to the Amazon SageMaker HTTPS endpoint to obtain inferences from a deployed model. You can also send requests to this endpoint from your Jupyter notebook during testing.

- You can deploy a model trained with Amazon SageMaker to your own deployment target. To do that, you need to know the algorithm-specific format of the model artifacts that were generated by model training. For more information about output formats, see the section corresponding to the algorithm you are using in Training Data Formats (p. 65).

- You can deploy multiple variants of a model to the same Amazon SageMaker HTTPS endpoint. This is useful for testing variations of a model in production. For example, suppose that you've deployed a model into production. You want to test a variation of the model by directing a small amount of traffic, say 5%, to the new model. To do this, create an endpoint configuration that describes both variants of the model. You specify the `ProductionVariant` in your request to the `CreateEndPointConfig`. For more information, see ProductionVariant (p. 981).

- You can configure a `ProductionVariant` to use Application Auto Scaling. For information about configuring automatic scaling, see Automatically Scale Amazon SageMaker Models (p. 365).

- You can modify an endpoint without taking models that are already deployed into production out of service. For example, you can add new model variants, update the ML Compute instance configurations of existing model variants, or change the distribution of traffic among model variants. To modify an endpoint, you provide a new endpoint configuration. Amazon SageMaker implements the changes without any downtime. For more information see, UpdateEndpoint (p. 840) and UpdateEndpointWeightsAndCapacities (p. 842).

- Changing or deleting model artifacts or changing inference code after deploying a model produces unpredictable results. If you need to change or delete model artifacts or change inference code, modify the endpoint by providing a new endpoint configuration. Once you provide the new endpoint configuration, you can change or delete the model artifacts corresponding to the old endpoint configuration.

- If you want to get inferences on entire datasets, consider using batch transform as an alternative to hosting services. For information, see Get Inferences for an Entire Dataset with Batch Transform (p. 10)

How It Works: Next Topic

Validate a Machine Learning Model (p. 11)

Get Inferences for an Entire Dataset with Batch Transform

To get inferences for an entire dataset, use batch transform. With batch transform, you create a batch transform job using a trained model and the dataset, which must be stored in Amazon S3. Amazon SageMaker saves the inferences in an S3 bucket that you specify when you create the batch transform job. Batch transform manages all of the compute resources required to get inferences. This includes launching instances and deleting them after the batch transform job has completed. Batch transform manages interactions between the data and the model with an object within the instance node called an agent.

Use batch transform when you:

- Want to get inferences for an entire dataset and index them to serve inferences in real time
- Don't need a persistent endpoint that applications (for example, web or mobile apps) can call to get inferences
- Don't need the subsecond latency that Amazon SageMaker hosted endpoints provide

You can also use batch transform to preprocess your data before using it to train a new model or generate inferences.

The following diagram shows the workflow of a batch transform job:
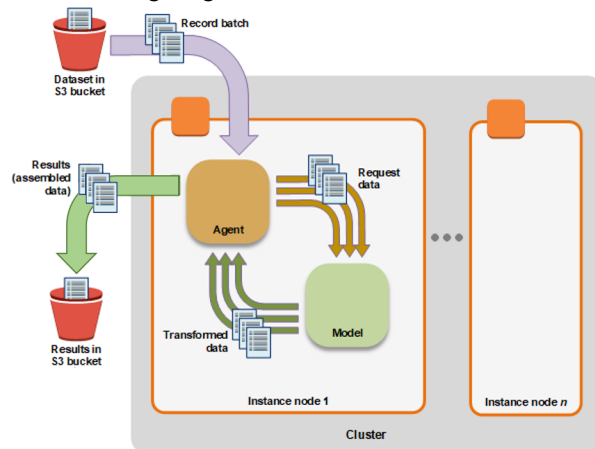
To perform a batch transform, create a batch transform job using either the Amazon SageMaker console or the API. Provide the following:

- The path to the S3 bucket where you've stored the data that you want to transform.
- The compute resources that you want Amazon SageMaker to use for the transform job. *Compute resources* are machine learning (ML) compute instances that are managed by Amazon SageMaker.

- The path to the S3 bucket where you want to store the output of the job.
- The name of the Amazon SageMaker model that you want to use to create inferences. You must use a model that you have already created either with the CreateModel (p. 648) operation or the console.

The following is an example of what a dataset file might look like.

```
An example of input file content:
              Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-
AttributeM
              Record2-Attribute1, Record2-Attribute2, Record2-Attribute3, ..., Record2-
AttributeM
              Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-
AttributeM
              ...
              RecordN-Attribute1, RecordN-Attribute2, RecordN-Attribute3, ..., RecordN-
AttributeM
```

A record is a single input data unit, for information on how to delimit records for batch transform jobs, see `SplitType` in TransformInput (p. 1024).

For an example of how to use batch transform, see Step 6.2: Deploy the Model with Batch Transform (p. 28).

How It Works: Next Topic

Validate a Machine Learning Model (p. 11)

Validate a Machine Learning Model

After training a model, evaluate it to determine whether its performance and accuracy allow you to achieve your business goals. You might generate multiple models using different methods and evaluate each. For example, you could apply different business rules for each model, and then apply various measures to determine each model's suitability. You might consider whether your model needs to be more sensitive than specific (or vice versa).

You can evaluate your model using historical data (offline) or live data:

- **Offline testing**—Use historical, not live, data to send requests to the model for inferences.

  Deploy your trained model to an alpha endpoint, and use historical data to send inference requests to it. To send the requests, use a Jupyter notebook in your Amazon SageMaker notebook instance and either the AWS SDK for Python (Boto) or the high-level Python library provided by Amazon SageMaker.

- **Online testing with live data**—Amazon SageMaker supports deploying multiple models (called production variants) to a single Amazon SageMaker endpoint. You configure the production variants so that a small portion of the live traffic goes to the model that you want to validate. For example, you might choose to send 10% of the traffic to a model variant for evaluation. After you are satisfied with the model's performance, you can route 100% traffic to the updated model.

For more information, see articles and books about how to evaluate models, for example, Evaluating Machine Learning Models.

Options for offline model evaluation include:

- **Validating using a "holdout set"**—Machine learning practitioners often set aside a part of the data as a "holdout set." They don't use this data for model training.

  With this approach, you evaluate how well your model provides inferences on the holdout set. You then assess how effectively the model generalizes what it learned in the initial training, as opposed to using model "memory." This approach to validation gives you an idea of how often the model is able to infer the correct answer.

  In some ways, this approach is similar to teaching elementary school students. First, you provide them with a set of examples to learn, and then test their ability to generalize from their learning. With homework and tests, you pose problems that were not included in the initial learning and determine whether they are able to generalize effectively. Students with perfect memories could memorize the problems, instead of learning the rules.

  Typically, the holdout dataset is of 20-30% of the training data.

- **k-fold validation**—In this validation approach, you split the example dataset into k parts. You treat each of these parts as a holdout set for k training runs, and use the other k-1 parts as the training set for that run. You produce k models using a similar process, and aggregate the models to generate your final model. The value k is typically in the range of 5-10.

How It Works: Next Topic

The Amazon SageMaker Programming Model

Amazon SageMaker provides APIs that you can use to create and manage notebook instances and train and deploy models. For more information, see API Reference (p. 616).

Making API calls directly from code is cumbersome, and requires you to write code to authenticate your requests. Amazon SageMaker provides the following alternatives:

- **Use the Amazon SageMaker console**—With the console, you don't write any code. You use the console UI to start model training or deploy a model. The console works well for simple jobs, where you use a built-in training algorithm and you don't need to preprocess training data.

- **Modify the example Jupyter notebooks**—Amazon SageMaker provides several Jupyter notebooks that train and deploy models using specific algorithms and datasets. Start with a notebook that has a suitable algorithm and modify it to accommodate your data source and specific needs.

- **Write model training and inference code from scratch**—Amazon SageMaker provides both an AWS SDK and a high-level Python library that you can use in your code to start model training jobs and deploy the resulting models.

  - **The high-level Python library**—The Python library simplifies model training and deployment. In addition to authenticating your requests, the library abstracts platform specifics by providing simple methods and default parameters. For example:

- To deploy your model, you call only the `deploy()` method. The method creates an Amazon SageMaker model, an endpoint configuration, and an endpoint.

- If you use a custom framework script for model training, you call the `fit()` method. The method creates a .gzip file of your script, uploads it to an Amazon S3 location, and then runs it for model training, and other tasks. For more information, see Use Machine Learning Frameworks with Amazon SageMaker (p. 440).

- **The AWS SDK** —The SDKs provide methods that correspond to the Amazon SageMaker API (see Actions (p. 616)). Use the SDKs to programmatically start a model training job and host the model in Amazon SageMaker. SDK clients authenticate your requests by using your access keys, so you don't need to write authentication code. They are available in multiple languages and platforms. For more information, see SDKs.

In Get Started (p. 16), you train and deploy a model using an algorithm provided by Amazon SageMaker. That exercise shows how to use both of these libraries. For more information, see Get Started (p. 16).

- **Integrate Amazon SageMaker into your Apache Spark workflow**—Amazon SageMaker provides a library for calling its APIs from Apache Spark. With it, you can use Amazon SageMaker-based estimators in an Apache Spark pipeline. For more information, see Use Apache Spark with Amazon SageMaker (p. 440).

How It Works: Next Topic

Get Started (p. 16)

Set Up Amazon SageMaker

In this section, you sign up for an AWS account and then create an IAM user, a security group, and create an Amazon S3 bucket.

If you're new to Amazon SageMaker, we recommend that you read How Amazon SageMaker Works (p. 2).

Topics

- Step 1: Create an AWS Account (p. 14)
- Step 2: Create an IAM Administrator User and Group (p. 14)

Step 1: Create an AWS Account

In this section, you sign up for an AWS account. If you already have an AWS account, skip this step.

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services, including Amazon SageMaker. You are charged only for the services that you use.

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

    Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

    Write down your AWS account ID because you'll need it for the next task.

Step 2: Create an IAM Administrator User and Group

When you create an AWS account, you get a single sign-in identity that has complete access to all of the AWS services and resources in the account. This identity is called the AWS account *root user*. Signing in to the AWS console using the email address and password that you used to create the account gives you complete access to all of the AWS resources in your account.

We strongly recommend that you *not* use the root user for everyday tasks, even the administrative ones. Instead, adhere to the Create Individual IAM Users, an AWS Identity and Access Management (IAM) administrator user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

To create an administrator user and sign in to the console

1. Create an administrator user in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.

    Note
    We assume that you use administrator user credentials for the exercises and procedures in this guide. If you choose to create and use another IAM user, grant that user minimum permissions. For more information, see Authenticating With Identities (p. 476).

2. Sign in to the AWS Management Console.

   To sign in to the AWS console as a IAM user, you must use a special URL. For more information, see
   How Users Sign In to Your Account in the *IAM User Guide.*

Next Step

Step 1: Create an Amazon S3 Bucket (p. 17)

Get Started

The best way to learn how to use Amazon SageMaker is to create, train, and deploy a simple machine learning model. To do this, you need the following:

- A dataset. You use the MNIST (Modified National Institute of Standards and Technology database) dataset of images of handwritten, single digit numbers. This dataset provides a training set of 50,000 example images of handwritten single-digit numbers, a validation set of 10,000 images, and a test dataset of 10,000 images. You provide this dataset to the algorithm for model training. For more information about the MNIST dataset, see MNIST Dataset.
- An algorithm. You use the XGBoost algorithm provided by Amazon SageMaker to train the model using the MNIST dataset. During model training, the algorithm assigns example data of handwritten numbers into 10 clusters: one for each number, 0 through 9. For more information about the algorithm, see XGBoost Algorithm (p. 255).

You also need a few resources for storing your data and running the code in this exercise:

- An Amazon Simple Storage Service (Amazon S3) bucket to store the training data and the model artifacts that Amazon SageMaker creates when it trains the model.
- An Amazon SageMaker notebook instance to prepare and process data and to train and deploy a machine learning model.
- A Jupyter notebook to use with the notebook instance to prepare your training data and train and deploy the model.

In this exercise, you learn how to create all of the resources that you need to create, train, and deploy a model.

Important
For model training, deployment, and validation, you can use either of the following:

- The high-level Amazon SageMaker Python SDK
- The AWS SDK for Python (Boto 3)

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. This exercise provides code examples for both libraries. If you're a first-time Amazon SageMaker user, we recommend that you use the Amazon SageMaker Python SDK. For more information, see https://sagemaker.readthedocs.io/en/stable/overview.html.

If you're new to Amazon SageMaker, we recommend that you read How Amazon SageMaker Works (p. 2) before starting this exercise.

Topics

Step 1: Create an Amazon S3 Bucket

Training a model produces the following

- The model training data
- Model artifacts, which Amazon SageMaker generates during model training

You save these in an Amazon Simple Storage Service (Amazon S3) bucket: You can store datasets that you use as your training data and model artifacts that are the output of a training job in a single bucket or in two separate buckets. For this exercise and others in this guide, one bucket is sufficient. If you already have S3 buckets, you can use them, or you can create new ones.

To create a bucket, follow the instructions in Create a Bucket in the *Amazon Simple Storage Service Console User Guide*. Include `sagemaker` in the bucket name. For example, `sagemaker-`*`datetime`*.

> **Note**
> Amazon SageMaker needs permission to access these buckets. You grant permission with an IAM role, which you create in the next step when you create an Amazon SageMaker notebook instance. This IAM role automatically gets permissions to access any bucket that has `sagemaker` in the name. It gets these permissions through the `AmazonSageMakerFullAccess` policy, which Amazon SageMaker attaches to the role. If you add a policy to the role that grants the SageMaker service principal `S3FullAccess` permission, the name of the bucket does not need to contain **sagemaker**.

Next Step

Step 2: Create an Amazon SageMaker Notebook Instance

An Amazon SageMaker notebook instance is a fully managed machine learning (ML) Amazon Elastic Compute Cloud (Amazon EC2) compute instance that runs the Jupyter Notebook App. You use the notebook instance to create and manage Jupyter notebooks that you can use to prepare and process data and to train and deploy machine learning models. For more information, see Explore and Preprocess Data (p. 4).

> **Note**
> If necessary, you can change the notebook instance settings, including the ML compute instance type, later.

To create an Amazon SageMaker notebook instance

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Notebook instances**, then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information (if a field is not mentioned, leave the default values):

a.   For **Notebook instance name**, type a name for your notebook instance.

b.   For **Instance type**, choose `ml.t2.medium`. This is the least expensive instance type that notebook instances support, and it suffices for this exercise.

c.   For **IAM role**, choose **Create a new role**, then choose **Create role**.

d.   Choose **Create notebook instance**.

In a few minutes, Amazon SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries.

Next Step

Step 3: Create a Jupyter Notebook

Create a Jupyter notebook in the notebook instance you created in Step 2: Create an Amazon SageMaker Notebook Instance (p. 17), and create a cell that gets the IAM role that your notebook needs to run Amazon SageMaker APIs and specifies the name of the Amazon S3 bucket that you will use to store the datasets that you use for your training data and the model artifacts that a Amazon SageMaker training job outputs.

To create a Jupyter notebook

1.   Open the notebook instance.

a.   Sign in to the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

b.   Open the notebook instance, by choosing either **Open Jupyter** for classic Juypter view or **Open JupyterLab** for JupyterLab view next to the name of the notebook instance. The Jupyter notebook server page appears:

2.   Create a notebook.

a.   If you opened the notebook in Jupyter classic view, on the **Files** tab, choose **New**, and **conda_python3**. This preinstalled environment includes the default Anaconda installation and Python 3.

b.   If you opened the notebook in JupyterLab view, on the **File** menu, choose **New**, and then choose **Notebook..**. For **Select Kernel**, choose **conda_python3**. This preinstalled environment includes the default Anaconda installation and Python 3.

3.   In the Jupyter notebook, choose **File** and **Save as**, and name the notebook.

4.   Copy the following Python code and paste it into the first cell in your notebook. Add the name of the S3 bucket that you created in Set Up Amazon SageMaker (p. 14), and run the code. The `get_execution_role` function retrieves the IAM role you created when you created your notebook instance.

```
import os
import boto3
import re
import copy
import time
from time import gmtime, strftime
from sagemaker import get_execution_role
```

```
role = get_execution_role()

region = boto3.Session().region_name

bucket='bucket-name' # Replace with your s3 bucket name
prefix = 'sagemaker/xgboost-mnist' # Used as part of the path in the bucket where you
 store data
bucket_path = 'https://s3-{}.amazonaws.com/{}'.format(region,bucket) # The URL to
 access the bucket
```

Next Step

Step 4: Download, Explore, and Transform the Training Data

Download the MNIST dataset to your notebook instance, review the data, transform it, and upload it to your S3 bucket.

You transform the data by changing its format from `numpy.array` to comma-separated values (CSV). The XGBoost Algorithm (p. 255) expects input in either the LIBSVM or CSV format. LIBSVM is an open source machine learning library. In this exercise , you use CSV format because it's simpler.

Topics

Step 4.1: Download the MNIST Dataset

To download the MNIST dataset, copy and paste the following code into the notebook and run it:.

```
%%time
import pickle, gzip, urllib.request, json
import numpy as np

# Load the dataset
urllib.request.urlretrieve("http://deeplearning.net/data/mnist/mnist.pkl.gz",
 "mnist.pkl.gz")
with gzip.open('mnist.pkl.gz', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
print(train_set[0].shape)
```

The code does the following:

1. Downloads the MNIST dataset (`mnist.pkl.gz`) from the MNIST Database website to your notebook.
2. Unzips the file and reads the following datasets into the notebook's memory:
   - `train_set` – You use these images of handwritten numbers to train a model.
   - `valid_set` – The XGBoost Algorithm (p. 255) uses these images to evaluate the progress of the model during training.

- `test_set` – You use this set to get inferences to test the deployed model.

Next Step

Step 4.2: Explore the Training Dataset

Typically, you explore training data to determine what you need to clean up and which transformations to apply to improve model training. For this exercise, you don't need to clean up the MNIST dataset.

To explore the dataset

- Type the following code in a cell in your notebook and run the cell to display the first 10 images in `train_set:`.

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (2,10)

for i in range(0, 10):
    img = train_set[0][i]
    label = train_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```

`train_set` contains the following structures:

- `train_set[0]` – Contains images.
- `train_set[1]` – Contains labels.

The code uses the `matplotlib` library to get and display the first 10 images from the training dataset.
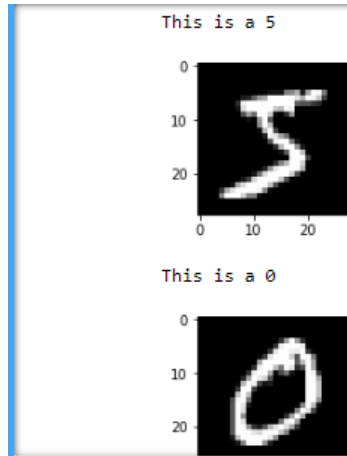
Next Step

Step 4.3: Transform the Training Dataset and Upload It to Amazon S3

The XGBoost Algorithm (p. 255) expects comma-separated values (CSV) for its training input. The format of the training dataset is numpy.array. Transform the dataset from numpy.array format to the CSV format. Then upload it to the Amazon S3 bucket that you created in Step 1: Create an Amazon S3 Bucket (p. 17)

To convert the dataset to CSV format and upload it

* Type the following code into a cell in your notebook and then run the cell.

```
%%time

import struct
import io
import csv
import boto3

def convert_data():
    data_partitions = [('train', train_set), ('validation', valid_set), ('test',
 test_set)]
    for data_partition_name, data_partition in data_partitions:
        print('{}: {} {}'.format(data_partition_name, data_partition[0].shape,
 data_partition[1].shape))
        labels = [t.tolist() for t in data_partition[1]]
        features = [t.tolist() for t in data_partition[0]]

        if data_partition_name != 'test':
            examples = np.insert(features, 0, labels, axis=1)
        else:
            examples = features
        #print(examples[50000,:])


        np.savetxt('data.csv', examples, delimiter=',')



        key = "{}/{}/examples".format(prefix,data_partition_name)
        url = 's3://{}/{}'.format(bucket, key)

 boto3.Session().resource('s3').Bucket(bucket).Object(key).upload_file('data.csv')
        print('Done writing to {}'.format(url))

convert_data()
```

After it converts the dataset to the CSV format, ,the code uploads the CSV file to the S3 bucket.

Next Step

Step 5: Train a Model (p. 21)

Step 5: Train a Model

To train, deploy, and validate a model in Amazon SageMaker, you can use either the Amazon SageMaker Python SDK or the AWS SDK for Python (Boto 3). (You can also use the console, but for this exercise,

you will use the notebook instance and one of the SDKs.) This exercise provides code examples for each library.

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. If you're a first-time Amazon SageMaker user, we recommend that you use it to train, deploy, and validate the model. For more information, see https://sagemaker.readthedocs.io/en/stable/overview.html.

Topics
- Choose the Training Algorithm (p. 22)
- Create and Run a Training Job (Amazon SageMaker Python SDK) (p. 22)
- Create and Run a Training Job (AWS SDK for Python (Boto 3)) (p. 23)

Choose the Training Algorithm

To choose the right algorithm for your model, you typically follow an evaluation process. For this exercise, you use the XGBoost Algorithm (p. 255) provided by Amazon SageMaker, so no evaluation is required. For information about choosing algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 56).

Create and Run a Training Job (Amazon SageMaker Python SDK)

The Amazon SageMaker Python SDK includes the `sagemaker.estimator.Estimator` estimator. You can use this class, in the `sagemaker.estimator` module, with any algorithm. For more information, see https://sagemaker.readthedocs.io/en/stable/estimators.html#sagemaker.estimator.Estimator.

To run a model training job (Amazon SageMaker Python SDK)

1. Import the Amazon SageMaker Python SDK and get the XGBoost container.

```
import sagemaker

from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

2. Download the training and validation data from the Amazon S3 location where you uploaded it in Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 21), and set the location where you store the training output.

```
train_data = 's3://{}/{}/{}'.format(bucket, prefix, 'train')

validation_data = 's3://{}/{}/{}'.format(bucket, prefix, 'validation')

s3_output_location = 's3://{}/{}/{}'.format(bucket, prefix, 'xgboost_model_sdk')
print(train_data)
```

3. Create an instance of the `sagemaker.estimator.Estimator` class.

```
xgb_model = sagemaker.estimator.Estimator(container,
                                          role,
                                          train_instance_count=1,
                                          train_instance_type='ml.m4.xlarge',
                                          train_volume_size = 5,
                                          output_path=s3_output_location,
                                          sagemaker_session=sagemaker.Session())
```

In the constructor, you specify the following parameters:

- `role` – The AWS Identity and Access Management (IAM) role that Amazon SageMaker can assume to perform tasks on your behalf (for example, reading training results, called model artifacts, from the S3 bucket and writing training results to Amazon S3). This is the role that you got in Step 3: Create a Jupyter Notebook (p. 18).

- `train_instance_count` and `train_instance_type` – The type and number of ML compute instances to use for model training. For this exercise, you use only a single training instance.

- `train_volume_size` – The size, in GB, of the Amazon Elastic Block Store (Amazon EBS) storage volume to attach to the training instance. This must be large enough to store training data if you use `File` mode (`File` mode is the default).

- `output_path` – The path to the S3 bucket where Amazon SageMaker stores the training results.

- `sagemaker_session` – The session object that manages interactions with Amazon SageMaker APIs and any other AWS service that the training job uses.

4. Set the hyperparameter values for the XGBoost training job by calling the `set_hyperparameters` method of the estimator. For a description of XGBoost hyperparameters, see XGBoost Hyperparameters (p. 258).

```
xgb_model.set_hyperparameters(max_depth = 5,
                              eta = .2,
                              gamma = 4,
                              min_child_weight = 6,
                              silent = 0,
                              objective = "multi:softmax",
                              num_class = 10,
                              num_round = 10)
```

5. Create the training channels to use for the training job. For this example, we use both `train` and `validation` channels.

```
train_channel = sagemaker.session.s3_input(train_data, content_type='text/csv')
valid_channel = sagemaker.session.s3_input(validation_data, content_type='text/csv')

data_channels = {'train': train_channel, 'validation': valid_channel}
```

6. To start model training, call the estimator's `fit` method.

```
xgb_model.fit(inputs=data_channels,  logs=True)
```

This is a synchronous operation. The method displays progress logs and waits until training completes before returning. For more information about model training, see Train a Model with Amazon SageMaker (p. 4).

Model training for this exercise can take up to 15 minutes.

Next Step

Create and Run a Training Job (AWS SDK for Python (Boto 3))

To train a model, Amazon SageMaker uses the CreateTrainingJob (p. 667) API. The AWS SDK for Python (Boto 3) provides the corresponding `create_training_job` method.

When using this method, you provide the following information:

- The training algorithm – Specify the registry path of the Docker image that contains the training code. For the registry paths for the algorithms provided by Amazon SageMaker, see Common Parameters for Built-In Algorithms (p. 58).
- Algorithm-specific hyperparameters – Specify algorithm-specific hyperparameters to influence the final quality of the model. For information, see XGBoost Hyperparameters (p. 258).
- The input and output configuration – Provide the S3 bucket where training data is stored and where Amazon SageMaker saves the results of model training (the model artifacts).

To run a model training job (AWS SDK for Python (Boto 3))

1. Import the `get_image_url` utility function Amazon SageMaker Python SDK and get the location of the XGBoost container.

```
import sagemaker

from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

2. Set up the training information for the job. You pass this information when you call `create_training_job`. For more information about the information that you need to send to a training job, see the section called "CreateTrainingJob" (p. 667).

```
#Ensure that the train and validation data folders generated above are reflected in the
 "InputDataConfig" parameter below.
common_training_params = \
{
    "AlgorithmSpecification": {
        "TrainingImage": container,
        "TrainingInputMode": "File"
    },
    "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": bucket_path + "/"+ prefix + "/xgboost"
    },
    "ResourceConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.m4.xlarge",
        "VolumeSizeInGB": 5
    },
    "HyperParameters": {
        "max_depth":"5",
        "eta":"0.2",
        "gamma":"4",
        "min_child_weight":"6",
        "silent":"0",
        "objective": "multi:softmax",
        "num_class": "10",
        "num_round": "10"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 86400
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
```

```
                              "S3Uri": bucket_path + "/"+ prefix+ '/train/',
                              "S3DataDistributionType": "FullyReplicated"
                    }
               },
               "ContentType": "text/csv",
               "CompressionType": "None"
          },
          {
               "ChannelName": "validation",
               "DataSource": {
                    "S3DataSource": {
                         "S3DataType": "S3Prefix",
                         "S3Uri": bucket_path + "/"+ prefix+ '/validation/',
                         "S3DataDistributionType": "FullyReplicated"
                    }
               },
               "ContentType": "text/csv",
               "CompressionType": "None"
          }
     ]
}
```

3. Name your training job, and finish configuring the parameters that you send to it.

```
#training job params
training_job_name = 'xgboost-mnist' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Job name is:", training_job_name)

training_job_params = copy.deepcopy(common_training_params)
training_job_params['TrainingJobName'] = training_job_name
training_job_params['ResourceConfig']['InstanceCount'] = 1
```

4. Call `create_training_job` to start the training job, and wait for it to complete. If the training job fails, print the reason that it failed.

```
%%time

region = boto3.Session().region_name
sm = boto3.Session().client('sagemaker')

sm.create_training_job(**training_job_params)


status = sm.describe_training_job(TrainingJobName=training_job_name)
['TrainingJobStatus']
print(status)
sm.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=training_job_name)
status = sm.describe_training_job(TrainingJobName=training_job_name)
['TrainingJobStatus']
print("Training job ended with status: " + status)
if status == 'Failed':
    message = sm.describe_training_job(TrainingJobName=training_job_name)
['FailureReason']
    print('Training failed with the following error: {}'.format(message))
    raise Exception('Training job failed')
```

You now have a trained model. Amazon SageMaker stores the resulting artifacts in your S3 bucket.

Next Step

Step 6: Deploy the Model to Amazon SageMaker

To get predictions, deploy your model. The method you use depends on how you want to generate inferences:

- To get one inference at a time in real time, set up a persistent endpoint using Amazon SageMaker hosting services.
- To get inferences for an entire dataset, use Amazon SageMaker batch transform.

Topics

Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services

To deploy a model in Amazon SageMaker, hosting services, you can use either the Amazon SageMaker Python SDK or the AWS SDK for Python (Boto 3). This exercise provides code examples for both libraries.

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. If you're a first-time Amazon SageMaker user, we recommend that you use it. For more information, see https://sagemaker.readthedocs.io/en/stable/overview.html.

Topics

Deploy the Model to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK)

Deploy the model that you trained in Create and Run a Training Job (Amazon SageMaker Python SDK) (p. 22) by calling the `deploy` method of the `sagemaker.estimator.Estimator` object. This is the same object that you used to train the model. When you call the `deploy` method, specify the number and type of ML instances that you want to use to host the endpoint.

```
xgb_predictor = xgb_model.deploy(initial_instance_count=1,
                                 instance_type='ml.m4.xlarge',
                                 )
```

The `deploy` method creates the deployable model, configures the Amazon SageMaker hosting services endpoint, and launches the endpoint to host the model. For more information, see https://sagemaker.readthedocs.io/en/stable/estimators.html#sagemaker.estimator.Estimator.deploy.

It also returns a `sagemaker.predictor.RealTimePredictor` object, which you can use to get inferences from the model. For information, see https://sagemaker.readthedocs.io/en/stable/predictors.html#sagemaker.predictor.RealTimePredictor.

Next Step

Deploy the Model to Amazon SageMaker Hosting Services (AWS SDK for Python (Boto 3).)

Deploying a model using the AWS SDK for Python (Boto 3) is a three-step process:

1. Create a model in Amazon SageMaker – Send a CreateModel (p. 648) request to provide information such as the location of the S3 bucket that contains your model artifacts and the registry path of the image that contains inference code.
2. Create an endpoint configuration – Send a CreateEndpointConfig (p. 635) request to provide the resource configuration for hosting. This includes the type and number of ML compute instances to launch to deploy the model.
3. Create an endpoint – Send a CreateEndpoint (p. 632) request to create an endpoint. Amazon SageMaker launches the ML compute instances and deploys the model. Amazon SageMaker returns an endpoint. Applications can send requests for inference to this endpoint.

To deploy the model (AWS SDK for Python (Boto 3))

For each of the following steps, paste the code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 18) and run the cell.

1. Create a deployable model by identifying the location of model artifacts and the Docker image that contains the inference code.

   ```
   model_name = training_job_name + '-mod'

   info = sm.describe_training_job(TrainingJobName=training_job_name)
   model_data = info['ModelArtifacts']['S3ModelArtifacts']
   print(model_data)

   primary_container = {
       'Image': container,
       'ModelDataUrl': model_data
   }

   create_model_response = sm.create_model(
       ModelName = model_name,
       ExecutionRoleArn = role,
       PrimaryContainer = primary_container)

   print(create_model_response['ModelArn'])
   ```

2. Create an Amazon SageMaker endpoint configuration by specifying the ML compute instances that you want to deploy your model to.

   ```
   endpoint_config_name = 'DEMO-XGBoostEndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S",
    gmtime())
   print(endpoint_config_name)
   create_endpoint_config_response = sm.create_endpoint_config(
       EndpointConfigName = endpoint_config_name,
       ProductionVariants=[{
           'InstanceType':'ml.m4.xlarge',
           'InitialVariantWeight':1,
           'InitialInstanceCount':1,
           'ModelName':model_name,
           'VariantName':'AllTraffic'}])

   print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])
   ```

3. Create an Amazon SageMaker endpoint.

```
%%time
import time

endpoint_name = 'DEMO-XGBoostEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_name)
create_endpoint_response = sm.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
print(create_endpoint_response['EndpointArn'])

resp = sm.describe_endpoint(EndpointName=endpoint_name)
status = resp['EndpointStatus']
print("Status: " + status)

while status=='Creating':
    time.sleep(60)
    resp = sm.describe_endpoint(EndpointName=endpoint_name)
    status = resp['EndpointStatus']
    print("Status: " + status)

print("Arn: " + resp['EndpointArn'])
print("Status: " + status)
```

This code continuously calls the `describe_endpoint` command in a `while` loop until the endpoint either fails or is in service, and then prints the status of the endpoint. When the status changes to `InService`, the endpoint is ready to serve inference requests.

Next Step

Step 6.2: Deploy the Model with Batch Transform

To get inference for an entire dataset, use batch transform. Amazon SageMaker stores the results in Amazon S3.

For information about batch transforms, see Get Inferences for an Entire Dataset with Batch Transform (p. 10). For an example that uses batch transform, see the batch transform sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker_batch_transform/introduction_to_batch_transform.

Topics

- Deploy a Model with Batch Transform (Amazon SageMaker High-level Python Library) (p. 28)
- Deploy a Model with Batch Transform (SDK for Python (Boto 3)) (p. 29)

Deploy a Model with Batch Transform (Amazon SageMaker High-level Python Library)

The following code creates a `sagemaker.transformer.Transformer` object from the model that you trained in Create and Run a Training Job (Amazon SageMaker Python SDK) (p. 22). Then it calls that object's `transform` method to create a transform job. When you create the `sagemaker.transformer.Transformer` object, you specify the number and type of ML instances to use to perform the batch transform job, and the location in Amazon S3 where you want to store the inferences.

Paste the following code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 18) and run the cell.

```
batch_input =
                's3://{}/{}/test/examples'.format(bucket, prefix) # The location of the
 test dataset

batch_output = 's3://{}/{}/batch-inference'.format(bucket, prefix) # The location to store
 the
results of the batch transform job

transformer = xgb_model.transformer(instance_count=1, instance_type='ml.m4.xlarge',
 output_path=batch_output)

transformer.transform(data=batch_input, data_type='S3Prefix', content_type='text/csv',
 split_type='Line')

transformer.wait()
```

For more information, see https://sagemaker.readthedocs.io/en/stable/transformer.html.

Next Step

Deploy a Model with Batch Transform (SDK for Python (Boto 3))

To run a batch transform job, call the `create_transform_job.` method using the model that you trained in Create and Run a Training Job (AWS SDK for Python (Boto 3)) (p. 23).

To create a batch transform job (SDK for Python (Boto 3))

For each of the following steps, paste the code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 18) and run the cell.

1. Name the batch transform job and specify where the input data (the test dataset) is stored and where to store the job's output.

    ```
    batch_job_name = 'xgboost-mnist-batch' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())

    batch_input = 's3://{}/{}/test/examples'.format(bucket, prefix)
    print(batch_input)

    batch_output = 's3://{}/{}/batch-inference'.format(bucket, prefix)
    ```

2. Configure the parameters that you pass when you call the `create_transform_job` method.

    ```
    request = \
    {
        "TransformJobName": batch_job_name,
        "ModelName": model_name,
        "BatchStrategy": "MultiRecord",
        "TransformOutput": {
            "S3OutputPath": batch_output
        },
        "TransformInput": {
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": batch_input
                }
            },
    ```

```
        "ContentType": "text/csv",
        "SplitType": "Line",
        "CompressionType": "None"
    },
    "TransformResources": {
            "InstanceType": "ml.m4.xlarge",
            "InstanceCount": 1
    }
}
```

For more information about the parameters, see the section called "CreateTransformJob" (p. 673).

3. Call the `create_transform_job` method, passing in the parameters that you configured in the previous step. Then call the `describe_transform_job` method in a loop until it completes.

Paste the following code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 18) and run the cell.

```
sm.create_transform_job(**request)

while(True):
    response = sm.describe_transform_job(TransformJobName=batch_job_name)
    status = response['TransformJobStatus']
    if  status == 'Completed':
        print("Transform job ended with status: " + status)
        break
    if status == 'Failed':
        message = response['FailureReason']
        print('Transform failed with the following error: {}'.format(message))
        raise Exception('Transform job failed')
    print("Transform job is still in status: " + status)
    time.sleep(30)
```

Next Step

Step 7: Validate the Model (p. 30)

Step 7: Validate the Model

Now that you have trained and deployed a model in Amazon SageMaker, validate it to ensure that it generates accurate predictions on new data. That is, on data that is different from the data that the model was trained on. For this, use the test dataset that you created in Step 4: Download, Explore, and Transform the Training Data (p. 19).

Topics
- Step 7.1: Validate a Model Deployed to Amazon SageMaker Hosting Services (p. 30)
- Step 7.2: Validate a Model Deployed with Batch Transform (p. 33)

Step 7.1: Validate a Model Deployed to Amazon SageMaker Hosting Services

If you deployed a model to Amazon SageMaker hosting services in Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26), you now have an endpoint that you can invoke to get inferences in real time. To validate the model, invoke the endpoint with example images from the test dataset and check whether the inferences you get match the actual labels of the images.

Topics

Validate a Model Deployed to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK)

To validate the model by using the Amazon SageMaker Python SDK, use the `sagemaker.predictor.RealTimePredictor` object that you created in Deploy the Model to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK) (p. 26). For information, see https://sagemaker.readthedocs.io/en/stable/predictors.html#sagemaker.predictor.RealTimePredictor.

To validate the model (Amazon SageMaker Python SDK)

1. Download the test data from Amazon S3.

```
s3 = boto3.resource('s3')

test_key = "{}/test/examples".format(prefix)

s3.Bucket(bucket).download_file(test_key, 'test_data')
```

2. Plot the first 10 images from the test dataset with their labels.

```
%matplotlib inline

for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```
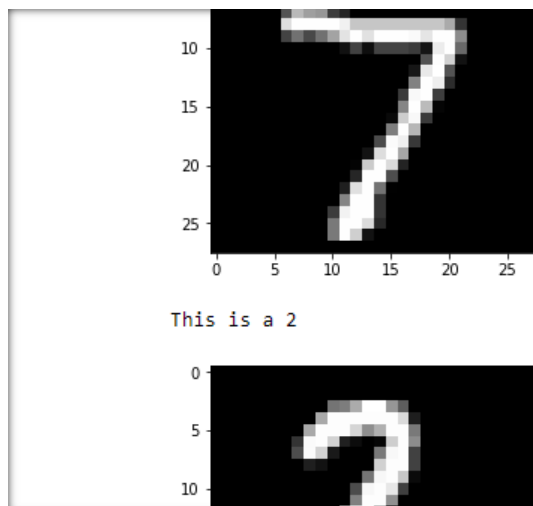

3. To get inferences for the first 10 examples in the test dataset, call the `predict` method of the `sagemaker.predictor.RealTimePredictor` object.

```
with open('test_data', 'r') as f:
    for j in range(0,10):
        single_test = f.readline()
        result = xgb_predictor.predict(single_test)
        print(result)
```

To see if the model is making accurate predictions, check the output from this step against the numbers that you plotted in the previous step.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

Validate a Model Deployed to Amazon SageMaker Hosting Services (AWS SDK for Python (Boto 3))

To use the AWS SDK for Python (Boto 3) to validate the model, call the `invoke_endpoint` method. This method corresponds to the InvokeEndpoint (p. 853) API provided by the Amazon SageMaker runtime.

To validate the model (AWS SDK for Python (Boto 3))

1. Download the test data from Amazon S3.

```
s3 = boto3.resource('s3')

test_key = "{}/test/examples".format(prefix)

s3.Bucket(bucket).download_file(test_key, 'test_data')
```

2. Plot the first 10 images from the test dataset with their labels.

```
%matplotlib inline

for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```

This is a 2
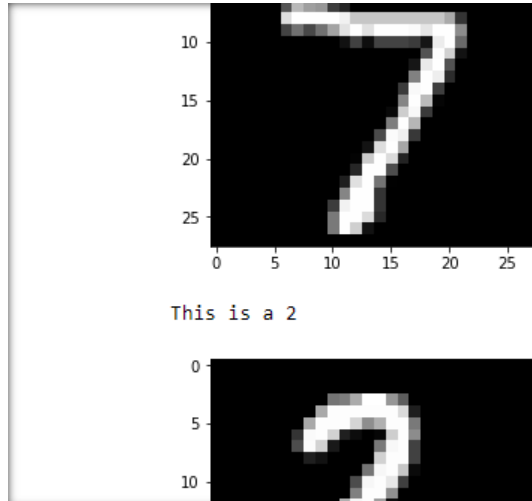
3. Get the Amazon SageMaker runtime client, which provides the `invoke_endpoint` method.

```
runtime_client = boto3.client('runtime.sagemaker')
```

4. Get inferences from the first 10 examples in the test dataset by calling `invoke_endpoint`.

```
with open('test_data', 'r') as f:

    for i in range(0,10):
        single_test = f.readline()
        response = runtime_client.invoke_endpoint(EndpointName = endpoint_name,
                                        ContentType = 'text/csv',
                                        Body = single_test)
        result = response['Body'].read().decode('ascii')
        print('Predicted label is {}.'.format(result))
```

5. To see if the model is making accurate predictions, check the output from this step against the numbers you plotted in the previous step.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

Step 7.2: Validate a Model Deployed with Batch Transform

You now have a file in Amazon S3 that contains inferences that you got by running a batch transform job in Step 6.2: Deploy the Model with Batch Transform (p. 28). To validate the model, check a subset of the inferences from the file to see whether they match the actual numbers from the test dataset.

To validate the batch transform inferences

1. Download the test data from Amazon S3.

```
s3 = boto3.resource('s3')
```

```
test_key = "{}/test/examples".format(prefix)

s3.Bucket(bucket).download_file(test_key, 'test_data')
```

2. Plot the first 10 images from the test dataset with their labels.

```
%matplotlib inline

for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```
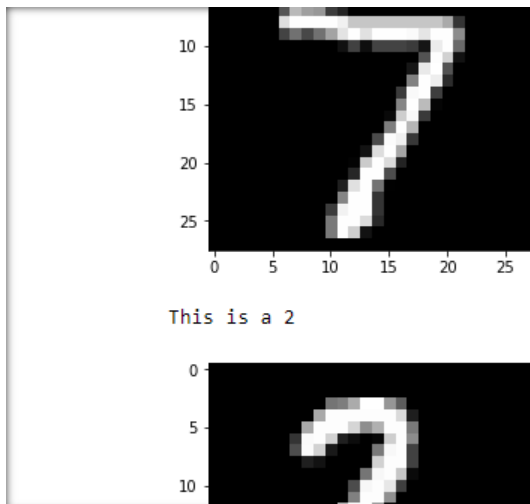

3. Download the output from the batch transform job from Amazon S3 to a local file.

```
s3.Bucket(bucket).download_file(prefix + '/batch-inference/examples.out',
  'batch_results')
```

4. Get the first 10 results from the batch transform job.

```
with open('batch_results') as f:
    results = f.readlines()
for j in range (0, 10):
    print(results[j])
```

5. To see if the batch transform job made accurate predictions, check the output from this step against the numbers that you plotted from the test data.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

Step 8: Clean Up

To avoid incurring unnecessary charges, use the AWS Management Console to delete the resources that you created for this exercise.

Note
If you plan to explore other exercises in this guide, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/ and delete the following resources:
   - The endpoint. Deleting the endpoint also deletes the ML compute instance or instances that support it.
   - The endpoint configuration.
   - The model.
   - The notebook instance. Before deleting the notebook instance, stop it.
2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/ and delete the bucket that you created for storing model artifacts and the training dataset.
3. Open the IAM console at https://console.aws.amazon.com/iam/ and delete the IAM role. If you created permission policies, you can delete them, too.
4. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/ and delete all of the log groups that have names starting with `/aws/sagemaker/`.

Step 9: Integrating Amazon SageMaker Endpoints into Internet-facing Applications

In a production environment, you might have an internet-facing application sending requests to the endpoint for inference. The following high-level example shows how to integrate your model endpoint into your application.

1. Create an IAM role that the AWS Lambda service principal can assume. Give the role permissions to call the Amazon SageMaker `InvokeEndpoint` API.
2. Create a Lambda function that calls the Amazon SageMaker `InvokeEndpoint` API.
3. Call the Lambda function from a mobile application. For an example of how to call a Lambda function from a mobile application using Amazon Cognito for credentials, see Tutorial: Using AWS Lambda as Mobile Application Backend.

Use Notebook Instances

An *Amazon SageMaker notebook instance* is a fully managed ML compute instance running the Jupyter Notebook App. Amazon SageMaker manages creating the instance and related resources. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to Amazon SageMaker hosting, and test or validate your models.

Topics

Create a Notebook Instance

To create a notebook instance, use either the Amazon SageMaker console or the CreateNotebookInstance (p. 656) API.

After receiving the request, Amazon SageMaker does the following:

- **Creates a network interface**—If you choose the optional VPC configuration, it creates the network interface in your VPC. It uses the subnet ID that you provide in the request to determine which Availability Zone to create the subnet in. Amazon SageMaker associates the security group that you provide in the request with the subnet. For more information, see Connect a Notebook Instance to Resources in a VPC (p. 516).
- **Launches an ML compute instance**—Amazon SageMaker launches an ML compute instance in an Amazon SageMaker VPC. Amazon SageMaker performs the configuration tasks that allow it to manage your notebook instance, and if you specified your VPC, it enables traffic between your VPC and the notebook instance.
- **Installs Anaconda packages and libraries for common deep learning platforms**—Amazon SageMaker installs all of the Anaconda packages that are included in the installer. For more information, see Anaconda package list. In addition, Amazon SageMaker installs the TensorFlow and Apache MXNet deep learning libraries.
- **Attaches an ML storage volume**—Amazon SageMaker attaches an ML storage volume to the ML compute instance. You can use the volume to clean up the training dataset or to temporarily store other data to work with. Choose any size between 5 GB and 16384 GB, in 1 GB increments, for the volume. The default is 5 GB. ML storage volumes are encrypted, so Amazon SageMaker can't determine the amount of available free space on the volume. Because of this, you can increase the volume size when you update a notebook instance, but you can't decrease the volume size. If you want to decrease the size of the ML storage volume in use, create a new notebook instance with the desired size.

Important

Only files and data saved within the `/home/ec2-user/SageMaker` folder persist between notebook instance sessions. Files and data that are saved outside this directory are overwritten when the notebook instance stops and restarts.

Note

Each notebook instance's /tmp directory provides a minimum of 10 GB of storage in an instant store. An instance store is temporary, block-level storage that isn't persistent. When the instance is stopped or restarted, Amazon SageMaker deletes the directory's contents. This temporary storage is part of the root volume of the notebook instance.

- **Copies example Jupyter notebooks**— These Python code examples illustrate model training and hosting exercises using various algorithms and training datasets.

To create a notebook instance:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

2. Choose **Notebook instances**, then choose **Create notebook instance**.

3. On the **Create notebook instance** page, provide the following information:

   a. For **Notebook instance name**, type a name for your notebook instance.

   b. For **Instance type**, choose an instance type for your notebook instance. For a list of supported instance types, see Amazon SageMaker Limits.

   c. For **Elastic Inference**, choose an inference accelerator type to associate with the notebook instance, or choose **none**. For information about elastic inference, see Amazon SageMaker Elastic Inference (EI) (p. 355).

   d. For **IAM role**, choose either an existing IAM role in your account that has the necessary permissions to access Amazon SageMaker resources or **Create a new role**. If you choose **Create a new role**, for **Create an IAM role**:

      i. If you want to use S3 buckets other than the one you created in Step 1: Create an Amazon S3 Bucket (p. 17) to store your input data and output, choose them.

      The IAM role automatically has permissions to use any bucket that has `sagemaker` as part of its name. The `AmazonSageMakerFullAccess` policy, which Amazon SageMaker attaches to the role, gives the role those permissions.

      To give access to other S3 buckets from your notebook instance

      - If you're not concerned about users in your AWS account accessing your data, choose **Any S3 bucket**.

      - If your account has sensitive data (such as Human Resources information), restrict access to certain buckets by choosing **Specific S3 buckets**. You can update the permissions policy attached to the role you are creating later.

      - To explicitly control access, restrict access by choosing **None**. Use bucket and object names and tags as supported by the `AmazonSageMakerFullAccess` policy. For more information, see AmazonSageMakerFullAccess Policy (p. 506).

      ii. Choose **Create role**.

      Amazon SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-`*`YYYYMMDD`*`T`*`HHmmSS`*. For example, `AmazonSageMaker-ExecutionRole-20171125T090800`.

      To see the policies that are attached to the role, use the IAM console.

      Open the IAM console at https://console.aws.amazon.com/iam/.

You can see that the following policies are attached to the role:

- A trust policy that allows Amazon SageMaker to assume the role.

- The `AmazonSageMakerFullAccess` AWS managed policy.

- If you gave access to additional S3 bucket(s) when creating this role, the customer managed policy attached to the role. The name of the customer managed policy is `AmazonSageMaker-ExecutionPolicy-YYYYMMDDTHHmmSS`.

For more information about creating your own IAM role, see Amazon SageMaker Roles (p. 496).

e. For **Root access**, to enable root access for all notebook instance users,choose **Enabled**. To disable root access for users, choose **Disabled**.If you enable root access, all notebook instance users have administrator privileges and can access and edit all files on it.

> **Note**
> If you disable root access, you will still be able to set up lifecycle configurations, as described later in this procedure.

f. (Optional) Allow access to resources in your Virtual Private Cloud (VPC).

To access resources in your VPC from the notebook instance

i. Choose the **VPC** and a **SubnetId**.

ii. For **Security Group**, choose your VPC's default security group. For this exercise and others in this guide) the inbound and outbound rules of the default security group are sufficient.

iii. To allow connecting to a resource in your VPC, ensure that the resource resolves to a private IP address in your VPC. For example, to ensure that an Amazon Redshift DNS name resolves to a private IP address, do one of the following:

- Ensure that the Amazon Redshift cluster is not publicly accessible.

- If the Amazon Redshift cluster is publicly accessible, set the `DNS resolution` and `DNS hostnames` VPC parameters to `true`. For more information, see Managing Clusters in an Amazon Virtual Private Cloud (VPC)

iv. By default, a notebook instance can't connect to on-premises resources or to a peer VPC. You can create a lifecycle configuration that creates an entry in your route table that enables connection to on-premises resources or to a peer VPC. For information, see Understanding Amazon SageMaker notebook instance networking configurations and advanced routing options.

g. If you allowed access to resources from your VPC, enable direct internet access. For **Direct internet access**, choose **Enable**. Without internet access, you can't train or host models from notebooks on this notebook instance unless your VPC has a NAT gateway and your security group allows outbound connections. For more information, see Connect a Notebook Instance to Resources in a VPC (p. 516).

h. (Optional) To use shell scripts that run when you create or start the instance, specify a lifecycle configuration. For information, see Customize a Notebook Instance (p. 40)

i. (Optional) If you want Amazon SageMaker to use an AWS Key Management Service (AWS KMS) key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.

j. Specify the size, in GB, of the ML storage volume that is attached to the notebook instance. You can choose a size between 5 GB and 16,384 GB, in 1 GB increments. You can use the volume to clean up the training dataset when you no longer need it or to temporarily store other data to work with.

k. (Optional) To associate Git repositories with the notebook instance, choose a default repository and up to three additional repositories. For more information, see Associate Git Repositories with Amazon SageMaker Notebook Instances (p. 46).

l. Choose **Create notebook instance**.

In a few minutes, Amazon SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see the CreateNotebookInstance (p. 656) API.

4. When the status of the notebook instance is `InService`, choose **Open Jupyter** next to its name to open the classic Jupyter dashboard, or choose **Open JupyterLab** to open the JupyterLab dashboard. For more information, see Access Notebook Instances (p. 39).

The dashboard provides access to:

- Sample notebooks. Amazon SageMaker provides sample notebooks that contain complete code walkthroughs. These walkthroughs show how to use Amazon SageMaker to perform common machine learning tasks. For more information, see Use Example Notebooks (p. 42).
- The kernels for Jupyter, including those that provide support for Python 2 and 3, Apache MXNet, TensorFlow, PySpark, and R. To create a new notebook and choose a kernel for that notebook, use the **New** menu.

For more information about Jupyter notebooks, see The Jupyter notebook.

Access Notebook Instances

To access your Amazon SageMaker notebook instances, choose one of the following options:

- Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

Choose **Notebook instances**. The console displays a list of notebook instances in your account. To open a notebook instance with a standard Jupyter interface, choose **Open Jupyter** for that instance. To open a notebook instance with a JupyterLab interface, choose **Open JupyterLab** for that instance.
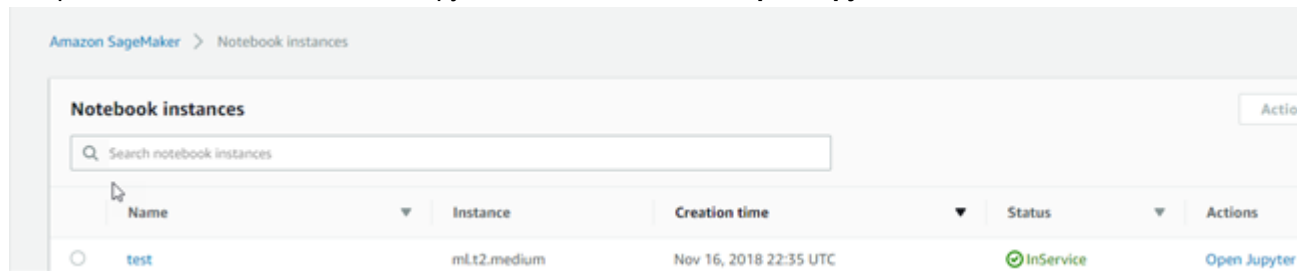
The console uses your sign-in credentials to send a CreatePresignedNotebookInstanceUrl (p. 665) API request to Amazon SageMaker. Amazon SageMaker returns the URL for your notebook instance, and the console opens the URL in another browser tab and displays the Jupyter notebook dashboard.

Note
The URL that you get from a call to CreatePresignedNotebookInstanceUrl (p. 665) is valid only for 5 minutes. If you try to use the URL after the 5-minute limit expires, you are directed to the AWS Management Console sign-in page.

- Use the API.

To get the URL for the notebook instance, call the CreatePresignedNotebookInstanceUrl (p. 665) API and use the URL that the API returns to open the notebook instance.

Use the Jupyter notebook dashboard to create and manage notebooks and to write code. For more information about Jupyter notebooks, see http://jupyter.org/documentation.html.

Control Root Access to a Notebook Instance

By default, when you create a notebook instance, users that log into that notebook instance have root access. Data science is an iterative process that might require the data scientist to test and use different software tools and packages, so many notebook instance users need to have root access to be able to install these tools and packages. Because users with root access have administrator privileges, users can access and edit all files on a notebook instance with root access enabled.

If you don't want users to have root access to a notebook instance, when you call CreateNotebookInstance (p. 656) or UpdateNotebookInstance (p. 844) operations, set the `RootAccess` field to `Disabled`. You can also disable root access for users when you create or update a notebook instance in the Amazon SageMaker console. For information, see Step 2: Create an Amazon SageMaker Notebook Instance (p. 17).

> **Note**
> Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.

Customize a Notebook Instance

To install packages or sample notebooks on your notebook instance, configure networking and security for it, or otherwise use a shell script to customize it, use a lifecycle configuration. A *lifecycle configuration* provides shell scripts that run only when you create the notebook instance or whenever you start one. When you create a notebook instance, you can create a new lifecycle configuration and the scripts it uses or apply one that you already have.

The Amazon SageMaker team maintains a public repository of notebook intance lifecycle configurations that address common use cases for customizing notebook instances at https://github.com/aws-samples/amazon-sagemaker-notebook-instance-lifecycle-configuration-samples.

> **Note**
> Each script has a limit of 16384 characters.
> The value of the `$PATH` environment variable that is available to both scripts is `/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/bin`. The working directory, which is the value of the `$PWD` environment variable, is `/`.
> View CloudWatch Logs for notebook instance lifecycle configurations in log group `/aws/sagemaker/NotebookInstances` in log stream `[notebook-instance-name]/[LifecycleConfigHook]`.
> Scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started. To help decrease the run time of scripts, try the following:
>
> - Cut down on necessary steps. For example, limit which conda environments in which to install large packages.
> - Run tasks in parallel processes.
> - Use the `nohup` command in your script.

To create a lifecycle configuration

1. For **Lifecycle configuration - *Optional***, choose **Create a new lifecycle configuration**.

2. For **Name**, type a name.
3. (Optional) To create a script that runs when you create the notebook and every time you start it, choose **Start notebook**.
4. In the **Start notebook** editor, type the script.
5. (Optional) To create a script that runs only once, when you create the notebook, choose **Create notebook**.
6. In the **Create notebook** editor, type the script configure networking.
7. Choose **Create configuration**.

You can see a list of notebook instance lifecycle configurations you previously created by choosing **Lifecycle configuration** in the Amazon SageMaker console. From there, you can view, edit, delete existing lifecycle configurations. You can create a new notebook instance lifecycle configuration by choosing **Create configuration**. These notebook instance lifecycle configurations are available when you create a new notebook instance.

Lifecycle Configuration Best Practices

The following are best practices for using lifecycle configurations:

- Lifecycle configurations run as the `root` user. If your script makes any changes within the `/home/ec2-user/SageMaker` directory, (for example, installing a package with `pip`), use the command `sudo -u ec2-user` command to run as the `ec2-user` user. This is the same user that Amazon SageMaker runs as.
- Amazon SageMaker notebook instances use `conda` environments to implement different kernels for Jupyter notebooks. If you want to install packages that are available to one or more notebook kernels, enclose the commands to install the packages with `conda` environment commands that activate the conda environment that contains the kernel where you want to install the packages.

  For example, if you want to install a package only in for the `python3` environment, use the following code:

  ```bash
  #!/bin/bash
  sudo -u ec2-user -i <<'EOF'

  # This will affect only the Jupyter kernel called "conda_python3".
  source activate python3

  # Replace myPackage with the name of the package you want to install.
  pip install myPackage
  # You can also perform "conda install" here as well.

  source deactivate

  EOF
  ```

  If you want to install a package in all conda environments in the notebook instance, use the following code:

  ```bash
  #!/bin/bash
  sudo -u ec2-user -i <<'EOF'

  # Note that "base" is special environment name, include it there as well.
  for env in base /home/ec2-user/anaconda3/envs/*; do
      source /home/ec2-user/anaconda3/bin/activate $(basename "$env")

      # Installing packages in the Jupyter system environment can affect stability of your
   SageMaker
  ```

```
     # Notebook Instance.  You can remove this check if you'd like to install Jupyter
 extensions, etc.
     if [ $env = 'JupyterSystemEnv' ]; then
       continue
     fi

     # Replace myPackage with the name of the package you want to install.
     pip install --upgrade --quiet myPackage
     # You can also perform "conda install" here as well.

     source /home/ec2-user/anaconda3/bin/deactivate
done

EOF
```

Important
When you create or change a script file, we recommend you use **Create notebook** editor or
a text editor that allows for Unix style line breaks. Copying text from a non Linux operating
system might include incompatible line breaks and result in an unexpected error.

Use Example Notebooks

Your notebook instance contains example notebooks provided by Amazon SageMaker. The example
notebooks contain code that shows how to apply machine learning solutions by using Amazon
SageMaker. Notebook instances use the `nbexamples` Jupyter extension, which enables you to view a
read-only version of an example notebook or create a copy of it so that you can modify and run it. For
more information about the `nbexamples` extension, see https://github.com/danielballan/nbexamples.

Note
Example notebooks typically download datasets from the internet. If you disable Amazon
SageMaker-provided internet access when you create you notebook instance, example
notebooks might not work. For more information, see Connect a Notebook Instance to
Resources in a VPC (p. 516).

Use or View Example Notebooks in Jupyter Classic

To view or use the example notebooks in the classic Jupyter view, choose the **SageMaker Examples** tab.
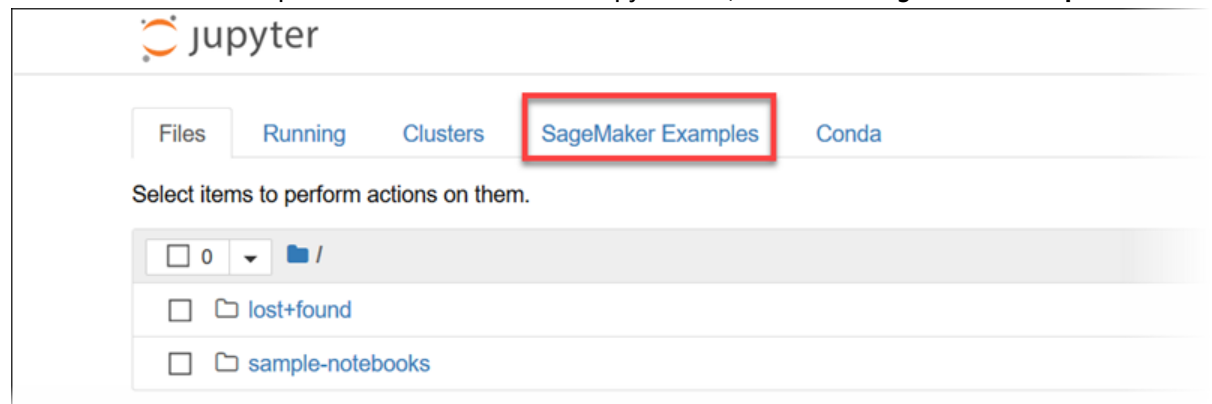
To view a read-only version of an example notebook in the Jupyter classic view, on the **SageMaker
Examples** tab, choose **Preview** for that notebook. To create a copy of an example notebook in the home
directory of your notebook instance, choose **Use**. In the dialog box, you can change the notebook's name
before saving it.
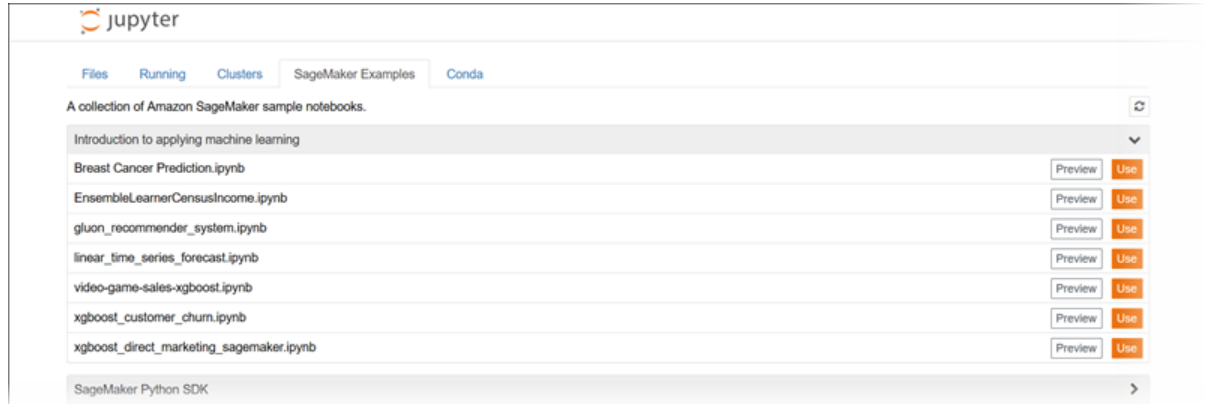
Use or View Example Notebooks in Jupyterlab

To view or use the example notebooks in the Jupyterlab view, choose the examples icon in the left navigation panel.
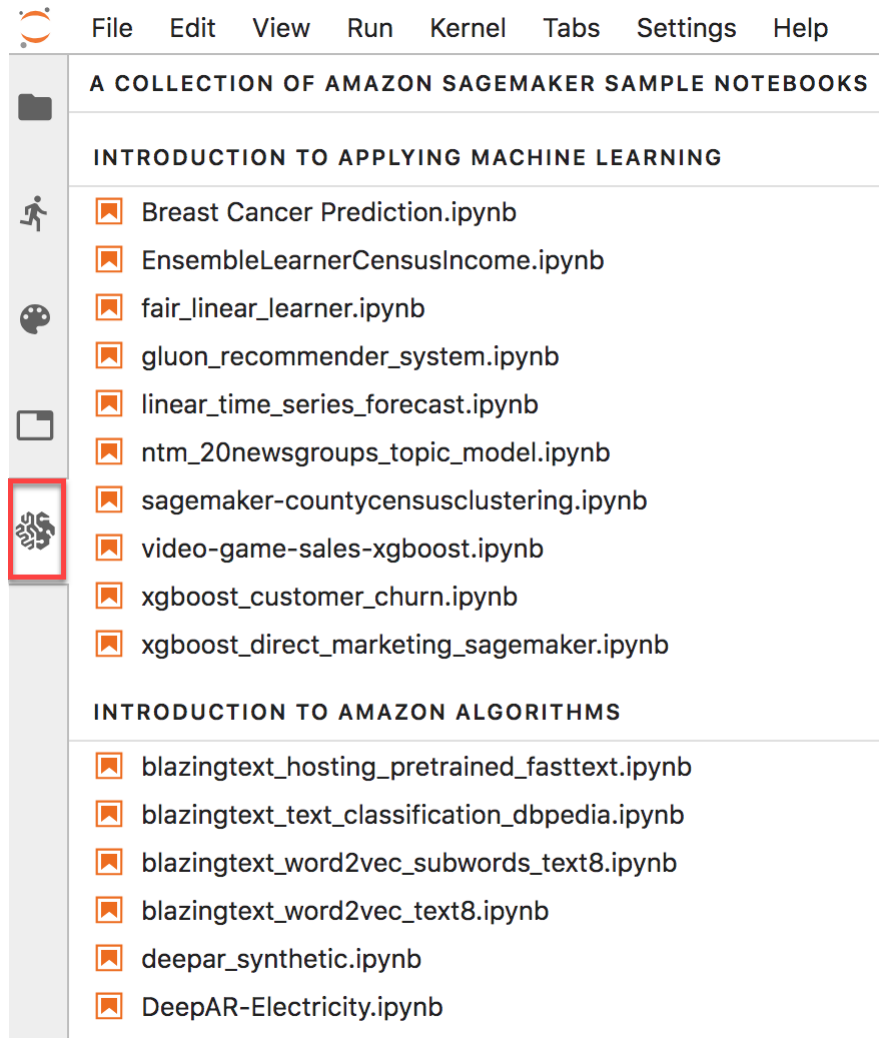
To view a read-only version of an example notebook, choose the name of the notebook. This opens the notebook as a tab in the main area. To create a copy of an example notebook in the home directory of your notebook instance, choose **Create a Copy** in the top banner. In the dialog box, type a name for the notebook and then choose **CREATE COPY**.

For more information about the example notebooks, see the Amazon SageMaker examples GitHub repository.

Notebook Instance Software Updates

Amazon SageMaker periodically tests and releases software that is installed on notebook instances. This includes:

- Kernel updates
- Security patches
- AWS SDK updates
- Amazon SageMaker Python SDK updates
- Open source software updates

Amazon SageMaker does not automatically update software on a notebook instance when it is in service. To ensure that you have the most recent software updates, stop and restart your notebook instance, either in the Amazon SageMaker console or by calling StopNotebookInstance (p. 832) followd by StartNotebookInstance (p. 824).

You can also manually update software installed on your notebook instance while it is running by using update commands in a terminal or in a notebook.

> **Note**
> Updating kernels and some packages might depend on whether root access is enabled for the notebook instance. For more information, see Control Root Access to a Notebook Instance (p. 40).

Notebook instances do not notify you if you are running outdated software. You can check the Personal Health Dashboard or the security bulletin at https://aws.amazon.com/security/security-bulletins/ for updates.

Set the Notebook Kernel

Amazon SageMaker provides several kernels for Jupyter that provide support for Python 2 and 3, Apache MXNet, TensorFlow, and PySpark. To set a kernel for a new notebook in the Jupyter notebook dashboard, choose **New**, and then choose the kernel from the list.

Install External Libraries and Kernels in Notebook Instances

Amazon SageMaker notebook instances come with multiple environments already installed. These environments contain Jupyter kernels and Python packages including: scikit, Pandas, NumPy, TensorFlow, and MXNet. These environments, along with all files in the `sample-notebooks` folder, are refreshed when you stop and start a notebook instance. You can also install your own environments that contain your choice of packages and kernels. This is typically done using `conda install` or `pip install`.

The different Jupyter kernels in Amazon SageMaker notebook instances are separate conda environments. For information about conda environments, see Managing environments in the *Conda* documentation. If you want to use an external library in a specific kernel, install the library in the environment for that kernel. You can do this either in the terminal or in a notebook cell. The following procedures show how to install Theano so that you can use it in a notebook with a conda_mxnet_p36 kernel.

To install Theano from a terminal

1. Open a notebook instance.
2. In the Jupyter dashboard, choose **New**, and then choose **Terminal**.
3. In the terminal, type the following commands:

```
conda install -n mxnet_p36 -c conda-forge theano
   python
   import theano
```

To install Theano from a Jupyter notebook cell

1. Open a notebook instance.
2. In the Jupyter dashboard, choose **New**, and then choose **conda_mxnet_p36**.
3. In a cell in the new notebook, type the following command:

```
!pip install theano
```

Maintain a Sandboxed Python Environment

Amazon SageMaker periodically updates the Python and dependency versions in the environments installed on a notebook instance when it is stopped and restarted. For more information, see Notebook Instance Software Updates (p. 44). To maintain an isolated Python environment that does not change versions, create a lifecycle configuration that runs each time you start your notebook instance. For information about creating lifecycle configurations, see Customize a Notebook Instance (p. 40).

The following example lifecycle configuration script installs Miniconda on your notebook instance. This allows you to create environments in your notebook instance with specific versions of Python and dependencies that Amazon SageMaker does not update:

```
#!/bin/bash

set -e

WORKING_DIR=/home/ec2-user/.myproject
```

```
mkdir -p "$WORKING_DIR"

# Install Miniconda to get a separate python and pip
wget https://repo.anaconda.com/miniconda/Miniconda3-4.5.12-Linux-x86_64.sh -O
 "$WORKING_DIR/miniconda.sh"

# Install Miniconda into the working directory
bash "$WORKING_DIR/miniconda.sh" -b -u -p "$WORKING_DIR/miniconda"

# Install pinned versions of any dependencies
source "$WORKING_DIR/miniconda/bin/activate"
pip install boto3==1.9.86
pip install requests==2.21.0

# Bootstrapping code

# Cleanup
source "$WORKING_DIR/miniconda/bin/deactivate"
rm -rf "$WORKING_DIR/miniconda.sh"
```

You can also add a sandboxed Python installation as a kernel that you can use in a Jupyter notebook by including the following code to the above lifecycle configuration:

```
source "$WORKING_DIR/miniconda/bin/activate"

# If required, add this as a kernel
pip install ipykernel
python -m ipykernel install --user --name MyProjectEnv --display-name "Python
 (myprojectenv)"

source "$WORKING_DIR/miniconda/bin/deactivate"
```

Associate Git Repositories with Amazon SageMaker Notebook Instances

Associate Git repositories with your notebook instance to save your notebooks in a source control environment that persists even if you stop or delete your notebook instance. You can associate one default repository and up to three additional repositories with a notebook instance. The repositories can be hosted in AWS CodeCommit, GitHub or on any other Git server. Associating Git repositories with your notebook instance can be useful for:

- Persistence - Notebooks in a notebook instance are stored on durable Amazon EBS volumes, but they do not persist beyond the life of your notebook instance. Storing notebooks in a Git repository enables you to store and use notebooks even if you stop or delete your notebook instance.
- Collaboration - Peers on a team often work on machine learning projects together. Storing your notebooks in Git repositories allows peers working in different notebook instances to share notebooks and collaborate on them in a source-control environment.
- Learning - Many Jupyter notebooks that demonstrate machine learning techniques are available in publicly hosted Git repositories, such as on GitHub. You can associate your notebook instance with a repository to easily load Jupyter notebooks contained in that repository.

There are two ways to associate a Git repository with a notebook instance:

- Add a Git repository as a resource in your Amazon SageMaker account. Then, to access the repository, you can specify an AWS Secrets Manager secret that contains credentials. That way, you can access repositories that require authentication.

- Associate a public Git repository that is not a resource in your account. If you do this, you cannot specify credentials to access the repository.

Topics

Add a Git Repository to Your Amazon SageMaker Account

To manage your GitHub repositories, easily associate them with your notebook instances, and associate credentials for repositories that require authentication, add the repositories as resources in your Amazon SageMaker account. You can view a list of repositories that are stored in your account and details about each repository in the Amazon SageMaker console and by using the API.

You can add Git repositories to your Amazon SageMaker account in the Amazon SageMaker console or by using the AWS CLI.

> **Note**
> You can use the Amazon SageMaker API CreateCodeRepository (p. 627) to add Git repositories to your Amazon SageMaker account, but step-by-step instructions are not provided here.

Add a Git Repository to Your Amazon SageMaker Account (Console)

To add a Git repository as a resource in your Amazon SageMaker account

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Git repositories**, then choose **Add repository**.
3. To add an CodeCommit repository, choose **AWS CodeCommit**.

    a. To use an existing CodeCommit repository:

      i. Choose **Use existing repository**.
      ii. For **Repository**, choose a repository from the list.
      iii. Enter a name to use for the repository in Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
      iv. Choose **Add repository**.

    b. To create a new CodeCommit repository:

      i. Choose **Create new repository**.
      ii. Enter a name for the repository that you can use in both CodeCommit and Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
      iii. Choose **Create repository**.

4. To add a Git repository hosted somewhere other than CodeCommit :

    a. Choose **GitHub/Other Git-based repo**.
    b. Enter a name to use for the repository in Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

    c.   Enter the URL for the repository.

> **Note**
> Do not provide a user name in the URL. Add the username and password in AWS
> Secrets Manager as described in the next step.

    d.   For **Git credentials**, choose the credentials to use to authenticate to the repository. This is
necessary only if the Git repository is private.

> **Note**
> If you have two-factor authentication enabled for your Git repository, use a personal
> access token generated by your Git service provider instead of a password.

    i.   To use an existing AWS Secrets Manager secret, choose **Use existing secret**, and then
choose a secret from the list. For information about creating and storing a secret, see
Creating a Basic Secret in the *AWS Secrets Manager User Guide*. The name of the secret you
use must contain the string `sagemaker`.

> **Note**
> The secret must have a staging label of `AWSCURRENT` and must be in the following
> format:
> `{"username": UserName, "password": Password}`
> For GitHub repositories, we recommend using a personal access token instead of
> your account password. For information, see https://help.github.com/articles/
> creating-a-personal-access-token-for-the-command-line/.

    ii.   To create a new AWS Secrets Manager secret, choose **Create secret**, enter a name for the
secret, and then enter the username and password to use to authenticate to the repository.
The name for the secret must contain the string `sagemaker`.

> **Note**
> The IAM role you use to create the secret must have the
> `secretsmanager:GetSecretValue` permission in its IAM policy.
> The secret must have a staging label of `AWSCURRENT` and must be in the following
> format:
> `{"username": UserName, "password": Password}`
> For GitHub repositories, we recommend using a personal access token instead of
> your account password.

    iii.   To not use any credentials, choose **No secret**.

    e.   Choose **Create secret**.

Add a Git Repository to Your Amazon SageMaker Account (CLI)

Use the `create-code-repository` AWS CLI command. Specify a name for the repository as the value
of the `code-repository-name` argument. The name must be 1 to 63 characters. Valid characters are a-
z, A-Z, 0-9, and - (hyphen). Also specify the following:

- The default branch
- The URL of the Git repository

> **Note**
> Do not provide a user name in the URL. Add the username and password in AWS Secrets
> Manager as described in the next step.

- The Amazon Resource Name (ARN) of an AWS Secrets Manager secret that contains the credentials to
use to authenticate the repository as the value of the `git-config` argument

For information about creating and storing a secret, see Creating a Basic Secret in the *AWS Secrets
Manager User Guide*. The following command creates a new repository named `MyRespository` in

your Amazon SageMaker account that points to a Git repository hosted at `https://github.com/myprofile/my-repo"`.

For Linux, OS X, or Unix:

```
aws sagemaker create-code-repository \
                --code-repository-name "MyRepository" \
                --git-config '{"Branch":"master", "RepositoryUrl" :
                "https://github.com/myprofile/my-repo", "SecretArn" :
 "arn:aws:secretsmanager:us-east-2:012345678901:secret:my-secret-ABc0DE"}'
```

For Windows:

```
aws sagemaker create-code-repository ^
                --code-repository-name "MyRepository" ^
                --git-config "{\"Branch\":\"master\", \"RepositoryUrl\" :
                \"https://github.com/myprofile/my-repo\", \"SecretArn\" :
 \"arn:aws:secretsmanager:us-east-2:012345678901:secret:my-secret-ABc0DE\"}"
```

> **Note**
> The secret must have a staging label of `AWSCURRENT` and must be in the following format:
> `{"username": `*`UserName`*`, "password": `*`Password`*`}`
> For GitHub repositories, we recommend using a personal access token instead of your account password.

Create a Notebook Instance with an Associated Git Repository

You can associate Git repositories with a notebook instance when you create the notebook instance by using the AWS Management Console, or the AWS CLI.

> **Note**
> You can use the Amazon SageMaker API CreateNotebookInstance (p. 656) to associate Git repositories with a notebook instance, but step-by-step instructions are not provided here.

> **Note**
> If you want to use a CodeCommit repository that is in a different AWS than the notebook instance,set up cross-account access for the repository. For information, see Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance (p. 51).

Topics

Create a Notebook Instance with an Associated Git Repository (Console)

To create a notebook instance and associate Git repositories in the AWS Management Console

1. Follow the instructions at Step 2: Create an Amazon SageMaker Notebook Instance (p. 17).
2. For **Git repositories**, choose Git repositories to associate with the notebook instance.

    a. For **Default repository**, choose a repository that you want to use as your default repository. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory

at `/home/ec2-user/SageMaker`. When you open your notebook instance, it opens in this repository. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose **Add a repository to Amazon SageMaker (opens the Add repository flow in a new window)** and then follow the instructions at Create a Notebook Instance with an Associated Git Repository (Console) (p. 49). To clone a public repository that is not stored in your account, choose **Clone a public Git repository to this notebook instance only**, and then specify the URL for that repository.

b. For **Additional repository 1**, choose a repository that you want to add as an additional directory. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose **Add a repository to Amazon SageMaker (opens the Add repository flow in a new window)** and then follow the instructions at Create a Notebook Instance with an Associated Git Repository (Console) (p. 49). To clone a repository that is not stored in your account, choose **Clone a public Git repository to this notebook instance only**, and then specify the URL for that repository.

Repeat this step up to three times to add up to three additional repositories to your notebook instance.

Create a Notebook Instance with an Associated Git Repository (CLI)

To create a notebook instance and associate Git repositories by using the AWS CLI, use the `create-notebook-instance` command as follows:

- Specify the repository that you want to use as your default repository as the value of the `default-code-repository` argument. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. When you open your notebook instance, it opens in this repository. To use a repository that is stored as a resource in your Amazon SageMaker account, specify the name of the repository as the value of the `default-code-repository` argument. To use a repository that is not stored in your account, specify the URL of the repository as the value of the `default-code-repository` argument.
- Specify up to three additional repositories as the value of the `additional-code-repositories` argument. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`, and the repository is excluded from the default repository by adding it to the `.git/info/exclude` directory of the default repository. To use repositories that are stored as resources in your Amazon SageMaker account, specify the names of the repositories as the value of the `additional-code-repositories` argument. To use repositories that are not stored in your account, specify the URLs of the repositories as the value of the `additional-code-repositories` argument.

For example, the following command creates a notebook instance that has a repository named `MyGitRepo`, that is stored as a resource in your Amazon SageMaker account, as a default repository, and an additional repository that is hosted on GitHub:

```
aws sagemaker create-notebook-instance \
                --notebook-instance-name "MyNotebookInstance" \
                --instance-type "ml.t2.medium" \
                --role-arn "arn:aws:iam::012345678901:role/service-role/
AmazonSageMaker-ExecutionRole-20181129T121390" \
                --default-code-repository "MyGitRepo" \
                --additional-code-repositories "https://github.com/myprofile/my-other-
repo"
```

Amazon SageMaker Developer Guide
Associate a CodeCommit Repository in a
Different AWS Account with a Notebook Instance

Note
If you use an AWS CodeCommit repository that does not contain "SageMaker" in its name, add
the `codecommit:GitPull` and `codecommit:GitPush` permissions to the role that you pass
as the `role-arn` argument to the `create-notebook-instance` command. For information
about how to add permissions to a role, see Adding and Removing IAM Policies in the *AWS
Identity and Access Management User Guide*.

Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance

To associate a CodeCommit repository in a different AWS account with your notebook instance, set up
cross-account access for the CodeCommit repository.

**To set up cross-account access for a CodeCommit repository and associate it with a notebook
instance:**

1. In the AWS account that contains the CodeCommit repository, create an IAM policy that allows
   access to the repository from users in the account that contains your notebook instance. For
   information, see Step 1: Create a Policy for Repository Access in AccountA in the *CodeCommit User
   Guide*.

2. In the AWS account that contains the CodeCommit repository, create an IAM role, and attach the
   policy that you created in the previous step to that role. For information, see Step 2: Create a Role
   for Repository Access in AccountA in the *CodeCommit User Guide*.

3. Create a profile in the notebook instance that uses the role that you created in the previous step:

   a. Open the notebook instance.

   b. Open a terminal in the notebook instance.

   c. Edit a new profile by typing the following in the terminal:

   ```
   vi /home/ec2-user/.aws/config
   ```

   d. Edit the file with the following profile information:

   ```
   [profile CrossAccountAccessProfile]
   region = us-west-2
   role_arn =
    arn:aws:iam::CodeCommitAccount:role/CrossAccountRepositoryContributorRole
   credential_source=Ec2InstanceMetadata
   output = json
   ```

   Where *CodeCommitAccount* is the account that contains the CodeCommit
   repository, *CrossAccountAccessProfile* is the name of the new profile, and
   CrossAccountRepositoryContributorRole is the name of the role you created in the
   previous step.

4. On the notebook instance, configure git to use the profile you created in the previous step:

   a. Open the notebook instance.

   b. Open a terminal in the notebook instance.

   c. Edit the Git configuration file typing the following in the terminal:

   ```
   vi /home/ec2-user/.gitconfig
   ```

   d. Edit the file with the following profile information:

```
[credential]
        helper = !aws codecommit credential-helper --
profile CrossAccountAccessProfile $@
        UseHttpPath = true
```

Where *CrossAccountAccessProfile* is the name of the profile that you created in the previous step.

Use Git Repositories in a Notebook Instance

When you open a notebook instance that has Git repositories associated with it, it opens in the default repository, which is installed in your notebook instance directly under `/home/ec2-user/SageMaker`. You can open and create notebooks, and you can manually run Git commands in a notebook cell. For example:

```
!git pull origin master
```

To open any of the additional repositories, navigate up one folder. The additional repositories are also installed as directories under `/home/ec2-user/SageMaker`.

If you open the notebook instance with a JupyterLab interface, the jupyter-git extension is installed and available to use. For information about the jupyter-git extension for JupyterLab, see https://github.com/jupyterlab/jupyterlab-git.

When you open a notebook instance in JupyterLab, you see the git repositories associated with it on the left menu:
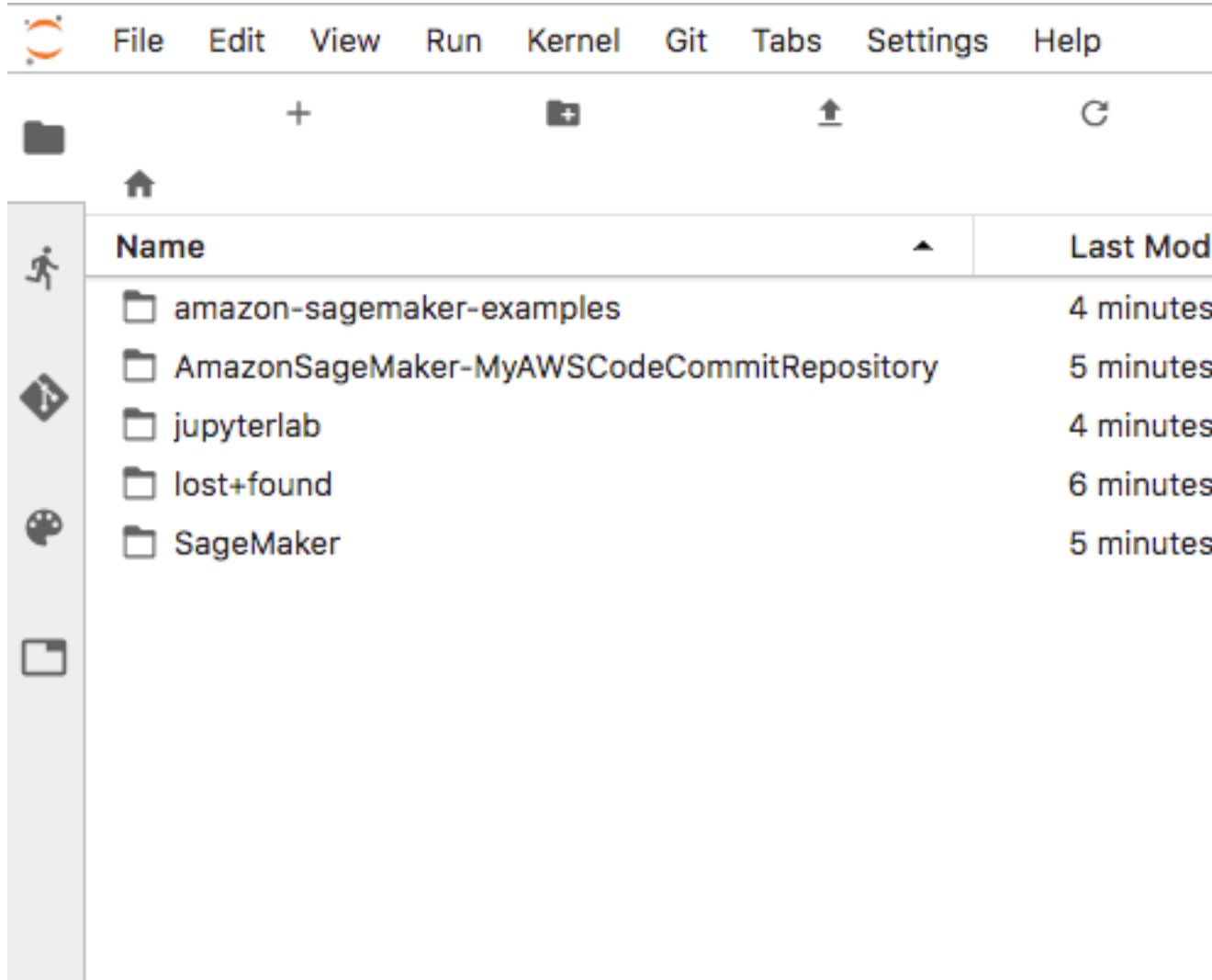
You can use the jupyter-git extension to manage git visually, instead of using the command line:
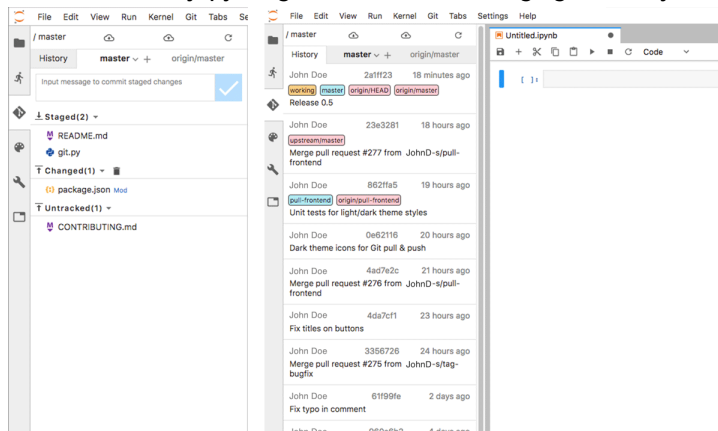
Get Notebook Instance Metadata

When you create a notebook instance, Amazon SageMaker creates a JSON file on the instance at the location `/opt/ml/metadata/resource-metadata.json` that contains the `ResourceName` and `ResourceArn` of the notebook instance. You can access this metadata from anywhere within the notebook instance, including in lifecycle configurations. For information about notebook instance lifecycle configurations, see Customize a Notebook Instance (p. 40).

The `resource-metadata.json` file has the following structure:

```
{
    "ResourceArn": "NotebookInstanceArn",
    "ResourceName": "NotebookInstanceName"
}
```

You can use this metadata from within the notebook instance to get other information about the notebook instance. For example, the following commands get the tags associated with the notebook instance:

```
NOTEBOOK_ARN=$(jq '.ResourceArn'
            /opt/ml/metadata/resource-metadata.json --raw-output)
aws sagemaker list-tags --resource-arn $NOTEBOOK_ARN
```

The out put looks like the following:

```
{
    "Tags": [
        {
            "Key": "test",
            "Value": "true"
        }
    ]
}
```

Monitor Jupyter Logs in Amazon CloudWatch Logs

Jupyter logs include important information such as events, metrics, and health information that provide actionable insights when running Amazon SageMaker notebooks. By importing Jupyter logs into CloudWatch Logs, customers can use CloudWatch Logs to detect anomalous behaviors, set alarms, and discover insights to keep the Amazon SageMaker notebooks running more smoothly. You can access the logs even when the Amazon EC2 instance that hosts the notebook is unresponsive, and use the logs to troubleshoot the unresponsive notebook. Sensitive information such as AWS account IDs, secret keys, and authentication tokens in presigned URLs are removed so that customers can share logs without leaking private information.

To view Jupyter logs for a notebook instance:

1. Sign in to the AWS Management Console and open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Notebook instances**.
3. In the list of notebook instances, choose the notebook instance for which you want to view Jupyter logs.
4. Under **Monitor** on the notebook instance details page, choose **View logs**.

5. In the CloudWatch console, choose the log stream for your notebook intance. Its name is in the form *NotebookInstanceName*/`jupyter.log`.

For more information about monitoring CloudWatch logs for Amazon SageMaker, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 466).

Build a Model

To build a machine learning model in Amazon SageMaker, you have the following options:

- Use one of the built-in algorithims. Amazon SageMaker provides several built-in machine learning algorithms that you can use for a variety of problem types. For more information, see Use Amazon SageMaker Built-in Algorithms (p. 56).
- Write a custom training script in a machine learning framework that Amazon SageMaker supports, and use one of the pre-built framework containers to run it in Amazon SageMaker. For information, see Use Machine Learning Frameworks with Amazon SageMaker (p. 440).
- Bring your own algorithm or model to train or host in Amazon SageMaker. For information, see Use Your Own Algorithms or Models with Amazon SageMaker (p. 384).
- Use an algorithm that you subscribe to from AWS Marketplace. For information, see Amazon SageMaker Resources in AWS Marketplace (p. 428).

Topics

Use Amazon SageMaker Built-in Algorithms

A machine learning algorithm uses example data to create a generalized solution (a *model*) that addresses the business question you are trying to answer. After you create a model using example data, you can use it to answer the same business question for a new set of data. This is also referred to as obtaining inferences.

Amazon SageMaker provides several built-in machine learning algorithms that you can use for a variety of problem types.

Because you create a model to address a business question, your first step is to understand the problem that you want to solve. Specifically, the format of the answer that you are looking for influences the algorithm that you choose. For example, suppose that you are a bank marketing manager, and that you want to conduct a direct mail campaign to attract new customers. Consider the potential types of answers that you're looking for:

- Answers that fit into discrete categories—For example, answers to these questions:

  - "Based on past customer responses, should I mail this particular customer?" Answers to this question fall into two categories, "yes" or "no." In this case, you use the answer to narrow the recipients of the mail campaign.

  - "Based on past customer segmentation, which segment does this customer fall into?" Answers might fall into categories such as "empty nester," "suburban family," or "urban professional." You could use these segments to decide who should receive the mailing.

For this type of discrete classification problem, Amazon SageMaker provides two algorithms: Linear Learner Algorithm (p. 162) and the XGBoost Algorithm (p. 255). You set the following hyperparameters to direct these algorithms to produce discrete results:

- For the Linear Learner algorithm, set the `predictor_type` hyperparameter to `binary_classifier`.

- For the XGBoost algorithm, set the `objective` hyperparameter to `reg:logistic`.

- Answers that are quantitative—Consider this question: "Based on the return on investment (ROI) from past mailings, what is the ROI for mailing this customer?" In this case, you use the ROI to target customers for the mail campaign. For these quantitative analysis problems, you can also use the Linear Learner Algorithm (p. 162) or the XGBoost Algorithm (p. 255) algorithms. You set the following hyperparameters to direct these algorithms to produce quantitative results:

  - For the Linear Learner algorithm, set the `predictor_type` hyperparameter to `regressor`.

  - For the XGBoost algorithm, set the `objective` hyperparameter to `reg:linear`.

- Answers in the form of discrete recommendations—Consider this question: "Based on past responses to mailings, what is the recommended content for each customer?" In this case, you are looking for a recommendation on what to mail, not whether to mail, the customer. For this problem, Amazon SageMaker provides the Factorization Machines Algorithm (p. 98) algorithm.

All of the questions in the preceding examples rely on having example data that includes answers. There are times that you don't need, or can't get, example data with answers. This is true for problems whose answers identify groups. For example:

- "I want to group current and prospective customers into 10 groups based on their attributes. How should I group them? " You might choose to send the mailing to customers in the group that has the highest percentage of current customers. That is, prospective customers that most resemble current customers based on the same set of attributes. For this type of question, Amazon SageMaker provides the K-Means Algorithm (p. 141).

- "What are the attributes that differentiate these customers, and what are the values for each customer along those dimensions." You use these answers to simplify the view of current and prospective customers, and, maybe, to better understand these customer attributes. For this type of question, Amazon SageMaker provides the Principal Component Analysis (PCA) Algorithm (p. 222) algorithm.

In addition to these general-purpose algorithms, Amazon SageMaker provides algorithms that are tailored to specific use cases. These include:

- Image Classification Algorithm (p. 108)—Use this algorithm to classify images. It uses example data with answers (referred to as *supervised algorithm*).

- Sequence-to-Sequence Algorithm (p. 242)—This supervised algorithm is commonly used for neural machine translation.

- Latent Dirichlet Allocation (LDA) Algorithm (p. 157)—This algorithm is suitable for determining topics in a set of documents. It is an *unsupervised algorithm*, which means that it doesn't use example data with answers during training.

- Neural Topic Model (NTM) Algorithm (p. 177)—Another unsupervised technique for determining topics in a set of documents, using a neural network approach.

Topics
- Common Elements of Built-in Algorithms (p. 58)
- BlazingText Algorithm (p. 74)
- DeepAR Forecasting Algorithm (p. 83)
- Factorization Machines Algorithm (p. 98)
- Image Classification Algorithm (p. 108)
- IP Insights Algorithm (p. 131)
- K-Means Algorithm (p. 141)
- K-Nearest Neighbors (k-NN) Algorithm (p. 148)
- Latent Dirichlet Allocation (LDA) Algorithm (p. 157)
- Linear Learner Algorithm (p. 162)
- Neural Topic Model (NTM) Algorithm (p. 177)
- Object2Vec Algorithm (p. 183)
- Object Detection Algorithm (p. 199)
- Principal Component Analysis (PCA) Algorithm (p. 222)
- Random Cut Forest (RCF) Algorithm (p. 226)
- Semantic Segmentation Algorithm (p. 234)
- Sequence-to-Sequence Algorithm (p. 242)
- XGBoost Algorithm (p. 255)

Common Elements of Built-in Algorithms

The following topics provide information common to all of the algorithms provided by Amazon SageMaker.

Topics
- Common Parameters for Built-In Algorithms (p. 58)
- Common Data Formats for Built-in Algorithms (p. 64)
- Instance Types for Built-in Algorithms (p. 72)
- Logs for Built-In Algorithms (p. 73)

Common Parameters for Built-In Algorithms

The following table lists parameters for each of the algorithms provided by Amazon SageMaker.

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class	Parallelizable
BlazingText	train	*<ecr_path>*/ blazingtext:*<tag>*	File or Pipe	Text file (one sentence per line with space-separated tokens)	GPU (single instance only) or CPU	No
DeepAR Forecasting	train and (optionally) test	*<ecr_path>*/forecasting-deepar:*<tag>*	File	JSON Lines or Parquet	GPU or CPU	Yes
Factorization Machines	train and (optionally) test	*<ecr_path>*/factorization-machines:*<tag>*	File or Pipe	recordIO-protobuf	CPU (GPU for dense data)	Yes
Image Classification	train and validation, (optionally) train_lst, validation_lst, and model	*<ecr_path>*/image-classification:*<tag>*	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes
IP Insights	train and (optionally) validation	*<ecr_path>*/ ipinsights:*<tag>*	File	CSV	CPU or GPU	Yes
k-means	train and (optionally) test	*<ecr_path>*/kmeans:*<tag>*	File or Pipe	recordIO-protobuf or CSV	CPU or GPUCommon (single GPU device on one or more instances)	No
k-nearest-neighbor (k-NN)	train and (optionally) test	*<ecr_path>*/knn:*<tag>*	File or Pipe	recordIO-protobuf or CSV	CPU or GPU (single GPU device on one or more instances)	Yes
LDA	train and (optionally) test	*<ecr_path>*/lda:*<tag>*	File or Pipe	recordIO-protobuf or CSV	CPU (single instance only)	No

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class	Parallelizable
Linear Learner	train and (optionally) validation, test, or both	*<ecr_path>*/linear-learner:*<tag>*	File or Pipe	recordIO-protobuf or CSV	CPU or GPU	Yes
Neural Topic Model	train and (optionally) validation, test, or both	*<ecr_path>*/ntm:*<tag>*	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes
Object2Vec	train and (optionally) validation, test, or both	*<ecr_path>*/object2vec:*<tag>*	File	JSON Lines	GPU or CPU (single instance only)	No
Object Detection	train and validation, (optionally) train_annotation, validation_annotation, and model	*<ecr_path>*/object-detection:*<tag>*	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes
PCA	train and (optionally) test	*<ecr_path>*/pca:*<tag>*	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes
Random Cut Forest	train and (optionally) test	*<ecr_path>*/randomcutforest:*<tag>*	File or Pipe	recordIO-protobuf or CSV	CPU	Yes
Semantic Segmentation	train and validation, train_annotation, validation_annotation, and (optionally) label_map and model	*<ecr_path>*/semantic-segmentation:*<tag>*	File or Pipe	image files	GPU (single instance only)	No
Seq2Seq Modeling	train, validation, and vocab	*<ecr_path>*/seq2seq:*<tag>*	File	recordIO-protobuf	GPU (single instance only)	No
XGBoost	train and (optionally) validation	*<ecr_path>*/xgboost:*<tag>*	File	CSV or LibSVM	CPU	Yes

Algorithms that are *parallelizable* can be deployed on multiple compute instances for distributed training. For the **Training Image and Inference Image Registry Path** column, use the `:1` version tag to ensure that you are using a stable version of the algorithm. You can reliably host a model trained using an image with the `:1` tag on an inference image that has the `:1` tag. Using the `:latest` tag in the registry path provides you with the most up-to-date version of the algorithm, but might cause problems with backward compatibility. Avoid using the `:latest` tag for production purposes.

For the **Training Image and Inference Image Registry Path** column, depending on algorithm and region use one of the following values for *<ecr_path>.*

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
Factorization Machines, IP Insights, k-means, k-nearest-neighbor, Linear Learner, Object2Vec, Neural Topic Model,PCA, and Random Cut Forest	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	174872318107.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	382416733822.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	404615174143.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-east-1	286214385809.dkr.ecr.ap-east-1.amazonaws.com
	ap-northeast-1	351501993468.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	835164637446.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	712309505854.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	664544806723.dkr.ecr.eu-central-1.amazonaws.com
	eu-north-1	669576153137.dkr.ecr.eu-north-1.amazonaws.com
	eu-west-1	438346466558.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	749696950732.dkr.ecr.eu-west-3.amazonaws.com
	sa-east-1	855470959533.dkr.ecr.sa-east-1.amazonaws.com
LDA	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	266724342769.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	766337827248.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	999911452149.dkr.ecr.us-east-2.amazonaws.com

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-northeast-1	258307448986.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	293181348795.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	297031611018.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	353608530281.dkr.ecr.eu-central-1.amazonaws.com
	eu-west-1	999678624901.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
BlazingText, Image Classification, Object Detection, Semantic Segmentation, Seq2Seq, and XGBoost (0.72)	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	433757028032.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	811284229777.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	825641698319.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-east-1	286214385809.dkr.ecr.ap-east-1.amazonaws.com
	ap-northeast-1	501404015308.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	306986355934.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	544295431143.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	813361260812.dkr.ecr.eu-central-1.amazonaws.com
	eu-north-1	669576153137.dkr.ecr.eu-north-1.amazonaws.com
	eu-west-1	685385470294.dkr.ecr.eu-west-1.amazonaws.com

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	749696950732.dkr.ecr.eu-west-3.amazonaws.com
	sa-east-1	855470959533.dkr.ecr.sa-east-1.amazonaws.com
XGBoost (0.90)	us-west-1	746614075791.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	246618743249.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	683313688378.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	257758044811.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	414596584902.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-northeast-1	354813040037.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	366743142698.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-southeast-1	121021644041.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	783357654285.dkr.ecr.ap-southeast-2.amazonaws.com
	ap-south-1	720646828776.dkr.ecr.ap-south-1.amazonaws.com
	ap-east-1	651117190479.dkr.ecr.ap-east-1.amazonaws.com
	ca-central-1	341280168497.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	492215442770.dkr.ecr.eu-central-1.amazonaws.com
	eu-north-1	662702820516.dkr.ecr.eu-north-1.amazonaws.com
	eu-west-1	141502667606.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	764974769150.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	659782779980.dkr.ecr.eu-west-3.amazonaws.com
	sa-east-1	737474898029.dkr.ecr.sa-east-1.amazonaws.com
DeepAR Forecasting	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	156387875391.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	522234722520.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	566113047672.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-east-1	286214385809.dkr.ecr.ap-east-1.amazonaws.com

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
	ap-northeast-1	633353088612.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	204372634319.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	514117268639.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-north-1	669576153137.dkr.ecr.eu-north-1.amazonaws.com
	eu-central-1	495149712605.dkr.ecr.eu-central-1.amazonaws.com
	eu-west-1	224300973850.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	749696950732.dkr.ecr.eu-west-3.amazonaws.com
	sa-east-1	855470959533.dkr.ecr.sa-east-1.amazonaws.com

Use the paths and training input mode as follows:

- To create a training job (with a request to the CreateTrainingJob (p. 667) API), specify the Docker Registry path and the training input mode for the training image. You create a training job to train a model using a specific dataset.

- To create a model (with a CreateModel (p. 648) request), specify the Docker Registry path for the inference image. Amazon SageMaker launches machine learning compute instances that are based on the endpoint configuration and deploys the model, which includes the artifacts (the result of model training).

Common Data Formats for Built-in Algorithms

The following topics explain the data formats for the algorithms provided by Amazon SageMaker.

Topics

Common Data Formats for Training

To prepare for training, you can preprocess your data using a variety of AWS services, including AWS Glue, Amazon EMR, Amazon Redshift, Amazon Relational Database Service, and Amazon Athena. After preprocessing, publish the data to an Amazon S3 bucket. For training, the data need to go through a series of conversions and transformations, including:

- Training data serialization (handled by you)
- Training data deserialization (handled by the algorithm)
- Training model serialization (handled by the algorithm)
- Trained model deserialization (optional, handled by you)

When using Amazon SageMaker in the training portion of the algorithm, make sure to upload all data at once. If more data is added to that location, a new training call would need to be made to construct a brand new model.

The following table lists supported ContentType values:

Content Type	Definition
text/csv; label_size=n	Comma-separated values, where n specifies the number of starting columns in a row that are labels. The default value for n is 1.
application/x-recordio-protobuf	A protobuf message wrapped in a RecordIO record.

Training Data Formats

Many Amazon SageMaker algorithms support training with data in CSV format. To use data in CSV format for training, in the input data channel specification, specify **text/csv** as the ContentType. Amazon SageMaker requires that a CSV file doesn't have a header record and that the target variable is in the first column. To run unsupervised learning algorithms that don't have a target, specify the number of label columns in the content type. For example, in this case **'text/csv;label_size=0'**.

Most Amazon SageMaker algorithms work best when you use the optimized protobuf recordIO format for the training data. Using this format allows you to take advantage of *Pipe mode* when training the algorithms that support it. *File mode* loads all of your data from Amazon Simple Storage Service (Amazon S3) to the training instance volumes. In *Pipe mode*, your training job streams data directly from Amazon S3. Streaming can provide faster start times for training jobs and better throughput. With Pipe mode, you also reduce the size of the Amazon Elastic Block Store volumes for your training instances. Pipe mode needs only enough disk space to store your final model artifacts. File mode needs disk space to store both your final model artifacts and your full training dataset. See the AlgorithmSpecification (p. 863) for additional details on the training input mode. For a summary of the data formats supported by each algorithm, see the documentation for the individual algorithms or this table.

> **Note**
> For an example that shows how to convert the commonly used numPy array into the protobuf recordIO format, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/factorization_machines_mnist/factorization_machines_mnist.ipynb. .

In the protobuf recordIO format, Amazon SageMaker converts each observation in the dataset into a binary representation as a set of 4-byte floats and is then loads it to the protobuf values field. If you are using Python for your data preparation, we strongly recommend that you use these existing transformations. However, if you are using another language, the protobuf definition file below provides the schema that you use to convert your data into Amazon SageMaker protobuf format.

```
syntax = "proto2";

package aialgs.data;

option java_package = "com.amazonaws.aialgorithms.proto";
option java_outer_classname = "RecordProtos";
```

```
// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float32Tensor   {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated float values = 1 [packed = true];

    // If key is not empty, the vector is treated as sparse, with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For example, if shape = [ 10, 20 ], floor(keys[i] / 10) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensonal tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float64Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated double values = 1 [packed = true];

    // If this is not empty, the vector is treated as sparse, with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For example, if shape = [ 10, 20 ], floor(keys[i] / 10) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensonal tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// A sparse or dense rank-R tensor that stores data as 32-bit ints (int32).
message Int32Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated int32 values = 1 [packed = true];

    // If this is not empty, the vector is treated as sparse with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For Exmple, if shape = [ 10, 20 ], floor(keys[i] / 10) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensonal tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// Support for storing binary data for parsing in other ways (such as JPEG/etc).
// This is an example of another type of value and may not immediately be supported.
message Bytes {
    repeated bytes value = 1;

    // If the content type of the data is known, stores it.
    // This allows for the possibility of using decoders for common formats
    // in the future.
    optional string content_type = 2;
}
```

```
message Value {
    oneof value {
        // The numbering assumes the possible use of:
        // - float16, float128
        // - int8, int16, int32
        Float32Tensor float32_tensor = 2;
        Float64Tensor float64_tensor = 3;
        Int32Tensor int32_tensor = 7;
        Bytes bytes = 9;
    }
}

message Record {
    // Map from the name of the feature to the value.
    //
    // For vectors and libsvm-like datasets,
    // a single feature with the name `values`
    // should be specified.
    map<string, Value> features = 1;

    // An optional set of labels for this record.
    // Similar to the features field above, the key used for
    // generic scalar / vector labels should ve 'values'.
    map<string, Value> label = 2;

    // A unique identifier for this record in the dataset.
    //
    // Whilst not necessary, this allows better
    // debugging where there are data issues.
    //
    // This is not used by the algorithm directly.
    optional string uid = 3;

    // Textual metadata describing the record.
    //
    // This may include JSON-serialized information
    // about the source of the record.
    //
    // This is not used by the algorithm directly.
    optional string metadata = 4;

    // An optional serialized JSON object that allows per-record
    // hyper-parameters/configuration/other information to be set.
    //
    // The meaning/interpretation of this field is defined by
    // the algorithm author and may not be supported.
    //
    // This is used to pass additional inference configuration
    // when batch inference is used (e.g. types of scores to return).
    optional string configuration = 5;
}
```

After creating the protocol buffer, store it in an Amazon S3 location that Amazon SageMaker can access and that can be passed as part of `InputDataConfig` in `create_training_job`.

> **Note**
> For all Amazon SageMaker algorithms, the `ChannelName` in `InputDataConfig` must be set to `train`. Some algorithms also support a validation or test `input channels`. These are typically used to evaluate the model's performance by using a hold-out dataset. Hold-out datasets are not used in the initial training but can be used to further tune the model.

Trained Model Deserialization

Amazon SageMaker models are stored as model.tar.gz in the S3 bucket specified in `OutputDataConfig` `S3OutputPath` parameter of the `create_training_job` call. You can specify most of these model

artifacts when creating a hosting model. You can also open and review them in your notebook instance. When `model.tar.gz` is untarred, it contains `model_algo-1`, which is a serialized Apache MXNet object. For example, you use the following to load the k-means model into memory and view it:

```
import mxnet as mx
print(mx.ndarray.load('model_algo-1'))
```

Common Data Formats for Inference

Amazon SageMaker algorithms accept and produce several different MIME types for the http payloads used in retrieving online and mini-batch predictions. You can use various AWS services to transform or preprocess records prior to running inference. At a minimum, you need to convert the data for the following:

- Inference request serialization (handled by you)
- Inference request deserialization (handled by the algorithm)
- Inference response serialization (handled by the algorithm)
- Inference response deserialization (handled by you)

Convert Data for Inference Request Serialization

Content type options for Amazon SageMaker algorithm inference requests include: `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Algorithms that don't support these types, such as XGBoost, which is incompatible, support other types, such as `text/x-libsvm`.

For `text/csv` the value for the Body argument to `invoke_endpoint` should be a string with commas separating the values for each feature. For example, a record for a model with four features might look like: `1.5,16.0,14,23.0`. Any transformations performed on the training data should also be performed on the data before obtaining inference. The order of the features matters, and must remain unchanged.

`application/json` is significantly more flexible and provides multiple possible formats for developers to use in their applications. At a high level, in JavaScript, the payload might look like:

```
let request = {
  // Instances might contain multiple rows that predictions are sought for.
  "instances": [
    {
      // Request and algorithm specific inference parameters.
      "configuration": {},
      // Data in the specific format required by the algorithm.
      "data": {
        "<field name>": dataElement
      }
    }
  ]
}
```

You have the following options for specifying the `dataElement`:

Protocol buffers equivalent:

```
// Has the same format as the protocol buffers implementation described for training.
let dataElement = {
  "keys": [],
  "values": [],
  "shape": []
}
```

Simple numeric vector:

```
// An array containing numeric values is treated as an instance containing a
// single dense vector.
let dataElement = [1.5, 16.0, 14.0, 23.0]

// It will be converted to the following representation by the SDK.
let converted = {
  "features": {
    "values": dataElement
  }
}
```

And, for multiple records:

```
let request = {
  "instances": [
    // First instance.
    {
      "features": [ 1.5, 16.0, 14.0, 23.0 ]
    },
    // Second instance.
    {
      "features": [ -2.0, 100.2, 15.2, 9.2 ]
    }
  ]
}
```

Convert Data for Inference Response Deserialization

Amazon SageMaker algorithms return JSON in several layouts. At a high level, the structure is:

```
let response = {
  "predictions": [{
    // Fields in the response object are defined on a per algorithm-basis.
  }]
}
```

The fields that are included in predictions differ across algorithms. The following are examples of output for the k-means algorithm.

Single-record inference:

```
let response = {
  "predictions": [{
    "closest_cluster": 5,
    "distance_to_cluster": 36.5
  }]
}
```

Multi-record inference:

```
let response = {
  "predictions": [
    // First instance prediction.
    {
      "closest_cluster": 5,
      "distance_to_cluster": 36.5
    },
    // Second instance prediction.
```

```
    {
      "closest_cluster": 2,
      "distance_to_cluster": 90.3
    }
  ]
}
```

Multi-record inference with protobuf input:

```
{
  "features": [],
  "label": {
    "closest_cluster": {
      "values": [ 5.0 ] // e.g. the closest centroid/cluster was 1.0
    },
    "distance_to_cluster": {
      "values": [ 36.5 ]
    }
  },
  "uid": "abc123",
  "metadata": "{ "created_at": '2017-06-03' }"
}
```

Amazon SageMaker algorithms also support jsonlines format, where the per-record response content is same as that in JSON format. The multi-record structure is a concatenation of per-record response objects separated by newline characters. The response content for the built-in KMeans algorithm for 2 input data points is:

```
{"distance_to_cluster": 23.40593910217285, "closest_cluster": 0.0}
{"distance_to_cluster": 27.250282287597656, "closest_cluster": 0.0}
```

While running batch transform, it is recommended to use `jsonlines` response type by setting the `Accept` field in the `CreateTransformJobRequest` to `application/jsonlines`.

Common Request Formats for All Algorithms

Most algorithms use several of the following inference request formats.

JSON Request Format

Content-type: application/json

Dense Format

```
let request =   {
    "instances":    [
        {
            "features": [1.5, 16.0, 14.0, 23.0]
        }
    ]
}


let request =   {
    "instances":    [
        {
            "data": {
                "features": {
                    "values": [ 1.5, 16.0, 14.0, 23.0]
                }
            }
        }
```

```
        }
    ]
}
```

Sparse Format

```
{
 "instances": [
  {"data": {"features": {
     "keys": [26, 182, 232, 243, 431],
     "shape": [2000],
     "values": [1, 1, 1, 4, 1]
    }
   }
  },
  {"data": {"features": {
     "keys": [0, 182, 232, 243, 431],
     "shape": [2000],
     "values": [13, 1, 1, 4, 1]
    }
   }
  },
 ]
}
```

JSONLINES Request Format

Content-type: application/jsonlines

Dense Format

A single record in dense format can be represented as either:

```
{ "features": [1.5, 16.0, 14.0, 23.0] }
```

or:

```
{ "data": { "features": { "values": [ 1.5, 16.0, 14.0, 23.0] } } }
```

Sparse Format

A single record in sparse format is represented as:

```
{"data": {"features": { "keys": [26, 182, 232, 243, 431], "shape": [2000], "values": [1, 1,
 1, 4, 1] } } }
```

Multiple records are represented as a concatenation of the above single-record representations,
separated by newline characters:

```
{"data": {"features": { "keys": [0, 1, 3], "shape": [4], "values": [1, 4, 1] } } }
{ "data": { "features": { "values": [ 1.5, 16.0, 14.0, 23.0] } } }
{ "features": [1.5, 16.0, 14.0, 23.0] }
```

CSV Request Format

Content-type: text/csv;label_size=0

> **Note**
> CSV support is not available for factorization machines.

RECORDIO Request Format

Content-type: application/x-recordio-protobuf

Use Batch Transform with Build-in Algorithms

While running batch transform, it's recommended to use jsonlines response type instead of JSON, if supported by the algorithm. This is accomplished by setting the `Accept` field in the `CreateTransformJobRequest` to `application/jsonlines`.

When you create a transform job, the `SplitType` must be set according to the `ContentType` of the input data. Similarly, depending on the `Accept` field in the `CreateTransformJobRequest`, `AssembleWith` must be set accordingly. Please use the following table to help appropriately set these fields:

ContentType	Recommended SplitType
`application/x-recordio-protobuf`	`RecordIO`
`text/csv`	`Line`
`application/jsonlines`	`Line`
`application/json`	`None`
`application/x-image`	`None`
`image/*`	`None`
Accept	**Recommended AssembleWith**
`application/x-recordio-protobuf`	`None`
`application/json`	`None`
`application/jsonlines`	`Line`

For more information on response formats for specific algorithms, see the following:

- PCA Response Formats (p. 226)
- Linear Learner Response Formats (p. 175)
- NTM Response Formats (p. 182)
- K-Means Response Formats (p. 147)
- Factorization Machine Response Formats (p. 107)

Instance Types for Built-in Algorithms

For training and hosting Amazon SageMaker algorithms, we recommend using the following EC2 instance types:

- ml.m4.xlarge, ml.m4.4xlarge, and ml.m4.10xlarge
- ml.c4.xlarge, ml.c4.2xlarge, and ml.c4.8xlarge
- ml.p2.xlarge, ml.p2.8xlarge, and ml.p2.16xlarge

Most Amazon SageMaker algorithms have been engineered to take advantage of GPU computing for training. Despite higher per-instance costs, GPUs train more quickly, making them more cost effective.

Exceptions, such as XGBoost, are noted in this guide. (XGBoost implements an open-source algorithm that has been optimized for CPU computation.)

The size and type of data can have a great effect on which hardware configuration is most effective. When the same model is trained on a recurring basis, initial testing across a spectrum of instance types can discover configurations that are more cost effective in the long run. Additionally, algorithms that train most efficiently on GPUs might not require GPUs for efficient inference. Experiment to determine the most cost effectiveness solution.

For more information on Amazon SageMaker hardware specifications, see Amazon SageMaker ML Instance Types.

Logs for Built-In Algorithms

Amazon SageMaker algorithms produce Amazon CloudWatch logs, which provide detailed information on the training process. To see the logs, in the AWS management console, choose **CloudWatch**, choose **Logs**, and then choose the /aws/sagemaker/TrainingJobs **log group**. Each training job has one log stream per node that it was trained on. The log stream's name begins with the value specified in the `TrainingJobName` parameter when the job was created.

> **Note**
> If a job fails and logs do not appear in CloudWatch, it's likely that an error occurred before the start of training. Reasons include specifying the wrong training image or S3 location.

The contents of logs vary by algorithms. However, you can typically find the following information:

- Confirmation of arguments provided at the beginning of the log
- Errors that occurred during training
- Measurement of an algorithms accuracy or numerical performance
- Timings for the algorithm, and any major stages within the algorithm

Common Errors

If a training job fails, some details about the failure are provided by the `FailureReason` return value in the training job description, as follows:

```
sage = boto3.client('sagemaker')
sage.describe_training_job(TrainingJobName=job_name)['FailureReason']
```

Others are reported only in the CloudWatch logs. Common errors include the following:

1. Misspecifying a hyperparameter or specifying a hyperparameter that is invalid for the algorithm.

    From the CloudWatch Log:

    ```
    [10/16/2017 23:45:17 ERROR 139623806805824 train.py:48]
    Additional properties are not allowed (u'mini_batch_siz' was
    unexpected)
    ```

2. Specifying an invalid value for a hyperparameter.

    FailureReason:

    ```
    AlgorithmError: u'abc' is not valid under any of the given
    schemas\n\nFailed validating u'oneOf' in
    schema[u'properties'][u'feature_dim']:\n    {u'oneOf':
    [{u'pattern': u'^([1-9][0-9]*)$', u'type': u'string'},\n
    ```

```
{u'minimum': 1, u'type': u'integer'}]}\
```

FailureReason:

```
[10/16/2017 23:57:17 ERROR 140373086025536 train.py:48] u'abc'
is not valid under any of the given schemas
```

3. Inaccurate protobuf file format.

   From the CloudWatch log:

```
[10/17/2017 18:01:04 ERROR 140234860816192 train.py:48] cannot
              copy sequence with size 785 to array axis with dimension 784
```

BlazingText Algorithm

The Amazon SageMaker BlazingText algorithm provides highly optimized implementations of the Word2vec and text classification algorithms. The Word2vec algorithm is useful for many downstream natural language processing (NLP) tasks, such as sentiment analysis, named entity recognition, machine translation, etc. Text classification is an important task for applications that perform web searches, information retrieval, ranking, and document classification.

The Word2vec algorithm maps words to high-quality distributed vectors. The resulting vector representation of a word is called a *word embedding*. Words that are semantically similar correspond to vectors that are close together. That way, word embeddings capture the semantic relationships between words.

Many natural language processing (NLP) applications learn word embeddings by training on large collections of documents. These pretrained vector representations provide information about semantics and word distributions that typically improves the generalizability of other models that are later trained on a more limited amount of data. Most implementations of the Word2vec algorithm are optimized for multi-core CPU architectures. This makes it difficult to scale to large datasets.

With the BlazingText algorithm, you can scale to large datasets easily. Similar to Word2vec, it provides the Skip-gram and continuous bag-of-words (CBOW) training architectures. BlazingText's implementation of the supervised multi-class, multi-label text classification algorithm extends the fastText text classifier to use GPU acceleration with custom CUDA kernels. You can train a model on more than a billion words in a couple of minutes using a multi-core CPU or a GPU. And, you achieve performance on par with the state-of-the-art deep learning text classification algorithms.

The Amazon SageMaker BlazingText algorithms provides the following features:

- Accelerated training of the fastText text classifier on multi-core CPUs or a GPU and Word2Vec on GPUs using highly optimized CUDA kernels. For more information, see BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs.
- Enriched Word Vectors with Subword Information by learning vector representations for character n-grams. This approach enables BlazingText to generate meaningful vectors for out-of-vocabulary (OOV) words by representing their vectors as the sum of the character n-gram (subword) vectors.
- A `batch_skipgram` mode for the Word2Vec algorithm that allows faster training and distributed computation across multiple CPU nodes. The `batch_skipgram` mode does mini-batching using the Negative Sample Sharing strategy to convert level-1 BLAS operations into level-3 BLAS operations. This efficiently leverages the multiply-add instructions of modern architectures. For more information, see Parallelizing Word2Vec in Shared and Distributed Memory.

To summarize, the following modes are supported by BlazingText on different types instances:

Modes	Word2Vec (Unsupervised Learning)	Text Classification (Supervised Learning)
Single CPU instance	cbow Skip-gram Batch Skip-gram	supervised
Single GPU instance (with 1 or more GPUs)	cbow Skip-gram	supervised with one GPU
Multiple CPU instances	Batch Skip-gram	None

For more information about the mathematics behind BlazingText, see BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs.

Topics

Input/Output Interface for the BlazingText Algorithm

The BlazingText algorithm expects a single preprocessed text file with space-separated tokens. Each line in the file should contain a single sentence. If you need to train on multiple text files, concatenate them into one file and upload the file in the respective channel.

Training and Validation Data Format

Training and Validation Data Format for the Word2Vec Algorithm

For Word2Vec training, upload the file under the *train* channel. No other channels are supported. The file should contain a training sentence per line.

Training and Validation Data Format for the Text Classification Algorithm

For supervised mode, you can train with file mode or with the augmented manifest text format.

Train with File Mode

For supervised mode, the training/validation file should contain a training sentence per line along with the labels. Labels are words that are prefixed by the string *__label__*. Here is an example of a training/validation file:

```
__label__4  linux ready for prime time , intel says , despite all the linux hype , the
 open-source movement has yet to make a huge splash in the desktop market . that may be
 about to change , thanks to chipmaking giant intel corp .

__label__2  bowled by the slower one again , kolkata , november 14 the past caught up with
 sourav ganguly as the indian skippers return to international cricket was short lived .
```

Note

The order of labels within the sentence doesn't matter.

Upload the training file under the train channel, and optionally upload the validation file under the validation channel.

Train with Augmented Manifest Text Format

The supervised mode also supports the augmented manifest format, which enables you to do training in pipe mode without needing to create RecordIO files. While using the format, an S3 manifest file needs to be generated that contains the list of sentences and their corresponding labels. The manifest file format should be in JSON Lines format in which each line represents one sample. The sentences are specified using the `source` tag and the label can be specified using the `label` tag. Both `source` and `label` tags should be provided under the `AttributeNames` parameter value as specified in the request.

```
{"source":"linux ready for prime time , intel says , despite all the linux hype",
 "label":1}
{"source":"bowled by the slower one again , kolkata , november 14 the past caught up with
 sourav ganguly", "label":2}
```

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 308).

Model Artifacts and Inference

Model Artifacts for the Word2Vec Algorithm

For Word2Vec training, the model artifacts consist of *vectors.txt*, which contains words-to-vectors mapping, and *vectors.bin*, a binary used by BlazingText for hosting, inference, or both. *vectors.txt* stores the vectors in a format that is compatible with other tools like Gensim and Spacy. For example, a Gensim user can run the following commands to load the vectors.txt file:

```
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format('vectors.txt', binary=False)
word_vectors.most_similar(positive=['woman', 'king'], negative=['man'])
word_vectors.doesnt_match("breakfast cereal dinner lunch".split())
```

If the evaluation parameter is set to `True`, an additional file, *eval.json*, is created. This file contains the similarity evaluation results (using Spearman's rank correlation coefficients) on WS-353 dataset. The number of words from the WS-353 dataset that aren't there in the training corpus are reported.

For inference requests, the model accepts a JSON file containing a list of strings and returns a list of vectors. If the word is not found in vocabulary, inference returns a vector of zeros. If subwords is set to `True` during training, the model is able to generate vectors for out-of-vocabulary (OOV) words.

Sample JSON Request

Mime-type: `application/json`

```
{
"instances": ["word1", "word2", "word3"]
}
```

Model Artifacts for the Text Classification Algorithm

Training with supervised outputs creates a *model.bin* file that can be consumed by BlazingText hosting. For inference, the BlazingText model accepts a JSON file containing a list of sentences and returns a list of corresponding predicted labels and probability scores. Each sentence is expected to be a string with space-separated tokens, words, or both.

Sample JSON Request

Mime-type: `application/json`

```
{
 "instances": ["the movie was excellent", "i did not like the plot ."]
}
```

By default, the server returns only one prediction, the one with the highest probability. For retrieving the top *k* predictions, you can set *k* in the configuration, as follows:

```
{
 "instances": ["the movie was excellent", "i did not like the plot ."],
 "configuration": {"k": 2}
}
```

For BlazingText, the `content-type` and `accept` parameters must be equal. For batch transform, they both need to be `application/jsonlines`. If they differ, the `Accept` field is ignored. The format for input follows:

```
content-type: application/jsonlines

{"source": "source_0"}
{"source": "source_1"}

if you need to pass the value of k for top-k, then you can do it in the following way:

{"source": "source_0", "k": 2}
{"source": "source_1", "k": 3}
```

The format for output follows:

```
accept: application/jsonlines


{"prob": [prob_1], "label": ["__label__1"]}
{"prob": [prob_1], "label": ["__label__1"]}

If you have passed the value of k to be more than 1, then response will be in this format:

{"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
{"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
```

For both supervised (text classification) and unsupervised (Word2Vec) modes, the binaries (**.bin*) produced by BlazingText can be cross-consumed by fastText and vice versa. You can use binaries produced by BlazingText by fastText. Likewise, you can host the model binaries created with fastText using BlazingText.

For more details on dataset formats and model hosting, see the example notebooks Text Classification with the BlazingText Algorithm, FastText Models, and Generating Subword Embeddings with the Word2Vec Algorithm.

EC2 Instance Recommendation for the BlazingText Algorithm

For `cbow` and `skipgram` modes, BlazingText supports single CPU and single GPU instances. Both of these modes support learning of `subwords` embeddings. To achieve the highest speed without compromising accuracy, we recommend that you use an ml.p3.2xlarge instance.

For `batch_skipgram` mode, BlazingText supports single or multiple CPU instances. When training on multiple instances, set the value of the `S3DataDistributionType` field of the

S3DataSource (p. 994) object that you pass to CreateTrainingJob (p. 667) to
FullyReplicated. BlazingText takes care of distributing data across machines.

For the supervised text classification mode, a C5 instance is recommended if the training dataset is less than 2 GB. For larger datasets, use an instance with a single GPU (ml.p2.xlarge or ml.p3.2xlarge).

BlazingText Sample Notebooks

For a sample notebook that uses the Amazon SageMaker BlazingText algorithm to train and deploy supervised binary and multiclass classification models, see Blazing Text classification on the DBPedia dataset. For instructions for creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker examples. The topic modeling example notebooks that use the Blazing Text are located in the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

BlazingText Hyperparameters

When you start a training job with a CreateTrainingJob request, you specify a training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The hyperparameters for the BlazingText algorithm depend on which mode you use: Word2Vec (unsupervised) and Text Classification (supervised).

Word2Vec Hyperparameters

The following table lists the hyperparameters for the BlazingText Word2Vec training algorithm provided by Amazon SageMaker.

Parameter Name	Description
mode	The Word2vec architecture used for training. **Required** Valid values: batch_skipgram, skipgram, or cbow
batch_size	The size of each batch when mode is set to batch_skipgram. Set to a number between 10 and 20. **Optional** Valid values: Positive integer Default value: 11
buckets	The number of hash buckets to use for subwords. **Optional** Valid values: positive integer Default value: 2000000
epochs	The number of complete passes through the training data. **Optional** Valid values: Positive integer

Parameter Name	Description
	Default value: 5
evaluation	Whether the trained model is evaluated using the WordSimilarity-353 Test. **Optional** Valid values: (Boolean) `True` or `False` Default value: `True`
learning_rate	The step size used for parameter updates. **Optional** Valid values: Positive float Default value: 0.05
min_char	The minimum number of characters to use for subwords/character n-grams. **Optional** Valid values: positive integer Default value: 3
min_count	Words that appear less than `min_count` times are discarded. **Optional** Valid values: Non-negative integer Default value: 5
max_char	The maximum number of characters to use for subwords/character n-grams **Optional** Valid values: positive integer Default value: 6
negative_samples	The number of negative samples for the negative sample sharing strategy. **Optional** Valid values: Positive integer Default value: 5

Parameter Name	Description
sampling_threshold	The threshold for the occurrence of words. Words that appear with higher frequency in the training data are randomly down-sampled. **Optional** Valid values: Positive fraction. The recommended range is (0, 1e-3] Default value: 0.0001
subwords	Whether to learn subword embeddings on not. **Optional** Valid values: (Boolean) `True` or `False` Default value: `False`
vector_dim	The dimension of the word vectors that the algorithm learns. **Optional** Valid values: Positive integer Default value: 100
window_size	The size of the context window. The context window is the number of words surrounding the target word used for training. **Optional** Valid values: Positive integer Default value: 5

Text Classification Hyperparameters

The following table lists the hyperparameters for the Text Classification training algorithm provided by Amazon SageMaker.

Note
Although some of the parameters are common between the Text Classification and Word2Vec modes, they might have different meanings depending on the context.

Parameter Name	Description
mode	The training mode. **Required** Valid values: `supervised`
buckets	The number of hash buckets to use for word n-grams. **Optional** Valid values: Positive integer Default value: 2000000

Parameter Name	Description
`early_stopping`	Whether to stop training if validation accuracy doesn't improve after a `patience` number of epochs. **Optional** Valid values: (Boolean) `True` or `False` Default value: `False`
`epochs`	The maximum number of complete passes through the training data. **Optional** Valid values: Positive integer Default value: 5
`learning_rate`	The step size used for parameter updates. **Optional** Valid values: Positive float Default value: 0.05
`min_count`	Words that appear less than `min_count` times are discarded. **Optional** Valid values: Non-negative integer Default value: 5
`min_epochs`	The minimum number of epochs to train before early stopping logic is invoked. **Optional** Valid values: Positive integer Default value: 5
`patience`	The number of epochs to wait before applying early stopping when no progress is made on the validation set. Used only when `early_stopping` is `True`. **Optional** Valid values: Positive integer Default value: 4

Parameter Name	Description
`vector_dim`	The dimension of the embedding layer. **Optional** Valid values: Positive integer Default value: 100
`word_ngrams`	The number of word n-gram features to use. **Optional** Valid values: Positive integer Default value: 2

Tune a BlazingText Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the BlazingText Algorithm

The BlazingText Word2Vec algorithm (`skipgram`, `cbow`, and `batch_skipgram` modes) reports on a single metric during training: `train:mean_rho`. This metric is computed on WS-353 word similarity datasets. When tuning the hyperparameter values for the Word2Vec algorithm, use this metric as the objective.

The BlazingText Text Classification algorithm (`supervised` mode), also reports on a single metric during training: the `validation:accuracy`. When tuning the hyperparameter values for the text classification algorithm, use these metrics as the objective.

Metric Name	Description	Optimization Direction
`train:mean_rho`	The mean rho (Spearman's rank correlation coefficient) on WS-353 word similarity datasets	Maximize
`validation:accuracy`	The classification accuracy on the user-specified validation dataset	Maximize

Tunable BlazingText Hyperparameters

Tunable Hyperparameters for the Word2Vec Algorithm

Tune an Amazon SageMaker BlazingText Word2Vec model with the following hyperparameters. The hyperparameters that have the greatest impact on Word2Vec objective metrics are: `mode`, `learning_rate`, `window_size`, `vector_dim`, and `negative_samples`.

Parameter Name	Parameter Type	Recommended Ranges or Values
batch_size	IntegerParameterRange	[8-32]
epochs	IntegerParameterRange	[5-15]
learning_rate	ContinuousParameterRange	MinValue: 0.005, MaxValue: 0.01
min_count	IntegerParameterRange	[0-100]
mode	CategoricalParameterRange	['batch_skipgram', 'skipgram', 'cbow']
negative_samples	IntegerParameterRange	[5-25]
sampling_threshold	ContinuousParameterRange	MinValue: 0.0001, MaxValue: 0.001
vector_dim	IntegerParameterRange	[32-300]
window_size	IntegerParameterRange	[1-10]

Tunable Hyperparameters for the Text Classification Algorithm

Tune an Amazon SageMaker BlazingText text classification model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges or Values
buckets	IntegerParameterRange	[1000000-10000000]
epochs	IntegerParameterRange	[5-15]
learning_rate	ContinuousParameterRange	MinValue: 0.005, MaxValue: 0.01
min_count	IntegerParameterRange	[0-100]
mode	CategoricalParameterRange	['supervised']
vector_dim	IntegerParameterRange	[32-300]
word_ngrams	IntegerParameterRange	[1-3]

DeepAR Forecasting Algorithm

The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN). Classical forecasting methods, such as autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS), fit a single model to each individual time series. They then use that model to extrapolate the time series into the future.

In many applications, however, you have many similar time series across a set of cross-sectional units. For example, you might have time series groupings for demand for different products, server loads, and requests for webpages. For this type of application, you can benefit from training a single model jointly

over all of the time series. DeepAR takes this approach. When your dataset contains hundreds of related time series, DeepAR outperforms the standard ARIMA and ETS methods. You can also use the trained model to generate forecasts for new time series that are similar to the ones it has been trained on.

The training input for the DeepAR algorithm is one or, preferably, more `target` time series that have been generated by the same process or similar processes. Based on this input dataset, the algorithm trains a model that learns an approximation of this process/processes and uses it to predict how the target time series evolves. Each target time series can be optionally associated with a vector of static (time-independent) categorical features provided by the `cat` field and a vector of dynamic (time-dependent) time series provided by the `dynamic_feat` field. Amazon SageMaker trains the DeepAR model by randomly sampling training examples from each target time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. To control how far in the past the network can see, use the `context_length` hyperparameter. To control how far in the future predictions can be made, use the `prediction_length` hyperparameter. For more information, see How the DeepAR Algorithm Works (p. 87).

Topics

- Input/Output Interface for the DeepAR Algorithm (p. 84)
- Best Practices for Using the DeepAR Algorithm (p. 86)
- EC2 Instance Recommendations for the DeepAR Algorithm (p. 87)
- DeepAR Sample Notebooks (p. 87)
- How the DeepAR Algorithm Works (p. 87)
- DeepAR Hyperparameters (p. 90)
- Tune a DeepAR Model (p. 94)
- DeepAR Inference Formats (p. 95)

Input/Output Interface for the DeepAR Algorithm

DeepAR supports two data channels. The required `train` channel describes the training dataset. The optional `test` channel describes a dataset that the algorithm uses to evaluate model accuracy after training. You can provide training and test datasets in JSON Lines format. Files can be in gzip or Parquet file format.

When specifying the paths for the training and test data, you can specify a single file or a directory that contains multiple files, which can be stored in subdirectories. If you specify a directory, DeepAR uses all files in the directory as inputs for the corresponding channel, except those that start with a period (.) and those named *_SUCCESS*. This ensures that you can directly use output folders produced by Spark jobs as input channels for your DeepAR training jobs.

By default, the DeepAR model determines the input format from the file extension (`.json`, `.json.gz`, or `.parquet`) in the specified input path. If the path does not end in one of these extensions, you must explicitly specify the format in the SDK for Python. Use the `content_type` parameter of the s3_input class.

The records in your input files should contain the following fields:

- `start`—A string with the format `YYYY-MM-DD HH:MM:SS`. The start timestamp can't contain time zone information.
- `target`—An array of floating-point values or integers that represent the time series. You can encode missing values as `null` literals, or as `"NaN"` strings in JSON, or as `nan` floating-point values in Parquet.
- `dynamic_feat` (optional)—An array of arrays of floating-point values or integers that represents the vector of custom feature time series (dynamic features). If you set this field, all records must have the same number of inner arrays (the same number of feature time series). In addition, each inner array must have the same length as the associated `target` value. Missing values are not supported in the features. For example, if target time series represents the demand of different products, an associated

dynamic_feat might be a boolean time-series which indicates whether a promotion was applied (1) to the particular product or not (0):

```
{"start": ..., "target": [1, 5, 10, 2], "dynamic_feat": [[0, 1, 1, 0]]}
```

- cat (optional)—An array of categorical features that can be used to encode the groups that the record belongs to. Categorical features must be encoded as a 0-based sequence of positive integers. For example, the categorical domain {R, G, B} can be encoded as {0, 1, 2}. All values from each categorical domain must be represented in the training dataset. That's because the DeepAR algorithm can forecast only for categories that have been observed during training. And, each categorical feature is embedded in a low-dimensional space whose dimensionality is controlled by the embedding_dimension hyperparameter. For more information, see DeepAR Hyperparameters (p. 90).

If you use a JSON file, it must be in JSON Lines format. For example:

```
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1],
 "dynamic_feat": [[1.1, 1.2, 0.5, ...]]}
{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat":
 [[1.1, 2.05, ...]]}
{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat":
 [[1.3, 0.4]]}
```

In this example, each time series has two associated categorical features and one time series features.

For Parquet, you use the same three fields as columns. In addition, "start" can be the datetime type. You can compress Parquet files using gzip (gzip) or the Snappy compression library (snappy).

If the algorithm is trained without cat and dynamic_feat fields, it learns a "global" model, that is a model that is agnostic to the specific identity of the target time series at inference time and is conditioned only on its shape.

If the model is conditioned on the cat and dynamic_feat feature data provided for each time series, the prediction will probably be influenced by the character of time series with the corresponding cat features. For example, if the target time series represents the demand of clothing items, you can associate a two-dimensional cat vector that encodes the type of item (e.g. 0 = shoes, 1 = dress) in the first component and the color of an item (e.g. 0 = red, 1 = blue) in the second component. A sample input would look as follows:

```
{ "start": ..., "target": ..., "cat": [0, 0], ... } # red shoes
{ "start": ..., "target": ..., "cat": [1, 1], ... } # blue dress
```

At inference time, you can request predictions for targets with cat values that are combinations of the cat values observed in the training data, for example:

```
{ "start": ..., "target": ..., "cat": [0, 1], ... } # red dress
{ "start": ..., "target": ..., "cat": [1, 1], ... } # blue dress
```

The following guidelines apply to training data:

- The start time and length of the time series can differ. For example, in marketing, products often enter a retail catalog at different dates, so their start dates naturally differ. But all series must have the same frequency, number of categorical features, and number of dynamic features.
- Shuffle the training file with respect to the position of the time series in the file. In other words, the time series should occur in random order in the file.
- Make sure to set the start field correctly. The algorithm uses the start timestamp to derive the internal features.

- If you use categorical features (`cat`), all time series must have the same number of categorical features. If the dataset contains the `cat` field, the algorithm uses it and extracts the cardinality of the groups from the dataset. By default, `cardinality` is `"auto"`. If the dataset contains the `cat` field, but you don't want to use it, you can disable it by setting `cardinality` to `""`. If a model was trained using a `cat` feature, you must include it for inference.

- If your dataset contains the `dynamic_feat` field, the algorithm uses it automatically. All time series have to have the same number of feature time series. The time points in each of the feature time series correspond one-to-one to the time points in the target. In addition, the entry in the `dynamic_feat` field should have the same length as the `target`. If the dataset contains the `dynamic_feat` field, but you don't want to use it, disable it by setting(`num_dynamic_feat` to `""`). If the model was trained with the `dynamic_feat` field, you must provide this field for inference. In addition, each of the features has to have the length of the provided target plus the `prediction_length`. In other words, you must provide the feature value in the future.

If you specify optional test channel data, the DeepAR algorithm evaluates the trained model with different accuracy metrics. The algorithm calculates the root mean square error (RMSE) over the test data as follows:

$$\text{RMSE} = \sqrt{\frac{1}{nT} \sum_{i,t} \left(\hat{y}_{i,t} - y_{i,t}\right)^2}$$

$y_{i,t}$ is the true value of time series i at the time t. $\hat{y}_{i,t}$ is the mean prediction. The sum is over all n time series in the test set and over the last T time points for each time series, where T corresponds to the forecast horizon. You specify the length of the forecast horizon by setting the `prediction_length` hyperparameter. For more information, see DeepAR Hyperparameters (p. 90).

In addition, the algorithm evaluates the accuracy of the forecast distribution using weighted quantile loss. For a quantile in the range [0, 1], the weighted quantile loss is defined as follows:

$$\text{wQuantileLoss}[\tau] = 2\frac{\sum_{i,t} Q_{i,t}^{(\tau)}}{\sum_{i,t} |y_{i,t}|}, \quad \text{with} \quad Q_{i,t}^{(\tau)} = \begin{cases} (1-\tau)|q_{i,t}^{(\tau)} - y_{i,t}| & \text{if } q_{i,t}^{(\tau)} > y_{i,t} \\ \tau|q_{i,t}^{(\tau)} - y_{i,t}| & \text{otherwise} \end{cases}$$

$q_{i,t}^{(\tau)}$ is the τ-quantile of the distribution that the model predicts. To specify which quantiles to calculate loss for, set the `test_quantiles` hyperparameter. In addition to these, the average of the prescribed quantile losses is reported as part of the training logs. For information, see DeepAR Hyperparameters (p. 90).

For inference, DeepAR accepts JSON format and the following fields:

- `"instances"`, which includes one or more time series in JSON Lines format
- A name of `"configuration"`, which includes parameters for generating the forecast

For more information, see DeepAR Inference Formats (p. 95).

Best Practices for Using the DeepAR Algorithm

When preparing your time series data, follow these best practices to achieve the best results:

- Except for when splitting your dataset for training and testing, always provide the entire time series for training, testing, and when calling the model for inference. Regardless of how you set `context_length`, don't break up the time series or provide only a part of it. The model uses data points further back than the value set in `context_length` for the lagged values feature.

- When tuning a DeepAR model, you can split the dataset to create a training dataset and a test dataset. In a typical evaluation, you would test the model on the same time series used for training, but on the future `prediction_length` time points that follow immediately after the last time point visible during training. You can create training and test datasets that satisfy this criteria by using the entire dataset (the full length of all time series that are available) as a test set and removing the last `prediction_length` points from each time series for training. During training, the model doesn't see the target values for time points on which it is evaluated during testing. During testing, the algorithm withholds the last `prediction_length` points of each time series in the test set and generates a prediction. Then it compares the forecast with the withheld values. You can create more complex evaluations by repeating time series multiple times in the test set, but cutting them at different endpoints. With this approach, accuracy metrics are averaged over multiple forecasts from different time points. For more information, see Tune a DeepAR Model (p. 94).

- Avoid using very large values (>400) for the `prediction_length` because it makes the model slow and less accurate. If you want to forecast further into the future, consider aggregating your data at a higher frequency. For example, use `5min` instead of `1min`.

- Because lags are used, a model can look further back in the time series than the value specified for `context_length`. Therefore, you don't need to set this parameter to a large value. We recommend starting with the value that you used for `prediction_length`.

- We recommend training a DeepAR model on as many time series as are available. Although a DeepAR model trained on a single time series might work well, standard forecasting algorithms, such as ARIMA or ETS, might provide more accurate results. The DeepAR algorithm starts to outperform the standard methods when your dataset contains hundreds of related time series. Currently, DeepAR requires that the total number of observations available across all training time series is at least 300.

EC2 Instance Recommendations for the DeepAR Algorithm

You can train DeepAR on both GPU and CPU instances and in both single and multi-machine settings. We recommend starting with a single CPU instance (for example, ml.c4.2xlarge or ml.c4.4xlarge), and switching to GPU instances and multiple machines only when necessary. Using GPUs and multiple machines improves throughput only for larger models (with many cells per layer and many layers) and for large mini-batch sizes (for example, greater than 512).

For inference, DeepAR supports only CPU instances.

Specifying large values for `context_length`, `prediction_length`, `num_cells`, `num_layers`, or `mini_batch_size` can create models that are too large for small instances. In this case, use a larger instance type or reduce the values for these parameters. This problem also frequently occurs when running hyperparameter tuning jobs. In that case, use an instance type large enough for the model tuning job and consider limiting the upper values of the critical parameters to avoid job failures.

DeepAR Sample Notebooks

For a sample notebook that shows how to prepare a time series dataset for training the Amazon SageMaker DeepAR algorithm and how to deploy the trained model for performing inferences, see Time series forecasting with DeepAR - Synthetic data as well as DeepAR demo on electricity dataset, which illustrates the advanced features of DeepAR on a real world dataset. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all of the Amazon SageMaker examples. To open a notebook, choose its **Use** tab, and choose **Create copy**.

How the DeepAR Algorithm Works

During training, DeepAR accepts a training dataset and an optional test dataset. It uses the test dataset to evaluate the trained model. In general, the datasets don't have to contain the same set of time series.

You can use a model trained on a given training set to generate forecasts for the future of the time series in the training set, and for other time series. Both the training and the test datasets consist of one or, preferably, more target time series. Each target time series can optionally be associated with a vector of feature time series and a vector of categorical features. For more information, see Input/Output Interface for the DeepAR Algorithm (p. 84).

For example, the following is an element of a training set indexed by i which consists of a target time series, $Z_{i,t}$, and two associated feature time series, $X_{i,1,t}$ and $X_{i,2,t}$:
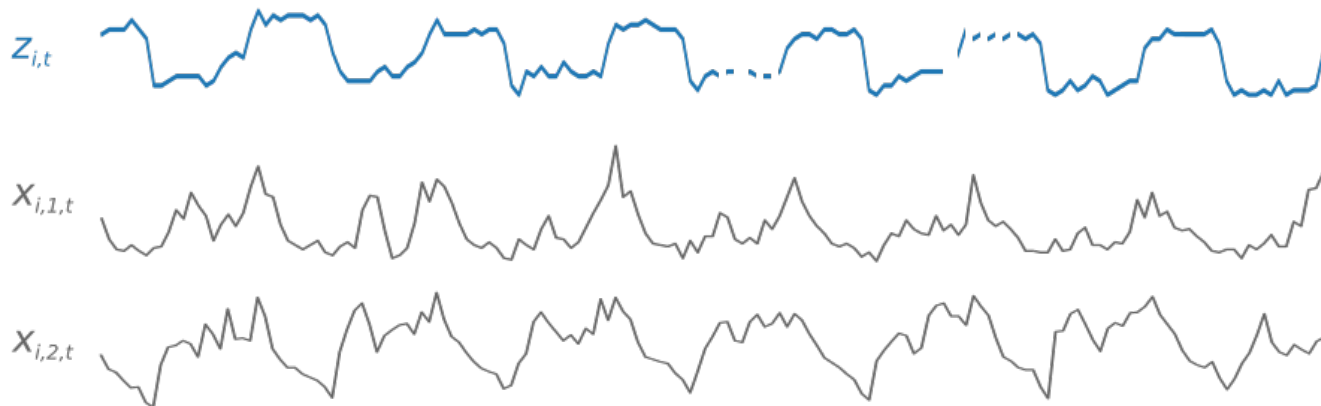
The target time series might contain missing values, which are represented by line breaks in the time series. DeepAR supports only feature time series that are known in the future. This allows you to run "what if?" scenarios. What happens, for example, if I change the price of a product in some way?

Each target time series can also be associated with a number of categorical features. You can use these features to encode which groupings a time series belongs to. Categorical features allow the model to learn typical behavior for groups, which it can use to increase model accuracy. DeepAR implements this by learning an embedding vector for each group that captures the common properties of all time series in the group.

How Feature Time Series Work in the DeepAR Algorithm

To facilitate learning time-dependent patterns, such as spikes during weekends, DeepAR automatically creates feature time series based on the frequency of the target time series. For example, DeepAR creates two feature time series (day of the month and day of the year) for a weekly time series frequency. It uses these derived feature time series with the custom feature time series that you provide during training and inference. The following figure shows two of these derived time series features: $u_{i,1,t}$ represents the hour of the day and $u_{i,2,t}$ the day of the week.
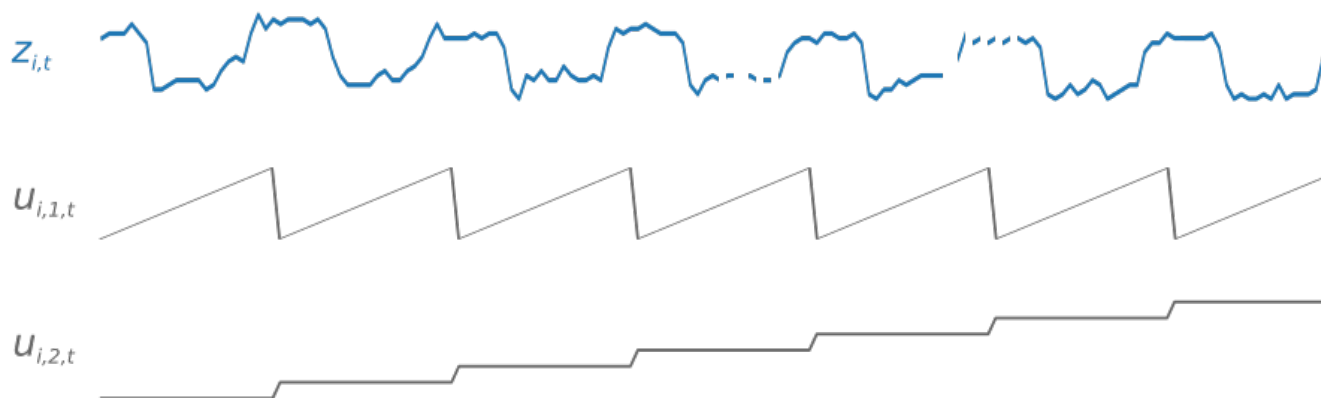
The DeepAR algorithm automatically generates these feature time series. The following table lists the derived features for the supported basic time frequencies.

Frequency of the Time Series	Derived Features
Minute	`minute-of-hour, hour-of-day, day-of-week, day-of-month, day-of-year`
Hour	`hour-of-day, day-of-week, day-of-month, day-of-year`
Day	`day-of-week, day-of-month, day-of-year`
Week	`day-of-month, week-of-year`
Month	`month-of-year`

DeepAR trains a model by randomly sampling several training examples from each of the time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. The `context_length` hyperparameter controls how far in the past the network can see, and the `prediction_length` hyperparameter controls how far in the future predictions can be made. During training, the algorithm ignores training set elements containing time series that are shorter than a specified prediction length. The following figure represents five samples with context lengths of 12 hours and prediction lengths of 6 hours drawn from element i. For brevity, we've omitted the feature time series $x_{i,1,t}$ and $u_{i,2,t}$.
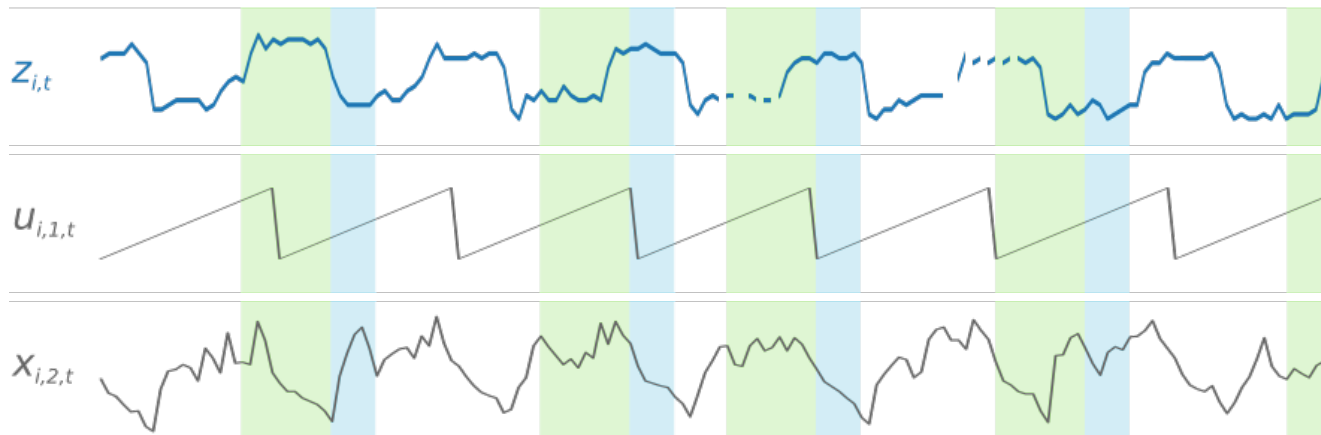
To capture seasonality patterns, DeepAR also automatically feeds lagged values from the target time series. In the example with hourly frequency, for each time index, $t = T$, the model exposes the $z_{i,t}$ values, which occurred approximately one, two, and three days in the past.
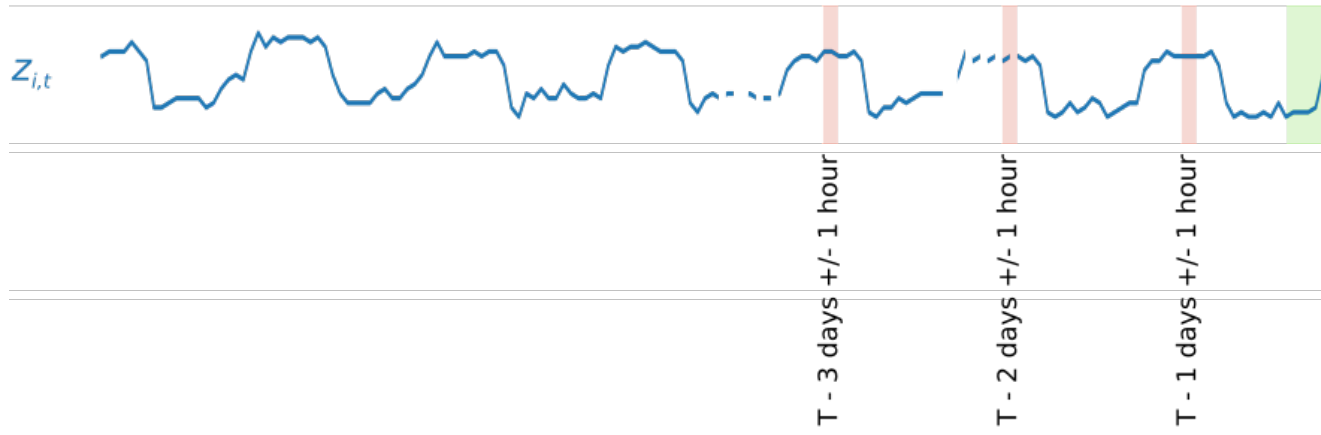
For inference, the trained model takes as input target time series, which might or might not have been used during training, and forecasts a probability distribution for the next `prediction_length` values. Because DeepAR is trained on the entire dataset, the forecast takes into account patterns learned from similar time series.

For information on the mathematics behind DeepAR, see DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks.

DeepAR Hyperparameters

Parameter Name	Description
context_length	The number of time-points that the model gets to see before making the prediction. The value for this parameter should be about the same as the `prediction_length`. The model also receives lagged inputs from the target, so `context_length` can be much smaller than typical seasonalities. For example, a daily time series can have yearly seasonality. The model automatically includes a lag of one year, so the context length can be shorter than a year. The lag values that the model picks depend on the frequency of the time series. For example, lag values for daily frequency are previous week, 2 weeks, 3 weeks, 4 weeks, and year. **Required** Valid values: Positive integer
epochs	The maximum number of passes over the training data. The optimal value depends on your data size and learning rate. See also `early_stopping_patience`. Typical values range from 10 to 1000. **Required** Valid values: Positive integer
prediction_length	The number of time-steps that the model is trained to predict, also called the forecast horizon. The trained model always generates forecasts with this length. It can't generate longer forecasts. The `prediction_length` is fixed when a model is trained and it cannot be changed later.

Parameter Name	Description
	Required Valid values: Positive integer
time_freq	The granularity of the time series in the dataset. Use `time_freq` to select appropriate date features and lags. The model supports the following basic frequencies. It also supports multiples of these basic frequencies. For example, `5min` specifies a frequency of 5 minutes. • *M*: monthly • *W*: weekly • *D*: daily • *H*: hourly • *min*: every minute **Required** Valid values: An integer followed by *M*, *W*, *D*, *H*, or *min*. For example, `5min`.
cardinality	When using the categorical features (`cat`), `cardinality` is an array specifying the number of categories (groups) per categorical feature. Set this to `auto` to infer the cardinality from the data. The `auto` mode also works when no categorical features are used in the dataset. This is the recommended setting for the parameter. Set cardinality to `ignore` to force DeepAR to not use categorical features, even it they are present in the data. To perform additional data validation, it is possible to explicitly set this parameter to the actual value. For example, if two categorical features are provided where the first has 2 and the other has 3 possible values, set this to [2, 3]. For more information on how to use categorical feature, see the data-section on the main documentation page of DeepAR. **Optional** Valid values: `auto`, `ignore`, array of positive integers, empty string, or Default value: `auto`
dropout_rate	The dropout rate to use during training. The model uses zoneout regularization. For each iteration, a random subset of hidden neurons are not updated. Typical values are less than 0.2. **Optional** Valid values: float Default value: 0.1

Parameter Name	Description
early_stopping_patience	If this parameter is set, training stops when no progress is made within the specified number of `epochs`. The model that has the lowest loss is returned as the final model. **Optional** Valid values: integer
embedding_dimension	Size of embedding vector learned per categorical feature (same value is used for all categorical features). The DeepAR model can learn group-level time series patterns when a categorical grouping feature is provided. To do this, the model learns an embedding vector of size `embedding_dimension` for each group, capturing the common properties of all time series in the group. A larger `embedding_dimension` allows the model to capture more complex patterns. However, because increasing the `embedding_dimension` increases the number of parameters in the model, more training data is required to accurately learn these parameters. Typical values for this parameter are between 10-100. **Optional** Valid values: positive integer Default value: 10
learning_rate	The learning rate used in training. Typical values range from 1e-4 to 1e-1. **Optional** Valid values: float Default value: 1e-3

Parameter Name	Description
`likelihood`	The model generates a probabilistic forecast, and can provide quantiles of the distribution and return samples. Depending on your data, select an appropriate likelihood (noise model) that is used for uncertainty estimates. The following likelihoods can be selected: • *gaussian*: Use for real-valued data. • *beta*: Use for real-valued targets between 0 and 1 inclusive. • *negative-binomial*: Use for count data (non-negative integers). • *student-T*: An alternative for real-valued data that works well for bursty data. • *deterministic-L1*: A loss function that does not estimate uncertainty and only learns a point forecast. **Optional** Valid values: One of *gaussian*, *beta*, *negative-binomial*, *student-T*, or *deterministic-L1*. Default value: `student-T`
`mini_batch_size`	The size of mini-batches used during training. Typical values range from 32 to 512. **Optional** Valid values: positive integer Default value: 128
`num_cells`	The number of cells to use in each hidden layer of the RNN. Typical values range from 30 to 100. **Optional** Valid values: positive integer Default value: 40
`num_dynamic_feat`	The number of `dynamic_feat` provided in the data. Set this to `auto` to infer the number of dynamic features from the data. The `auto` mode also works when no dynamic features are used in the dataset. This is the recommended setting for the parameter. To force DeepAR to not use dynamic features, even it they are present in the data, set `num_dynamic_feat` to `ignore`. To perform additional data validation, it is possible to explicitly set this parameter to the actual integer value. For example, if two dynamic features are provided, set this to 2. **Optional** Valid values: `auto`, `ignore`, positive integer, or empty string Default value: `auto`

Parameter Name	Description
num_eval_samples	The number of samples that are used per time-series when calculating test accuracy metrics. This parameter does not have any influence on the training or the final model. In particular, the model can be queried with a different number of samples. This parameter only affects the reported accuracy scores on the test channel after training. Smaller values result in faster evaluation, but then the evaluation scores are typically worse and more uncertain. When evaluating with higher quantiles, for example 0.95, it may be important to increase the number of evaluation samples. **Optional** Valid values: integer Default value: 100
num_layers	The number of hidden layers in the RNN. Typical values range from 1 to 4. **Optional** Valid values: positive integer Default value: 2
test_quantiles	Quantiles for which to calculate quantile loss on the test channel. **Optional** Valid values: array of floats Default value: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

Tune a DeepAR Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the DeepAR Algorithm

The DeepAR algorithm reports three metrics, which are computed during training. When tuning a model, choose one of these as the objective. For the objective, use either the forecast accuracy on a provided test channel (recommended) or the training loss. For recommendations for the training/test split for the DeepAR algorithm, see Best Practices for Using the DeepAR Algorithm (p. 86).

Metric Name	Description	Optimization Direction
test:RMSE	The root mean square error between the forecast and the actual target computed on the test set.	Minimize

Metric Name	Description	Optimization Direction
`test:mean_wQuantileLoss`	The average overall quantile losses computed on the test set. To control which quantiles are used, set the `test_quantiles` hyperparameter.	Minimize
`train:final_loss`	The training negative log-likelihood loss averaged over the last training epoch for the model.	Minimize

Tunable Hyperparameters for the DeepAR Algorithm

Tune a DeepAR model with the following hyperparameters. The hyperparameters that have the greatest impact, listed in order from the most to least impactful, on DeepAR objective metrics are: `epochs`, `context_length`, `mini_batch_size`, `learning_rate`, and `num_cells`.

Parameter Name	Parameter Type	Recommended Ranges
`mini_batch_size`	IntegerParameterRanges	MinValue: 32, MaxValue: 1028
`epochs`	IntegerParameterRanges	MinValue: 1, MaxValue: 1000
`context_length`	IntegerParameterRanges	MinValue: 1, MaxValue: 200
`num_cells`	IntegerParameterRanges	MinValue: 30, MaxValue: 200
`num_layers`	IntegerParameterRanges	MinValue: 1, MaxValue: 8
`dropout_rate`	ContinuousParameterRange	MinValue: 0.00, MaxValue: 0.2
`embedding_dimension`	IntegerParameterRanges	MinValue: 1, MaxValue: 50
`learning_rate`	ContinuousParameterRange	MinValue: 1e-5, MaxValue: 1e-1

DeepAR Inference Formats

DeepAR JSON Request Formats

Query a trained model by using the model's endpoint. The endpoint takes the following JSON request format.

In the request, the `instances` field corresponds to the time series that should be forecast by the model.

If the model was trained with categories, you must provide a `cat` for each instance. If the model was trained without the `cat` field, it should be omitted.

If the model was trained with a custom feature time series (`dynamic_feat`), you have to provide the same number of `dynamic_feat` values for each instance. Each of them should have a length given by `length(target) + prediction_length`, where the last `prediction_length` values correspond to

the time points in the future that will be predicted. If the model was trained without custom feature time series, the field should not be included in the request.

```
{
    "instances": [
        {
            "start": "2009-11-01 00:00:00",
            "target": [4.0, 10.0, "NaN", 100.0, 113.0],
            "cat": [0, 1],
            "dynamic_feat": [[1.0, 1.1, 2.1, 0.5, 3.1, 4.1, 1.2, 5.0, ...]]
        },
        {
            "start": "2012-01-30",
            "target": [1.0],
            "cat": [2, 1],
            "dynamic_feat": [[2.0, 3.1, 4.5, 1.5, 1.8, 3.2, 0.1, 3.0, ...]]
        },
        {
            "start": "1999-01-30",
            "target": [2.0, 1.0],
            "cat": [1, 3],
            "dynamic_feat": [[1.0, 0.1, -2.5, 0.3, 2.0, -1.2, -0.1, -3.0, ...]]
        }
    ],
    "configuration": {
        "num_samples": 50,
        "output_types": ["mean", "quantiles", "samples"],
        "quantiles": ["0.5", "0.9"]
    }
}
```

The `configuration` field is optional. `configuration.num_samples` sets the number of sample paths that the model generates to estimate the mean and quantiles. `configuration.output_types` describes the information that will be returned in the request. Valid values are `"mean"` `"quantiles"` and `"samples"`. If you specify `"quantiles"`, each of the quantile values in `configuration.quantiles` is returned as a time series. If you specify `"samples"`, the model also returns the raw samples used to calculate the other outputs.

DeepAR JSON Response Formats

The following is the format of a response, where `[...]` are arrays of numbers:

```
{
    "predictions": [
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]
        },
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]
        },
        {
            "quantiles": {
                "0.9": [...],
```

```
            "0.5": [...]
        },
        "samples": [...],
        "mean": [...]
    }
  ]
}
```

DeepAR has a response timeout of 60 seconds. When passing multiple time series in a single request, the forecasts are generated sequentially. Because the forecast for each time series typically takes about 300 to 1000 milliseconds or longer, depending on the model size, passing too many time series in a single request can cause time outs. It's better to send fewer time series per request and send more requests. Because the DeepAR algorithm uses multiple workers per instance, you can achieve much higher throughput by sending multiple requests in parallel.

By default, DeepAR uses one worker per CPU for inference, if there is sufficient memory per CPU. If the model is large and there isn't enough memory to run a model on each CPU, the number of workers is reduced. The number of workers used for inference can be overwritten using the environment variable `MODEL_SERVER_WORKERS` For example, by setting `MODEL_SERVER_WORKERS=1`) when calling the Amazon SageMaker CreateModel (p. 648) API.

Batch Transform with the DeepAR Algorithm

DeepAR forecasting supports getting inferences by using batch transform from data using the JSON Lines format. In this format, each record is represented on a single line as a JSON object, and lines are separated by newline characters. The format is identical to the JSON Lines format used for model training. For information, see Input/Output Interface for the DeepAR Algorithm (p. 84). For example:

```
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1],
 "dynamic_feat": [[1.1, 1.2, 0.5, ..]]}
{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat":
 [[1.1, 2.05, ...]]}
{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat":
 [[1.3, 0.4]]}
```

> **Note**
> When creating the transformation job with CreateTransformJob (p. 673), set the `BatchStrategy` value to `SingleRecord` and set the `SplitType` value in the TransformInput (p. 1024) configuration to `Line`, as the default values currently cause runtime failures.

Similar to the hosted endpoint inference request format, the `cat` and the `dynamic_feat` fields for each instance are required if both of the following are true:

- The model is trained on a dataset that contained both the `cat` and the `dynamic_feat` fields.
- The corresponding `cardinality` and `num_dynamic_feat` values used in the training job are not set to `""`.

Unlike hosted endpoint inference, the configuration field is set once for the entire batch inference job using an environment variable named `DEEPAR_INFERENCE_CONFIG`. The value of `DEEPAR_INFERENCE_CONFIG` can be passed when the model is created by calling CreateTransformJob (p. 673) API. If `DEEPAR_INFERENCE_CONFIG` is missing in the container environment, the inference container uses the following default:

```
{
    "num_samples": 100,
    "output_types": ["mean", "quantiles"],
```

```
    "quantiles": ["0.1", "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9"]
}
```

The output is also in JSON Lines format, with one line per prediction, in an order identical to the instance order in the corresponding input file. Predictions are encoded as objects identical to the ones returned by responses in online inference mode. For example:

```
{ "quantiles": { "0.1": [...], "0.2": [...] }, "samples": [...], "mean": [...] }
```

Note that in the TransformInput (p. 1024) configuration of the Amazon SageMaker CreateTransformJob (p. 673) request clients must explicitly set the `AssembleWith` value to `Line`, as the default value `None` concatenates all JSON objects on the same line.

For example, here is an Amazon SageMaker CreateTransformJob (p. 673) request for a DeepAR job with a custom `DEEPAR_INFERENCE_CONFIG`:

```
{
    "BatchStrategy": "SingleRecord",
    "Environment": {
        "DEEPAR_INFERENCE_CONFIG" : "{ \"num_samples\": 200, \"output_types\": [\"mean\"] }",
        ...
    },
    "TransformInput": {
        "SplitType": "Line",
        ...
    },
    "TransformOutput": {
        "AssembleWith": "Line",
        ...
    },
    ...
}
```

Factorization Machines Algorithm

A factorization machine is a general-purpose supervised learning algorithm that you can use for both classification and regression tasks. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets economically. For example, in a click prediction system, the factorization machine model can capture click rate patterns observed when ads from a certain ad-category are placed on pages from a certain page-category. Factorization machines are a good choice for tasks dealing with high dimensional sparse datasets, such as click prediction and item recommendation.

> **Note**
> The Amazon SageMaker implementation of factorization machines considers only pair-wise (2nd order) interactions between features.

Topics

Input/Output Interface for the Factorization Machines Algorithm

The factorization machine algorithm can be run in either in binary classification mode or regression mode. In each mode, a dataset can be provided to the **test** channel along with the train channel dataset. The scoring depends on the mode used. In regression mode, the testing dataset is scored using Root Mean Square Error (RMSE). In binary classification mode, the test dataset is scored using Binary Cross Entropy (Log Loss), Accuracy (at threshold=0.5) and F1 Score (at threshold =0.5).

For **training**, the factorization machines algorithm currently supports only the `recordIO-protobuf` format with `Float32` tensors. Because their use case is predominantly on sparse data, `CSV` is not a good candidate. Both File and Pipe mode training are supported for recordIO-wrapped protobuf.

For **inference**, factorization machines support the `application/json` and `x-recordio-protobuf` formats.

- For the **binary classification** problem, the algorithm predicts a score and a label. The label is a number and can be either `0` or `1`. The score is a number that indicates how strongly the algorithm believes that the label should be `1`. The algorithm computes score first and then derives the label from the score value. If the score is greater than or equal to 0.5, the label is `1`.
- For the **regression** problem, just a score is returned and it is the predicted value. For example, if Factorization Machines is used to predict a movie rating, score is the predicted rating value.

Please see for more details on training and inference file formats.

EC2 Instance Recommendation for the Factorization Machines Algorithm

The Amazon SageMaker Factorization Machines algorithm is highly scalable and can train across distributed instances. We recommend training and inference with CPU instances for both sparse and dense datasets. In some circumstances, training with one or more GPUs on dense data might provide some benefit. Training with GPUs is available only on dense data. Use CPU instances for sparse data.

Factorization Machines Sample Notebooks

For a sample notebook that uses the Amazon SageMaker factorization machine learning algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to Factorization Machines with MNIST. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see . Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Factorization Machines Work

The prediction task for a factorization machine model is to estimate a function $\hat{y}$ from a feature set x_i to a target domain. This domain is real-valued for regression and binary for classification. The factorization machine model is supervised and so has a training dataset (x_i, y_j) available. The advantages this model presents lie in the way it uses a factorized parametrization to capture the pairwise feature interactions. It can be represented mathematically as follows:

$$\hat{y} = w_0 + \sum_i w_i x_i + \sum_i \sum_{j>i} <v_i, v_j> x_i x_j$$

The three terms in this equation correspond respectively to the three components of the model:

- The w_0 term represents the global bias.
- The w_i linear terms model the strength of the ith variable.
- The $<v_i,v_j>$ factorization terms model the pairwise interaction between the ith and j^{th} variable.

The global bias and linear terms are the same as in a linear model. The pairwise feature interactions are modeled in the third term as the inner product of the corresponding factors learned for each feature. Learned factors can also be considered as embedding vectors for each feature. For example, in a classification task, if a pair of features tends to co-occur more often in positive labeled samples, then the inner product of their factors would be large. In other words, their embedding vectors would be close to each other in cosine similarity. For more information about the factorization machine model, see Factorization Machines.

For regression tasks, the model is trained by minimizing the squared error between the model prediction $\hat{y}_n$ and the target value y_n. This is known as the square loss:

$$L = \frac{1}{N} \sum_n (y_n - \hat{y}_n)^2$$

For a classification task, the model is trained by minimizing the cross entropy loss, also known as the log loss:

$$L = \frac{1}{N} \sum_n [y_n \log \hat{p}_n + (1 - y_n) \log (1 - \hat{p}_n)]$$

where:

$$\hat{p}_n = \frac{1}{1 + e^{-\hat{y}_n}}$$

For more information about loss functions for classification, see Loss functions for classification.

Factorization Machines Hyperparameters

The following table contains the hyperparameters for the factorization machines algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order.

Parameter Name	Description
`feature_dim`	The dimension of the input feature space. This could be very high with sparse input. **Required** Valid values: Positive integer. Suggested value range: [10000,10000000]
`num_factors`	The dimensionality of factorization. **Required** Valid values: Positive integer. Suggested value range: [2,1000], 64 is usually optimal.

Parameter Name	Description
`predictor_type`	The type of predictor. • `binary_classifier`: For binary classification tasks. • `regressor`: For regression tasks. **Required** Valid values: String: `binary_classifier` or `regressor`
`bias_init_method`	The initialization method for the bias term: • `normal`: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by `bias_init_sigma`. • `uniform`: Initializes weights with random values uniformly sampled from a range specified by [-`bias_init_scale`, +`bias_init_scale`]. • `constant`: Initializes the weights to a scalar value specified by `bias_init_value`. **Optional** Valid values: `uniform`, `normal`, or `constant` Default value: `normal`
`bias_init_scale`	Range for initialization of the bias term. Takes effect if `bias_init_method` is set to `uniform`. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: None
`bias_init_sigma`	The standard deviation for initialization of the bias term. Takes effect if `bias_init_method` is set to `normal`. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01
`bias_init_value`	The initial value of the bias term. Takes effect if `bias_init_method` is set to `constant`. **Optional** Valid values: Float. Suggested value range: [1e-8, 512]. Default value: None

Parameter Name	Description
`bias_lr`	The learning rate for the bias term. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.1
`bias_wd`	The weight decay for the bias term. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01
`clip_gradient`	Gradient clipping optimizer parameter. Clips the gradient by projecting onto the interval [`-clip_gradient`, `+clip_gradient`]. **Optional** Valid values: Float Default value: None
`epochs`	The number of training epochs to run. **Optional** Valid values: Positive integer Default value: 1
`eps`	Epsilon parameter to avoid division by 0. **Optional** Valid values: Float. Suggested value: small. Default value: None

Parameter Name	Description
factors_init_method	The initialization method for factorization terms: • `normal` Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by `factors_init_sigma`. • `uniform`: Initializes weights with random values uniformly sampled from a range specified by [-`factors_init_scale`, +`factors_init_scale`]. • `constant`: Initializes the weights to a scalar value specified by `factors_init_value`. **Optional** Valid values: `uniform`, `normal`, or `constant`. Default value: `normal`
factors_init_scale	The range for initialization of factorization terms. Takes effect if `factors_init_method` is set to `uniform`. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: None
factors_init_sigma	The standard deviation for initialization of factorization terms. Takes effect if `factors_init_method` is set to `normal`. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.001
factors_init_value	The initial value of factorization terms. Takes effect if `factors_init_method` is set to `constant`. **Optional** Valid values: Float. Suggested value range: [1e-8, 512]. Default value: None
factors_lr	The learning rate for factorization terms. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.0001

Parameter Name	Description
`factors_wd`	The weight decay for factorization terms. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.00001
`linear_lr`	The learning rate for linear terms. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.001
`linear_init_method`	The initialization method for linear terms: • `normal` Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by `linear_init_sigma`. • `uniform` Initializes weights with random values uniformly sampled from a range specified by [`-linear_init_scale`, `+linear_init_scale`]. • `constant` Initializes the weights to a scalar value specified by `linear_init_value`. **Optional** Valid values: `uniform`, `normal`, or `constant`. Default value: `normal`
`linear_init_scale`	Range for initialization of linear terms. Takes effect if `linear_init_method` is set to `uniform`. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: None
`linear_init_sigma`	The standard deviation for initialization of linear terms. Takes effect if `linear_init_method` is set to `normal`. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01

Parameter Name	Description
linear_init_value	The initial value of linear terms. Takes effect if linear_init_method is set to *constant*. **Optional** Valid values: Float. Suggested value range: [1e-8, 512]. Default value: None
linear_wd	The weight decay for linear terms. **Optional** Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.001
mini_batch_size	The size of mini-batch used for training. **Optional** Valid values: Positive integer Default value: 1000
rescale_grad	Gradient rescaling optimizer parameter. If set, multiplies the gradient with rescale_grad before updating. Often choose to be 1.0/batch_size. **Optional** Valid values: Float Default value: None

Tune a Factorization Machines Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see .

Metrics Computed by the Factorization Machines Algorithm

The factorization machines algorithm has both binary classification and regression predictor types. The predictor type determines which metric you can use for automatic model tuning. The algorithm reports a test:rmse regressor metric, which is computed during training. When tuning the model for regression tasks, choose this metric as the objective.

Metric Name	Description	Optimization Direction
test:rmse	Root Mean Square Error	Minimize

The factorization machines algorithm reports three binary classification metrics, which are computed during training. When tuning the model for binary classification tasks, choose one of these as the objective.

Metric Name	Description	Optimization Direction
test:binary_classification_accuracy	Accuracy	Maximize
test:binary_classification_cross_entropy	Cross Entropy	Minimize
test:binary_f_beta	Beta	Maximize

Tunable Factorization Machines Hyperparameters

You can tune the following hyperparameters for the factorization machines algorithm. The initialization parameters that contain the terms bias, linear, and factorization depend on their initialization method. There are three initialization methods: `uniform`, `normal`, and `constant`. These initialization methods are not themselves tunable. The parameters that are tunable are dependent on this choice of the initialization method. For example, if the initialization method is `uniform`, then only the `scale` parameters are tunable. Specifically, if `bias_init_method==uniform`, then `bias_init_scale`, `linear_init_scale`, and `factors_init_scale` are tunable. Similarly, if the initialization method is `normal`, then only `sigma` parameters are tunable. If the initialization method is `constant`, then only `value` parameters are tunable. These dependencies are listed in the following table.

Parameter Name	Parameter Type	Recommended Ranges	Dependency
bias_init_scale	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==uniform
bias_init_sigma	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==normal
bias_init_value	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==constant
bias_lr	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
bias_wd	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
epoch	IntegerParameterRange	MinValue: 1, MaxValue: 1000	None
factors_init_scale	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==uniform
factors_init_sigma	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==normal
factors_init_value	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==constant
factors_lr	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None

Parameter Name	Parameter Type	Recommended Ranges	Dependency
`factors_wd`	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512]	None
`linear_init_scale`	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==uniform
`linear_init_sigma`	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==normal
`linear_init_value`	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==constant
`linear_lr`	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
`linear_wd`	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
`mini_batch_size`	IntegerParameterRange	MinValue: 100, MaxValue: 10000	None

Factorization Machine Response Formats

JSON Response Format

Binary classification

```
let response =   {
    "predictions":    [
        {
            "score": 0.4,
            "predicted_label": 0
        }
    ]
}
```

Regression

```
let response =   {
    "predictions":    [
        {
            "score": 0.4
        }
    ]
}
```

JSONLINES Response Format

Binary classification

```
{"score": 0.4, "predicted_label": 0}
```

Regression

```
{"score": 0.4}
```

RECORDIO Response Format

Binary classification

```
[
    Record = {
        features = {},
        label = {
            'score': {
                keys: [],
                values: [0.4]  # float32
            },
            'predicted_label': {
                keys: [],
                values: [0.0]  # float32
            }
        }
    }
]
```

Regression

```
[
    Record = {
        features = {},
        label = {
            'score': {
                keys: [],
                values: [0.4]  # float32
            }
        }
    }
]
```

Image Classification Algorithm

The Amazon SageMaker image classification algorithm is a supervised learning algorithm that supports multi-label classification. It takes an image as input and outputs one or more labels assigned to that image. It uses a convolutional neural network (ResNet) that can be trained from scratch or trained using transfer learning when a large number of training images are not available.

The recommended input format for the Amazon SageMaker image classification algorithms is Apache MXNet RecordIO. However, you can also use raw images in .jpg or .png format.

> **Note**
> To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

For more information on convolutional networks, see:

- Deep residual learning for image recognition Kaiming He, et al., 2016 IEEE Conference on Computer Vision and Pattern Recognition
- ImageNet image database
- Image classification in MXNet

Topics

Input/Output Interface for the Image Classification Algorithm

The Amazon SageMaker Image Classification algorithm supports both RecordIO (`application/x-recordio`) and image (`image/png`, `image/jpeg`, and `application/x-image`) content types for training in file mode and supports RecordIO (`application/x-recordio`) content type for training in pipe mode. However you can also train in pipe mode using the image files (`image/png`, `image/jpeg`, and `application/x-image`), without creating RecordIO files, by using the augmented manifest format. Distributed training is currently not supported in pipe mode and can only be used in file mode. The algorithm supports `image/png`, `image/jpeg`, and `application/x-image` for inference.

Train with RecordIO Format

If you use the RecordIO format for training, specify both `train` and `validation` channels as values for the `InputDataConfig` parameter of the CreateTrainingJob (p. 667) request. Specify one RecordIO (`.rec`) file in the `train` channel and one RecordIO file in the `validation` channel. Set the content type for both channels to `application/x-recordio`.

Train with Image Format

If you use the Image format for training, specify `train`, `validation`, `train_lst`, and `validation_lst` channels as values for the `InputDataConfig` parameter of the CreateTrainingJob (p. 667) request. Specify the individual image data (`.jpg` or `.png` files) for the `train` and `validation` channels. Specify one `.lst` file in each of the `train_lst` and `validation_lst` channels. Set the content type for all four channels to `application/x-image`.

> **Note**
> Amazon SageMaker reads the training and validation data separately from different channels, so you must store the training and validation data in different folders.

A `.lst` file is a tab-separated file with three columns that contains a list of image files. The first column specifies the image index, the second column specifies the class label index for the image, and the third column specifies the relative path of the image file. The image index in the first column must be unique across all of the images. The set of class label indices are numbered successively and the numbering should start with 0. For example, 0 for the cat class, 1 for the dog class, and so on for additional classes.

The following is an example of a `.lst` file:

```
5       1    your_image_directory/train_img_dog1.jpg
1000    0    your_image_directory/train_img_cat1.jpg
22      1    your_image_directory/train_img_dog2.jpg
```

For example, if your training images are stored in `s3://<your_bucket>/train/class_dog`, `s3://<your_bucket>/train/class_cat`, and so on, specify the path for your `train` channel as `s3://<your_bucket>/train`, which is the top-level directory for your data. In the `.lst` file, specify the relative path for an individual file named `train_image_dog1.jpg` in the `class_dog` class directory as `class_dog/train_image_dog1.jpg`. You can also store all your image files under one subdirectory inside the `train` directory. In that case, use that subdirectory for the relative path. For example, `s3://<your_bucket>/train/your_image_directory`.

Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the `InputDataConfig` parameter of the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- `AlgorithmSpecification` - Identifies the training algorithm to use.

- `HyperParameters` - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- `InputDataConfig` - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- `OutputDataConfig` - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.
- `ResourceConfig` - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- `EnableManagedSpotTraining` - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.
- `RoleARN` - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- `StoppingCondition` - To help cap training costs, use `MaxRuntimeInSeconds` to set a time limit for training. Use `MaxWaitTimeInSeconds` to specify how long you are willing to to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see How It Works.

Request Syntax

```
{
    "AlgorithmSpecification": {

        "AlgorithmName": "string",

        "MetricDefinitions": [

            {

                "Name": "string",

                "Regex": "string"

            }

        ],
                                              110
        "TrainingImage": "string",

        "TrainingInputMode": "string"
```

```
   },

   "EnableInterContainerTrafficEncryption": boolean,

   "EnableManagedSpotTraining": boolean,

   "EnableNetworkIsolation": boolean,

   "HyperParameters": {

      "string" : "string"

   },

   "InputDataConfig": [

      {

         "ChannelName": "string",

         "CompressionType": "string",

         "ContentType": "string",

         "DataSource": {

            "FileSystemDataSource": {

               "DirectoryPath": "string",

               "FileSystemAccessMode": "string",

               "FileSystemId": "string",

               "FileSystemType": "string"

            },

            "S3DataSource": {

               "AttributeNames": [ "string" ],

               "S3DataDistributionType": "string",

               "S3DataType": "string",

               "S3Uri": "string"

            }

         },

         "InputMode": "string",

         "RecordWrapperType": "string",

         "ShuffleConfig": {

            "Seed": number
```
```
         }

      }
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 667)

> The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.
>
> Type: AlgorithmSpecification (p. 863) object
>
> Required: Yes

CheckpointConfig (p. 667)

> Contains information about the output location for managed spot training checkpoint data.
>
> Type: CheckpointConfig (p. 880) object
>
> Required: No

EnableInterContainerTrafficEncryption (p. 667)

> To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.
>
> Type: Boolean
>
> Required: No

EnableManagedSpotTraining (p. 667)

> To train models using managed spot training, choose `True`. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. this option is useful when training jobs can be interrupted and when there is flexibility when the training job is run.
>
> The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.
>
> Type: Boolean
>
> Required: No

EnableNetworkIsolation (p. 667)

> Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon

SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

> **Note**
> The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 667)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the `Length Constraint`.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

InputDataConfig (p. 667)

An array of `Channel` objects. Each channel is a named input source. `InputDataConfig` describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, `training_data` and `validation_data`. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

OutputDataConfig (p. 667)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 976) object

Required: Yes

ResourceConfig (p. 667)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose `File` as the `TrainingInputMode` in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 991) object

Required: Yes

RoleArn (p. 667)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

> **Note**
> To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StoppingCondition (p. 667)

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1004) object

Required: Yes

Tags (p. 667)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobName (p. 667)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

VpcConfig (p. 667)

A VpcConfig (p. 1039) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

Response Syntax

```
{
    "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 672)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(p.   ) request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in JSON Lines format in which each line represents one sample. The images are specified using the `'source-ref'` tag that points to the S3 location of the image. The annotations are provided under the `"AttributeNames"` parameter value as specified in the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- `AlgorithmSpecification` - Identifies the training algorithm to use.
- `HyperParameters` - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- `InputDataConfig` - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- `OutputDataConfig` - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.
- `ResourceConfig` - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- `EnableManagedSpotTraining` - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.
- `RoleARN` - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- `StoppingCondition` - To help cap training costs, use `MaxRuntimeInSeconds` to set a time limit for training. Use `MaxWaitTimeInSeconds` to specify how long you are willing to to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see How It Works.

Request Syntax

```
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
    },
    "EnableInterContainerTrafficEncryption": boolean,
    "EnableManagedSpotTraining": boolean,
    "EnableNetworkIsolation": boolean,
    "HyperParameters": {
        "string" : "string"
    },
    "InputDataConfig": [
        {
            "ChannelName": "string",
            "CompressionType": "string",
            "ContentType": "string",
            "DataSource": {
                "FileSystemDataSource": {
                    "DirectoryPath": "string",
```

```
        "MaxWaitTimeInSeconds": number

    },

    "Tags": [

        {

            "Key": "string",

            "Value": "string"

        }

    ],

    "TrainingJobName": "string",

    "VpcConfig": {

        "SecurityGroupIds": [ "string" ],

        "Subnets": [ "string" ]

    }

}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 667)

> The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.
>
> Type: AlgorithmSpecification (p. 863) object
>
> Required: Yes

CheckpointConfig (p. 667)

> Contains information about the output location for managed spot training checkpoint data.
>
> Type: CheckpointConfig (p. 880) object
>
> Required: No

EnableInterContainerTrafficEncryption (p. 667)

> To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.
>
> Type: Boolean

provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No

EnableNetworkIsolation (p. 667)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

> **Note**
> The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 667)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the `Length Constraint`.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

InputDataConfig (p. 667)

An array of `Channel` objects. Each channel is a named input source. `InputDataConfig` describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, `training_data` and `validation_data`. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

OutputDataConfig (p. 667)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 976) object

Required: Yes

ResourceConfig (p. 667)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose `File` as the `TrainingInputMode` in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 991) object

Required: Yes

RoleArn (p. 667)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

> **Note**
> To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StoppingCondition (p. 667)

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1004) object

Required: Yes

Tags (p. 667)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobName (p. 667)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

VpcConfig (p. 667)

A VpcConfig (p. 1039) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

Response Syntax

```
{
    "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 672)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the following example, the `"AttributeNames"` are contained in the list of image and annotation references `["source-ref", "class"]`. The corresponding label value is `"0"` for the first image and "1" for the second image:

```
{"source-ref":"s3://image/filename1.jpg", "class":"0"}
{"source-ref":"s3://image/filename2.jpg", "class":"1", "class-metadata": {"class-name":
 "cat", "type" : "groundtruth/image-classification"}}
```

The order of `"AttributeNames"` in the input files matters when training the ImageClassification algorithm. It accepts piped data in a specific order, with `image` first, followed by `label`. So the "AttributeNames" in this example are provided with `"source-ref"` first, followed by `"class"`. When using the ImageClassification algorithm with Augmented Manifest, the value of the `RecordWrapperType` parameter must be `"RecordIO"`.

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 308).

Incremental Training

You can also seed the training of a new model with the artifacts from a model that you trained previously with Amazon SageMaker. Incremental training saves training time when you want to train a new model with the same or similar data. Amazon SageMaker image classification models can be seeded only with another build-in image classification model trained in Amazon SageMaker.

To use a pretrained model, in the CreateTrainingJob (p. 667) request, specify the `ChannelName` as "model" in the `InputDataConfig` parameter. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the `num_layers`, `image_shape` and `num_classes` input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by Amazon SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the Amazon SageMaker image classification algorithm, see the End-to-End Incremental Training Image Classification Example. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 282).

Inference with the Image Format Algorithm

The generated models can be hosted for inference and support encoded `.jpg` and `.png` image formats as `image/png, image/jpeg`, and `application/x-image` content-type. The output is the probability values for all classes encoded in JSON format, or in JSON Lines text format for batch transform. The image classification model processes a single image per request and so outputs only one line in the JSON or JSON Lines format. The following is an example of a response in JSON Lines format:

```
accept: application/jsonlines

 {"prediction": [prob_0, prob_1, prob_2, prob_3, ...]}
```

For more details on training and inference, see the image classification sample notebook instances referenced in the introduction.

EC2 Instance Recommendation for the Image Classification Algorithm

For image classification, we support the following GPU instances for training: `ml.p2.xlarge`, `ml.p2.8xlarge`, `ml.p2.16xlarge`, `ml.p3.2xlarge`, `ml.p3.8xlarge` and `ml.p3.16xlarge`. We recommend using GPU instances with more memory for training with large batch sizes. However, both CPU (such as C4) and GPU (such as P2 and P3) instances can be used for the inference. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training.

Both P2 and P3 instances are supported in the image classification algorithm.

Image Classification Sample Notebooks

For a sample notebook that uses the Amazon SageMaker image classification algorithm to train a model on the caltech-256 dataset and then to deploy it to perform inferences, see the End-to-End Multiclass Image Classification Example. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The example image classification notebooks are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Image Classification Works

The image classification algorithm takes an image as input and classifies it into one of the output categories. Deep learning has revolutionized the image classification domain and has achieved great performance. Various deep learning networks such as ResNet [1], DenseNet, inception, and so on, have been developed to be highly accurate for image classification. At the same time, there have been efforts to collect labeled image data that are essential for training these networks. ImageNet[2] is one such large dataset that has more than 11 million images with about 11,000 categories. Once a network is trained with ImageNet data, it can then be used to generalize with other datasets as well, by simple re-adjustment or fine-tuning. In this transfer learning approach, a network is initialized with weights (in this example, trained on ImageNet), which can be later fine-tuned for an image classification task in a different dataset.

Image classification in Amazon SageMaker can be run in two modes: full training and transfer learning. In full training mode, the network is initialized with random weights and trained on user data from

scratch. In transfer learning mode, the network is initialized with pre-trained weights and just the top fully connected layer is initialized with random weights. Then, the whole network is fine-tuned with new data. In this mode, training can be achieved even with a smaller dataset. This is because the network is already trained and therefore can be used in cases without sufficient training data.

Image Classification Hyperparameters

Parameter Name	Description
num_classes	Number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset. **Required** Valid values: positive integer
num_training_samples	Number of training examples in the input dataset. If there is a mismatch between this value and the number of samples in the training set, then the behavior of the `lr_scheduler_step` parameter is undefined and distributed training accuracy might be affected. **Required** Valid values: positive integer
augmentation_type	Data augmentation type. The input images can be augmented in multiple ways as specified below. • `crop`: Randomly crop the image and flip the image horizontally • `crop_color`: In addition to 'crop', three random values in the range [-36, 36], [-50, 50], and [-50, 50] are added to the corresponding Hue-Saturation-Lightness channels respectively • `crop_color_transform`: In addition to `crop_color`, random transformations, including rotation, shear, and aspect ratio variations are applied to the image. The maximum angle of rotation is 10 degrees, the maximum shear ratio is 0.1, and the maximum aspect changing ratio is 0.25. **Optional** Valid values: `crop`, `crop_color`, or `crop_color_transform`. Default value: no default value
beta_1	The beta1 for `adam`, that is the exponential decay rate for the first moment estimates. **Optional** Valid values: float. Range in [0, 1]. Default value: 0.9
beta_2	The beta2 for `adam`, that is the exponential decay rate for the second moment estimates.

Parameter Name	Description
	Optional
	Valid values: float. Range in [0, 1].
	Default value: 0.999
`checkpoint_frequency`	Period to store model parameters (in number of epochs).
	Optional
	Valid values: positive integer no greater than `epochs`.
	Default value: None (Save checkpoint at the epoch that has the best validation accuracy.)
`early_stopping`	`True` to use early stopping logic during training. `False` not to use it.
	Optional
	Valid values: `True` or `False`
	Default value: `False`
`early_stopping_min_epochs`	The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when `early_stopping` = `True`.
	Optional
	Valid values: positive integer
	Default value: 10
`early_stopping_patience`	The number of epochs to wait before ending training if no improvement is made in the relevant metric. It is used only when `early_stopping` = `True`.
	Optional
	Valid values: positive integer
	Default value: 5
`early_stopping_tolerance`	Relative tolerance to measure an improvement in accuracy validation metric. If the ratio of the improvement in accuracy divided by the previous best accuracy is smaller than the `early_stopping_tolerance` value set, early stopping considers there is no improvement. It is used only when `early_stopping` = `True`.
	Optional
	Valid values: 0 ≤ float ≤ 1
	Default value: 0.0

Parameter Name	Description
epochs	Number of training epochs. **Optional** Valid values: positive integer Default value: 30
eps	The epsilon for `adam` and `rmsprop`. It is usually set to a small value to avoid division by 0. **Optional** Valid values: float. Range in [0, 1]. Default value: 1e-8
gamma	The gamma for `rmsprop`, the decay factor for the moving average of the squared gradient. **Optional** Valid values: float. Range in [0, 1]. Default value: 0.9
image_shape	The input image dimensions, which is the same size as the input layer of the network. The format is defined as 'num_channels, height, width'. The image dimension can take on any value as the network can handle varied dimensions of the input. However, there may be memory constraints if a larger image dimension is used. Typical image dimensions for image classification are '3, 224, 224'. This is similar to the ImageNet dataset. **Optional** Valid values: string Default value: '3, 224, 224'

Parameter Name	Description
kv_store	Weight update synchronization mode during distributed training. The weight updates can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See distributed training in MXNet for more details. This parameter is not applicable to single machine training. • dist_sync: The gradients are synchronized after every batch with all the workers. With dist_sync, batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then dist_sync behaves like local with batch size n*b • dist_async: Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed. **Optional** Valid values: dist_sync or dist_async Default value: no default value
learning_rate	Initial learning rate. **Optional** Valid values: float. Range in [0, 1]. Default value: 0.1
lr_scheduler_factor	The ratio to reduce learning rate used in conjunction with the lr_scheduler_step parameter, defined as lr_new = lr_old * lr_scheduler_factor. **Optional** Valid values: float. Range in [0, 1]. Default value: 0.1
lr_scheduler_step	The epochs at which to reduce the learning rate. As explained in the lr_scheduler_factor parameter, the learning rate is reduced by lr_scheduler_factor at these epochs. For example, if the value is set to "10, 20", then the learning rate is reduced by lr_scheduler_factor after 10th epoch and again by lr_scheduler_factor after 20th epoch. The epochs are delimited by ",". **Optional** Valid values: string Default value: no default value

Parameter Name	Description
`mini_batch_size`	The batch size for training. In a single-machine multi-GPU setting, each GPU handles `mini_batch_size/num_gpu` training samples. For the multi-machine training in dist_sync mode, the actual batch size is `mini_batch_size*number` of machines. See MXNet docs for more details. **Optional** Valid values: positive integer Default value: 32
`momentum`	The momentum for `sgd` and `nag`, ignored for other optimizers. **Optional** Valid values: float. Range in [0, 1]. Default value: 0.9
`multi_label`	Flag to use for multi-label classification where each sample can be assigned multiple labels. Average accuracy across all classes is logged. **Optional** Valid values: 0 or 1 Default value: 0
`num_layers`	Number of layers for the network. For data with large image size (for example, 224x224 - like ImageNet), we suggest selecting the number of layers from the set [18, 34, 50, 101, 152, 200]. For data with small image size (for example, 28x28 - like CIFAR), we suggest selecting the number of layers from the set [20, 32, 44, 56, 110]. The number of layers in each set is based on the ResNet paper. For transfer learning, the number of layers defines the architecture of base network and hence can only be selected from the set [18, 34, 50, 101, 152, 200]. **Optional** Valid values: positive integer in [18, 34, 50, 101, 152, 200] or [20, 32, 44, 56, 110] Default value: 152

Parameter Name	Description
`optimizer`	The optimizer type. For more details of the parameters for the optimizers, please refer to MXNet's API. **Optional** Valid values: One of `sgd`, `adam`, `rmsprop`, or `nag`. - `sgd`: Stochastic gradient descent - `adam`: Adaptive momentum estimation - `rmsprop`: Root mean square propagation - `nag`: Nesterov accelerated gradient Default value: `sgd`
`precision_dtype`	The precision of the weights used for training. The algorithm can use either single precision (`float32`) or half precision (`float16`) for the weights. Using half-precision for weights results in reduced memory consumption. **Optional** Valid values: `float32` or `float16` Default value: `float32`
`resize`	Resizes the image before using it for training. The images are resized so that the shortest side has the number of pixels specified by this parameter. If the parameter is not set, then the training data is used without resizing. **Optional** Valid values: positive integer Default value: no default value
`top_k`	Reports the top-k accuracy during training. This parameter has to be greater than 1, since the top-1 training accuracy is the same as the regular training accuracy that has already been reported. **Optional** Valid values: positive integer larger than 1. Default value: no default value

Parameter Name	Description
use_pretrained_model	Flag to use pre-trained model for training. If set to 1, then the pretrained model with the corresponding number of layers is loaded and used for training. Only the top FC layer are reinitialized with random weights. Otherwise, the network is trained from scratch. **Optional** Valid values: 0 or 1 Default value: 0
use_weighted_loss	Flag to use weighted cross-entropy loss for multi-label classification (used only when `multi_label` = 1), where the weights are calculated based on the distribution of classes. **Optional** Valid values: 0 or 1 Default value: 0
weight_decay	The coefficient weight decay for `sgd` and `nag`, ignored for other optimizers. **Optional** Valid values: float. Range in [0, 1]. Default value: 0.0001

Tune an Image Classification Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the Image Classification Algorithm

The image classification algorithm is a supervised algorithm. It reports an accuracy metric that is computed during training. When tuning the model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
validation:accuracy	The ratio of the number of correct predictions to the total number of predictions made.	Maximize

Tunable Image Classification Hyperparameters

Tune an image classification model with the following hyperparameters. The hyperparameters that have the greatest impact on image classification objective metrics are: `mini_batch_size`, `learning_rate`,

and `optimizer`. Tune the optimizer-related hyperparameters, such as `momentum`, `weight_decay`, `beta_1`, `beta_2`, eps, and `gamma`, based on the selected `optimizer`. For example, use `beta_1` and `beta_2` only when `adam` is the `optimizer`.

For more information about which hyperparameters are used in each optimizer, see Image Classification Hyperparameters (p. 124).

Parameter Name	Parameter Type	Recommended Ranges
`beta_1`	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
`beta_2`	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
`eps`	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 1.0
`gamma`	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 0.999
`learning_rate`	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.5
`mini_batch_size`	IntegerParameterRanges	MinValue: 8, MaxValue: 512
`momentum`	ContinuousParameterRanges	MinValue: 0.0, MaxValue: 0.999
`optimizer`	CategoricalParameterRanges	['sgd', 'adam', 'rmsprop', 'nag']
`weight_decay`	ContinuousParameterRanges	MinValue: 0.0, MaxValue: 0.999

IP Insights Algorithm

Amazon SageMaker IP Insights is an unsupervised learning algorithm that learns the usage patterns for IPv4 addresses. It is designed to capture associations between IPv4 addresses and various entities, such as user IDs or account numbers. You can use it to identify a user attempting to log into a web service from an anomalous IP address, for example. Or you can use it to identify an account that is attempting to create computing resources from an unusual IP address. Trained IP Insight models can be hosted at an endpoint for making real-time predictions or used for processing batch transforms

Amazon SageMaker IP insights ingests historical data as (entity, IPv4 Address) pairs and learns the IP usage patterns of each entity. When queried with an (entity, IPv4 Address) event, an Amazon SageMaker IP Insights model returns a score that infers how anomalous the pattern of the event is. For example, when a user attempts to log in from an IP address, if the IP Insights score is high enough, a web login server might decide to trigger a multi-factor authentication system. In more advanced solutions, you can feed the IP Insights score into another machine learning model. For example, you can combine the IP Insight score with other features to rank the findings of another security system, such as those from Amazon GuardDuty.

The Amazon SageMaker IP Insights algorithm can also learn vector representations of IP addresses, known as *embeddings*. You can use vector-encoded embeddings as features in downstream machine learning tasks that use the information observed in the IP addresses. For example, you can use them in tasks such as measuring similarities between IP addresses in clustering and visualization tasks.

Topics

Input/Output Interface for the IP Insights Algorithm

Training and Validation

The Amazon SageMaker IP Insights algorithm supports training and validation data channels. It uses the optional validation channel to compute an area-under-curve (AUC) score on a predefined negative sampling strategy. The AUC metric validates how well the model discriminates between positive and negative samples. Training and validation data content types need to be in `text/csv` format. The first column of the CSV data is an opaque string that provides a unique identifier for the entity. The second column is an IPv4 address in decimal-dot notation. IP Insights currently supports only File mode. For more information and some examples, see IP Insights Training Data Formats (p. 138).

Inference

For inference, IP Insights supports `text/csv`, `application/json`, and `application/jsonlines` data content types. For more information about the common data formats for inference provided by Amazon SageMaker, see Common Data Formats for Inference (p. 68). IP Insights inference returns output formatted as either `application/json` or `application/jsonlines`. Each record in the output data contains the corresponding `dot_product` (or compatibility score) for each input data point. For more information and some examples, see IP Insights Inference Data Formats (p. 139).

EC2 Instance Recommendation for the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm can run on both GPU and CPU instances. For training jobs, we recommend using GPU instances. However, for certain workloads with large training datasets, distributed CPU instances might reduce training costs. For inference, we recommend using CPU instances.

GPU Instances for the IP Insights Algorithm

IP Insights supports all available GPUs. If you need to speed up training, we recommend starting with a single GPU instance, such as ml.p3.2xlarge, and then moving to a multi-GPU environment, such as ml.p3.8xlarge and ml.p3.16xlarge. Multi-GPUs automatically divide the mini batches of training data across themselves. If you switch from a single GPU to multiple GPUs, the `mini_batch_size` is divided equally into the number of GPUs used. You may want to increase the value of the `mini_batch_size` to compensate for this.

CPU Instances for the IP Insights Algorithm

The type of CPU instance that we recommend depends largely on the instance's available memory and the model size. The model size is determined by two hyperparameters: `vector_dim` and `num_entity_vectors`. The maximum supported model size is 8 GB. The following table lists typical EC2 instance types that you would deploy based on these input parameters for various model sizes. In Table 1, the value for `vector_dim` in the first column range from 32 to 2048 and the values for `num_entity_vectors` in the first row range from 10,000 to 50,00,000.

vector_dim \ num_entities	10,000	50,000	100,000	500,000	1,000,000	5,000,000	10,000,000	50,000,000
32	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.2xlarge	ml.m5.4xlarge
64	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge	ml.m5.2xlarge	
128	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge	ml.m5.4xlarge	
256	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.4xlarge		
512	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge			
1024	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.4xlarge			
2048	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.xlarge				

The values for the `mini_batch_size`, `num_ip_encoder_layers`, `random_negative_sampling_rate`, and `shuffled_negative_sampling_rate` hyperparameters also affect the amount of memory required. If these values are large, you might need to use a larger instance type than normal.

IP Insights Sample Notebooks

For a sample notebook that shows how to train the Amazon SageMaker IP Insights algorithm and perform inferences with it, see An Introduction to the Amazon SageMakerIP Insights Algorithm . For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). After creating a notebook instance, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker examples. To open a notebook, choose its **Use** tab and choose **Create copy**.

How IP Insights Works

Amazon SageMaker IP Insights is an unsupervised algorithm that consumes observed data in the form of (entity, IPv4 address) pairs that associates entities with IP addresses. IP Insights determines how likely it is that an entity would use a particular IP address by learning latent vector representations for both entities and IP addresses. The distance between these two representations can then serve as the proxy for how likely this association is.

The IP Insights algorithm uses a neural network to learn the latent vector representations for entities and IP addresses. Entities are first hashed to a large but fixed hash space and then encoded by a simple embedding layer. Character strings such as user names or account IDs can be fed directly into IP Insights as they appear in log files. You don't need to preprocess the data for entity identifiers. You can provide entities as an arbitrary string value during both training and inference. The hash size should be configured with a value that is high enough to insure that the number of *collisions*, which occur when distinct entities are mapped to the same latent vector, remain insignificant. For more information about how to select appropriate hash sizes, see Feature Hashing for Large Scale Multitask Learning. For representing IP addresses, on the other hand, IP Insights uses a specially designed encoder network to uniquely represent each possible IPv4 address by exploiting the prefix structure of IP addresses.

During training, IP Insights automatically generates negative samples by randomly pairing entities and IP addresses. These negative samples represent data that is less likely to occur in reality. The model is trained to discriminate between positive samples that are observed in the training data and these generated negative samples. More specifically, the model is trained to minimize the *cross entropy*, also known as the *log loss*, defined as follows:

$$L = \frac{1}{N} \sum_n [y_n \log p_n + (1 - y_n) \log (1 - p_n)]$$

y_n is the label that indicates whether the sample is from the real distribution governing observed data (y_n=1) or from the distribution generating negative samples (y_n=0). p_n is the probability that the sample is from the real distribution, as predicted by the model.

Generating negative samples is an important process that is used to achieve an accurate model of the observed data. If negative samples are extremely unlikely, for example, if all of the IP addresses in negative samples are 10.0.0.0, then the model trivially learns to distinguish negative samples and fails to accurately characterize the actual observed dataset. To keep negative samples more realistic, IP Insights generates negative samples both by randomly generating IP addresses and randomly picking IP addresses from training data. You can configure the type of negative sampling and the rates at which negative samples are generated with the `random_negative_sampling_rate` and `shuffled_negative_sampling_rate` hyperparameters.

Given an nth (entity, IP address pair), the IP Insights model outputs a *score*, S_n , that indicates how compatible the entity is with the IP address. This score corresponds to the log odds ratio for a given (entity, IP address) of the pair coming from a real distribution as compared to coming from a negative distribution. It is defined as follows:

$$S_n = \log \left(\frac{P_{real}(n)}{P_{neg}(n)} \right)$$

The score is essentially a measure of the similarity between the vector representations of the nth entity and IP address. It can be interpreted as how much more likely it would be to observe this event in reality than in a randomly generated dataset. During training, the algorithm uses this score to calculate an estimate of the probability of a sample coming from the real distribution, p_n, to use in the cross entropy minimization, where:

$$p_n = \frac{1}{1 + e^{-S_n}}$$

IP Insights Hyperparameters

In the request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker IP Insights algorithm.

Parameter Name	Description
`num_entity_vectors`	The number of entity vector representations (entity embedding vectors) to train. Each entity in the training set is randomly assigned to one of these vectors using a hash function. Because of hash collisions, it might be possible to have multiple entities assigned to the same vector. This would cause the same vector to represent multiple entities. This generally has a negligible effect on model performance, as long as the collision rate is not too severe. To keep the collision rate low, set this value as high as possible. However, the model size, and, therefore, the memory requirement, for both training and inference, scales linearly with this hyperparameter. We recommend that you set this value to twice the number of unique entity identifiers. **Required**

Parameter Name	Description
	Valid values: 1 ≤ positive integer ≤ 250,000,000
vector_dim	The size of embedding vectors to represent entities and IP addresses. The larger the value, the more information that can be encoded using these representations. In practice, model size scales linearly with this parameter and limits how large the dimension can be. In addition, using vector representations that are too large can cause the model to overfit, especially for small training datasets. Overfitting occurs when a model doesn't learn any pattern in the data but effectively memorizes the training data and, therefore, cannot generalize well and performs poorly during inference. The recommended value is 128. **Required** Valid values: 4 ≤ positive integer ≤ 4096
batch_metrics_publish_interval	The interval (every X batches) at which the Apache MXNet Speedometer function prints the training speed of the network (samples/second). **Optional** Valid values: positive integer ≥ 1 Default value: 1,000
epochs	The number of passes over the training data. The optimal value depends on your data size and learning rate. Typical values range from 5 to 100. **Optional** Valid values: positive integer ≥ 1 Default value: 10
learning_rate	The learning rate for the optimizer. IP Insights use a gradient-descent-based Adam optimizer. The learning rate effectively controls the step size to update model parameters at each iteration. Too large a learning rate can cause the model to diverge because the training is likely to overshoot a minima. On the other hand, too small a learning rate slows down convergence. Typical values range from 1e-4 to 1e-1. **Optional** Valid values: 1e-6 ≤ float ≤ 10.0 Default value: 0.001

Parameter Name	Description
`mini_batch_size`	The number of examples in each mini batch. The training procedure processes data in mini batches. The optimal value depends on the number of unique account identifiers in the dataset. In general, the larger the `mini_batch_size`, the faster the training and the greater the number of possible shuffled-negative-sample combinations. However, with a large `mini_batch_size`, the training is more likely to converge to a poor local minimum and perform relatively worse for inference. **Optional** Valid values: 1 ≤ positive integer ≤ 500000 Default value: 10,000
`num_ip_encoder_layers`	The number of fully connected layers used to encode the IP address embedding. The larger the number of layers, the greater the model's capacity to capture patterns among IP addresses. However, using a large number of layers increases the chance of overfitting. **Optional** Valid values: 0 ≤ positive integer ≤ 100 Default value: 1
`random_negative_sampling_rate`	The number of random negative samples, R, to generate per input example. The training procedure relies on negative samples to prevent the vector representations of the model collapsing to a single point. Random negative sampling generates R random IP addresses for each input account in the mini batch. The sum of the `random_negative_sampling_rate` (R) and `shuffled_negative_sampling_rate` (S) must be in the interval: 1 ≤ R + S ≤ 500. **Optional** Valid values: 0 ≤ positive integer ≤ 500 Default value: 1

Parameter Name	Description
`shuffled_negative_sampling_rate`	The number of shuffled negative samples, S, to generate per input example. In some cases, it helps to use more realistic negative samples that are randomly picked from the training data itself. This kind of negative sampling is achieved by shuffling the data within a mini batch. Shuffled negative sampling generates S negative IP addresses by shuffling the IP address and account pairings within a mini batch. The sum of the `random_negative_sampling_rate` (R) and `shuffled_negative_sampling_rate` (S) must be in the interval: 1 ≤ R + S ≤ 500. **Optional** Valid values: 0 ≤ positive integer ≤ 500 Default value: 1
`weight_decay`	The weight decay coefficient. This parameter adds an L2 regularization factor that is required to prevent the model from overfitting the training data. **Optional** Valid values: 0.0 ≤ float ≤ 10.0 Default value: 0.00001

Tune an IP Insights Model

Automatic model tuning, also called hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see .

Metrics Computed by the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm is an unsupervised learning algorithm that learns associations between IP addresses and entities. The algorithm trains a discriminator model , which learns to separate observed data points (*positive samples*) from randomly generated data points (*negative samples*). Automatic model tuning on IP Insights helps you find the model that can most accurately distinguish between unlabeled validation data and automatically generated negative samples. The model accuracy on the validation dataset is measured by the area under the receiver operating characteristic (ROC) curve. This `validation:discriminator_auc` metric can take values between 0.0 and 1.0, where 1.0 indicates perfect accuracy.

The IP Insights algorithm computes a `validation:discriminator_auc` metric during validation, the value of which is used as the objective function to optimize for hyperparameter tuning.

Metric Name	Description	Optimization Direction
validation:discriminator_auc	Area under the ROC curve on the validation dataset. The validation dataset is not labeled. AUC is a metric that describes the model's ability to discriminate validation data points from randomly generated data points.	Maximize

Tunable IP Insights Hyperparameters

You can tune the following hyperparameters for the Amazon SageMaker IP Insights algorithm.

Parameter Name	Parameter Type	Recommended Ranges
epochs	IntegerParameterRange	MinValue: 1, MaxValue: 100
learning_rate	ContinuousParameterRange	MinValue: 1e-4, MaxValue: 0.1
mini_batch_size	IntegerParameterRanges	MinValue: 100, MaxValue: 50000
num_entity_vectors	IntegerParameterRanges	MinValue: 10000, MaxValue: 1000000
num_ip_encoder_layers	IntegerParameterRanges	MinValue: 1, MaxValue: 10
random_negative_sampling_rate	IntegerParameterRanges	MinValue: 0, MaxValue: 10
shuffled_negative_sampling_rate	IntegerParameterRanges	MinValue: 0, MaxValue: 10
vector_dim	IntegerParameterRanges	MinValue: 8, MaxValue: 256
weight_decay	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0

IP Insights Data Formats

This section provides examples of the available input and output data formats used by the IP Insights algorithm during training and inference.

Topics

IP Insights Training Data Formats

The following are the available data input formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input training format described in Common Data Formats for

However, the Amazon SageMaker IP Insights algorithm currently supports only the CSV data input format.

IP Insights Training Data Input Formats

INPUT: CSV

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

IP Insights Inference Data Formats

The following are the available input and output formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input inference format described in However, the Amazon SageMaker IP Insights algorithm does not currently support RecordIO format.

IP Insights Input Request Formats

INPUT: CSV Format

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

INPUT: JSON Format

JSON data can be provided in different formats. IP Insights follows the common Amazon SageMaker formats. For more information about inference formats, see

content-type: application/json

```
{
  "instances": [
    {"data": {"features": {"values": ["entity_id_1", "192.168.1.2"]}}},
    {"features": ["entity_id_2", "10.10.1.2"]}
  ]
}
```

INPUT: JSONLINES Format

The JSON Lines content type is useful for running batch transform jobs. For more information on Amazon SageMaker inference formats, see For more information on running batch transform jobs, see

content-type: application/jsonlines

```
{"data": {"features": {"values": ["entity_id_1", "192.168.1.2"]}}},
{"features": ["entity_id_2", "10.10.1.2"]}]
```

IP Insights Output Response Formats

OUTPUT: JSON Response Format

The default output of the Amazon SageMaker IP Insights algorithm is the `dot_product` between the input entity and IP address. The dot_product signifies how compatible the model considers the entity and IP address. The `dot_product` is unbounded. To make predictions about whether an event is anomalous, you need to set a threshold based on your defined distribution. For information about how to use the `dot_product` for anomaly detection, see the An Introduction to the Amazon SageMakerIP Insights Algorithm.

accept: application/json

```
{
  "predictions": [
    {"dot_product": 0.0},
    {"dot_product": 2.0}
  ]
}
```

Advanced users can access the model's learned entity and IP embeddings by providing the additional content-type parameter `verbose=True` to the Accept heading. You can use the `entity_embedding` and `ip_embedding` for debugging, visualizing, and understanding the model. Additionally, you can use these embeddings in other machine learning techniques, such as classification or clustering.

accept: application/json;verbose=True

```
{
  "predictions": [
    {
        "dot_product": 0.0,
        "entity_embedding": [1.0, 0.0, 0.0],
        "ip_embedding": [0.0, 1.0, 0.0]
    },
    {
        "dot_product": 2.0,
        "entity_embedding": [1.0, 0.0, 1.0],
        "ip_embedding": [1.0, 0.0, 1.0]
    }
  ]
}
```

OUTPUT: JSONLINES Response Format

accept: application/jsonlines

```
{"dot_product": 0.0}
{"dot_product": 2.0}
```

accept: application/jsonlines; verbose=True

```
{"dot_product": 0.0, "entity_embedding": [1.0, 0.0, 0.0], "ip_embedding": [0.0, 1.0, 0.0]}
{"dot_product": 2.0, "entity_embedding": [1.0, 0.0, 1.0], "ip_embedding": [1.0, 0.0, 1.0]}
```

K-Means Algorithm

K-means is an unsupervised learning algorithm. It attempts to find discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups. You define the attributes that you want the algorithm to use to determine similarity.

Amazon SageMaker uses a modified version of the web-scale k-means clustering algorithm. Compared with the original version of the algorithm, the version used by Amazon SageMaker is more accurate. Like the original algorithm, it scales to massive datasets and delivers improvements in training time. To do this, the version used by Amazon SageMaker streams mini-batches (small, random subsets) of the training data. For more information about mini-batch k-means, see Web-scale k-means Clustering.

The k-means algorithm expects tabular data, where rows represent the observations that you want to cluster, and the columns represent attributes of the observations. The *n* attributes in each row represent a point in *n*-dimensional space. The Euclidean distance between these points represents the similarity of the corresponding observations. The algorithm groups observations with similar attribute values (the points corresponding to these observations are closer together). For more information about how k-means works in Amazon SageMaker, see How K-Means Clustering Works (p. 142).

Topics

Input/Output Interface for the K-Means Algorithm

For training, the k-means algorithm expects data to be provided in the *train* channel (recommended `S3DataDistributionType=ShardedByS3Key`), with an optional *test* channel (recommended `S3DataDistributionType=FullyReplicated`) to score the data on. Both `recordIO-wrapped-protobuf` and `CSV` formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, `text/csv`, `application/json`, and `application/x-recordio-protobuf` are supported. k-means returns a `closest_cluster` label and the `distance_to_cluster` for each observation.

For more information on input and output file formats, see K-Means Response Formats (p. 147) for inference and the K-Means Sample Notebooks (p. 141). The k-means algorithm does not support multiple instance learning, in which the training set consists of labeled "bags", each of which is a collection of unlabeled instances.

EC2 Instance Recommendation for the K-Means Algorithm

We recommend training k-means on CPU instances. You can train on GPU instances, but should limit GPU training to `p*.xlarge` instances because only one GPU per instance is used.

K-Means Sample Notebooks

For a sample notebook that uses the Amazon SageMaker K-means algorithm to segment the population of counties in the United States by attributes identified using principle component analysis, see Analyze

US census data for population segmentation using Amazon SageMaker. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its **Use** tab and select **Create copy**.

How K-Means Clustering Works

K-means is an algorithm that trains a model that groups similar objects together. The k-means algorithm accomplishes this by mapping each observation in the input dataset to a point in the *n*-dimensional space (where *n* is the number of attributes of the observation). For example, your dataset might contain observations of temperature and humidity in a particular location, which are mapped to points (*t, h*) in 2-dimensional space.

> **Note**
> Clustering algorithms are unsupervised. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

In k-means clustering, each cluster has a center. During model training, the k-means algorithm uses the distance of the point that corresponds to each observation in the dataset to the cluster centers as the basis for clustering. You choose the number of clusters (*k*) to create.

For example, suppose that you want to create a model to recognize handwritten digits and you choose the MNIST dataset for training. The dataset provides thousands of images of handwritten digits (0 through 9). In this example, you might choose to create 10 clusters, one for each digit (0, 1, …, 9). As part of model training, the k-means algorithm groups the input images into 10 clusters.

Each image in the MNIST dataset is a 28x28-pixel image, with a total of 784 pixels. Each image corresponds to a point in a 784-dimensional space, similar to a point in a 2-dimensional space (x,y). To find a cluster to which a point belongs, the k-means algorithm finds the distance of that point from all of the cluster centers. It then chooses the cluster with the closest center as the cluster to which the image belongs.

> **Note**
> Amazon SageMaker uses a customized version of the algorithm where, instead of specifying that the algorithm create *k* clusters, you might choose to improve model accuracy by specifying extra cluster centers *(K = k*x)*. However, the algorithm ultimately reduces these to *k* clusters.

In Amazon SageMaker, you specify the number of clusters when creating a training job. For more information, see CreateTrainingJob (p. 667). In the request body, you add the `HyperParameters` string map to specify the `k` and `extra_center_factor` strings.

The following is a summary of how k-means works for model training in Amazon SageMaker:

1. It determines the initial *K* cluster centers.

   > **Note**
   > In the following topics, *K* clusters refer to *k * x*, where you specify *k* and *x* when creating a model training job.

2. It iterates over input training data and recalculates cluster centers.

3. It reduces resulting clusters to *k* (if the data scientist specified the creation of *k*x* clusters in the request).

The following sections also explain some of the parameters that a data scientist might specify to configure a model training job as part of the `HyperParameters` string map.

Topics

Step 1: Determine the Initial Cluster Centers

When using k-means in Amazon SageMaker, the initial cluster centers are chosen from the observations in a small, randomly sampled batch. Choose one of the following strategies to determine how these initial cluster centers are selected:

- The random approach—Randomly choose K observations in your input dataset as cluster centers. For example, you might choose a cluster center that points to the 784-dimensional space that corresponds to any 10 images in the MNIST training dataset.

- The k-means++ approach, which works as follows:
  1. Start with one cluster and determine its center. You randomly select an observation from your training dataset and use the point corresponding to the observation as the cluster center. For example, in the MNIST dataset, randomly choose a handwritten digit image. Then choose the point in the 784-dimensional space that corresponds to the image as your cluster center. This is cluster center 1.
  2. Determine the center for cluster 2. From the remaining observations in the training dataset, pick an observation at random. Choose one that is different than the one you previously selected. This observation corresponds to a point that is far away from cluster center 1. Using the MNIST dataset as an example, you do the following:
     - For each of the remaining images, find the distance of the corresponding point from cluster center 1. Square the distance and assign a probability that is proportional to the square of the distance. That way, an image that is different from the one that you previously selected has a higher probability of getting selected as cluster center 2.
     - Choose one of the images randomly, based on probabilities assigned in the previous step. The point that corresponds to the image is cluster center 2.
  3. Repeat Step 2 to find cluster center 3. This time, find the distances of the remaining images from cluster center 2.
  4. Repeat the process until you have the K cluster centers.

To train a model in Amazon SageMaker, you create a training job. In the request, you provide configuration information by specifying the following `HyperParameters` string maps:

- To specify the number of clusters to create, add the `k` string.
- For greater accuracy, add the optional `extra_center_factor` string.
- To specify the strategy that you want to use to determine the initial cluster centers, add the `init_method` string and set its value to `random` or `k-means++`.

For more information, see CreateTrainingJob (p. 667). For an example, see Create and Run a Training Job (AWS SDK for Python (Boto 3)) (p. 23).

You now have an initial set of cluster centers.

Step 2: Iterate over the Training Dataset and Calculate Cluster Centers

The cluster centers that you created in the preceding step are mostly random, with some consideration for the training dataset. In this step, you use the training dataset to move these centers toward the true cluster centers. The algorithm iterates over the training dataset, and recalculates the K cluster centers.

1. Read a mini-batch of observations (a small, randomly chosen subset of all records) from the training dataset and do the following.

   Note
   When creating a model training job, you specify the batch size in the `mini_batch_size` string in the `HyperParameters` string map.

   a. Assign all of the observations in the mini-batch to one of the clusters with the closest cluster center.

   b. Calculate the number of observations assigned to each cluster. Then, calculate the proportion of new points assigned per cluster.

   For example, consider the following clusters:

   Cluster c1 = 100 previously assigned points. You added 25 points from the mini-batch in this step.

   Cluster c2 = 150 previously assigned points. You added 40 points from the mini-batch in this step.

   Cluster c3 = 450 previously assigned points. You added 5 points from the mini-batch in this step.

   Calculate the proportion of new points assigned to each of clusters as follows:

   ```
   p1 = proportion of points assigned to c1 = 25/(100+25)
   p2 = proportion of points assigned to c2 = 40/(150+40)
   p3 = proportion of points assigned to c3 = 5/(450+5)
   ```

   c. Compute the center of the new points added to each cluster:

   ```
   d1 = center of the new points added to cluster 1
   d2 = center of the new points added to cluster 2
   d3 = center of the new points added to cluster 3
   ```

   d. Compute the weighted average to find the updated cluster centers as follows:

   ```
   Center of cluster 1 = ((1 - p1) * center of cluster 1) + (p1 * d1)
   Center of cluster 2 = ((1 - p2) * center of cluster 2) + (p2 * d2)
   Center of cluster 3 = ((1 - p3) * center of cluster 3) + (p3 * d3)
   ```

2. Read the next mini-batch, and repeat Step 1 to recalculate the cluster centers.

3. For more information about mini-batch k-means, see Web-Scale k-means Clustering).

Step 3: Reduce the Clusters from K to k

If the algorithm created K clusters—*(K = k*x)* where *x* is greater than 1—then it reduces the K clusters to k clusters. (For more information, see `extra_center_factor` in the preceding discussion.) It does this by applying Lloyd's method with `kmeans++` initialization to the K cluster centers. For more information about Lloyd's method, see k-means clustering.

K-Means Hyperparameters

In the CreateTrainingJob (p. 667) request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the k-means training algorithm provided by Amazon SageMaker. For more information about how k-means clustering works, see How K-Means Clustering Works (p. 142).

Parameter Name	Description
`feature_dim`	The number of features in the input data. **Required** Valid values: Positive integer
`k`	The number of required clusters. **Required** Valid values: Positive integer
`epochs`	The number of passes done over the training data. **Optional** Valid values: Positive integer Default value: 1
`eval_metrics`	A JSON list of metric types used to report a score for the model. Allowed values are `msd` for Means Square Error and `ssd` for Sum of Square Distance. If test data is provided, the score is reported for each of the metrics requested. **Optional** Valid values: Either `[\"msd\"]` or `[\"ssd\"]` or `[\"msd\", \"ssd\"]`. Default value: `[\"msd\"]`
`extra_center_factor`	The algorithm creates K centers = `num_clusters` * `extra_center_factor` as it runs and reduces the number of centers from K to `k` when finalizing the model. **Optional** Valid values: Either a positive integer or `auto`. Default value: `auto`
`half_life_time_size`	Used to determine the weight given to an observation when computing a cluster mean. This weight decays exponentially as more points are observed. When a point is first observed, it is assigned a weight of 1 when computing the cluster mean. The decay constant for the exponential decay function is chosen so that after observing `half_life_time_size` points, its weight is 1/2. If set to 0, there is no decay. **Optional** Valid values: Non-negative integer Default value: 0
`init_method`	Method by which the algorithm chooses the initial cluster centers. The standard k-means approach chooses them at random. An

Parameter Name	Description
	alternative k-means++ method chooses the first cluster center at random. Then it spreads out the position of the remaining initial clusters by weighting the selection of centers with a probability distribution that is proportional to the square of the distance of the remaining data points from existing centers. **Optional** Valid values: Either `random` or `kmeans++`. Default value: `random`
`local_lloyd_init_method`	The initialization method for Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers. **Optional** Valid values: Either `random` or `kmeans++`. Default value: `kmeans++`
`local_lloyd_max_iter`	The maximum number of iterations for Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers. **Optional** Valid values: Positive integer Default value: 300
`local_lloyd_num_trials`	The number of times the Lloyd's expectation-maximization (EM) procedure with the least loss is run when building the final model containing k centers. **Optional** Valid values: Either a positive integer or `auto`. Default value: `auto`
`local_lloyd_tol`	The tolerance for change in loss for early stopping of Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers. **Optional** Valid values: Float. Range in [0, 1]. Default value: 0.0001
`mini_batch_size`	The number of observations per mini-batch for the data iterator. **Optional** Valid values: Positive integer Default value: 5000

Tune a K-Means Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker k-means algorithm is an unsupervised algorithm that groups data into clusters whose members are as similar as possible. Because it is unsupervised, it doesn't use a validation dataset that hyperparameters can optimize against. But it does take a test dataset and emits metrics that depend on the squared distance between the data points and the final cluster centroids at the end of each training run. To find the model that reports the tightest clusters on the test dataset, you can use a hyperparameter tuning job. The clusters optimize the similarity of their members.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the K-Means Algorithm

The k-means algorithm computes the following metrics during training. When tuning a model, choose one of these metrics as the objective metric.

Metric Name	Description	Optimization Direction
test:msd	Mean squared distances between each record in the test set and the closest center of the model.	Minimize
test:ssd	Sum of the squared distances between each record in the test set and the closest center of the model.	Minimize

Tunable K-Means Hyperparameters

Tune the Amazon SageMaker k-means model with the following hyperparameters. The hyperparameters that have the greatest impact on k-means objective metrics are: `mini_batch_size`, `extra_center_factor`, and `init_method`. Tuning the hyperparameter `epochs` generally results in minor improvements.

Parameter Name	Parameter Type	Recommended Ranges
epochs	IntegerParameterRanges	MinValue: 1, MaxValue:10
extra_center_factor	IntegerParameterRanges	MinValue: 4, MaxValue:10
init_method	CategoricalParameterRanges	['kmeans++', 'random']
mini_batch_size	IntegerParameterRanges	MinValue: 3000, MaxValue:15000

K-Means Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker k-means algorithm.

JSON Response Format

```
{
    "predictions": [
        {
            "closest_cluster": 1.0,
            "distance_to_cluster": 3.0,
        },
        {
            "closest_cluster": 2.0,
            "distance_to_cluster": 5.0,
        },

        ....
    ]
}
```

JSONLINES Response Format

```
{"closest_cluster": 1.0, "distance_to_cluster": 3.0}
{"closest_cluster": 2.0, "distance_to_cluster": 5.0}
```

RECORDIO Response Format

```
[
    Record = {
        features = {},
        label = {
            'closest_cluster': {
                keys: [],
                values: [1.0, 2.0]  # float32
            },
            'distance_to_cluster': {
                keys: [],
                values: [3.0, 5.0]  # float32
            },
        }
    }
]
```

K-Nearest Neighbors (k-NN) Algorithm

Amazon SageMaker k-nearest neighbors (k-NN) algorithm is an index-based algorithm. It uses a non-parametric method for classification or regression. For classification problems, the algorithm queries the k points that are closest to the sample point and returns the most frequently used label of their class as the predicted label. For regression problems, the algorithm queries the k closest points to the sample point and returns the average of their feature values as the predicted value.

Training with the k-NN algorithm has three steps: sampling, dimension reduction, and index building. Sampling reduces the size of the initial dataset so that it fits into memory. For dimension reduction, the algorithm decreases the feature dimension of the data to reduce the footprint of the k-NN model in memory and inference latency. We provide two methods of dimension reduction methods: random projection and the fast Johnson-Lindenstrauss transform. Typically, you use dimension reduction for high-dimensional (d >1000) datasets to avoid the "curse of dimensionality" that troubles the statistical analysis of data that becomes sparse as dimensionality increases. The main objective of k-NN's training is to construct the index. The index enables efficient lookups of distances between points whose values or class labels have not yet been determined and the k nearest points to use for inference.

Topics

Input/Output Interface for the k-NN Algorithm

Amazon SageMaker k-NN supports train and test data channels.

- Use a *train channel* for data that you want to sample and construct into the k-NN index.
- Use a *test channel* to emit scores in log files. Scores are listed as one line per mini-batch: accuracy for `classifier`, mean-squared error (mse) for `regressor` for score.

For training inputs, k-NN supports `text/csv` and `application/x-recordio-protobuf` data formats. For input type `text/csv`, the first `label_size` columns are interpreted as the label vector for that row. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference inputs, k-NN supports the `application/json`, `application/x-recordio-protobuf`, and `text/csv` data formats. The `text/csv` format accepts a `label_size` and encoding parameter. It assumes a `label_size` of 0 and a UTF-8 encoding.

For inference outputs, k-NN supports the `application/json` and `application/x-recordio-protobuf` data formats. These two data formats also support a verbose output mode. In verbose output mode, the API provides the search results with the distances vector sorted from smallest to largest, and corresponding elements in the labels vector.

For batch transform, k-NN supports the `application/jsonlines` data format for both input and output. An example input is as follows:

```
content-type: application/jsonlines

{"features": [1.5, 16.0, 14.0, 23.0]}
{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}}
```

An example output is as follows:

```
accept: application/jsonlines

{"predicted_label": 0.0}
{"predicted_label": 2.0}
```

For more information on input and output file formats, see Data Formats for K-nn Training Input (p. 154) for training, K-nn Request and Response Formats (p. 154) for inference, and the k-NN Sample Notebooks (p. 149).

k-NN Sample Notebooks

For a sample notebook that uses the Amazon SageMaker k-nearest neighbor algorithm to predict wilderness cover types from geological and forest service data, see the K-Nearest Neighbor Covertype

. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see . Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How the K-nn Algorithm Works

Step 1: Sample

To specify the total number of data points to be sampled from the training dataset, use the `sample_size`parameter. For example, if the initial dataset has 1,000 data points and the `sample_size` is set to 100, where the total number of instances is 2, each worker would sample 50 points. A total set of 100 data points would be collected. Sampling runs in linear time with respect to the number of data points.

Step 2: Perform Dimension Reduction

The current implementation of k-NN has two methods of dimension reduction. You specify the method in the `dimension_reduction_type` hyperparameter. The `sign` method specifies a random projection, which uses a linear projection using a matrix of random signs, and the `fjlt` method specifies a fast Johnson-Lindenstrauss transform, a method based on the Fourier transform. Both methods preserve the L2 and inner product distances. The `fjlt` method should be used when the target dimension is large and has better performance with CPU inference. The methods differ in their computational complexity. The `sign` method requires O(ndk) time to reduce the dimension of a batch of n points of dimension d into a target dimension k. The `fjlt` method requires O(nd log(d)) time, but the constants involved are larger. Using dimension reduction introduces noise into the data and this noise can reduce prediction accuracy.

Step 3: Build an Index

During inference, the algorithm queries the index for the k-nearest-neighbors of a sample point. Based on the references to the points, the algorithm makes the classification or regression prediction. It makes the prediction based on the class labels or values provided. k-NN provides three different types of indexes: a flat index, an inverted index, and an inverted index with product quantization. You specify the type with the `index_type` parameter.

Serialize the Model

When the k-NN algorithm finishes training, it serializes three files to prepare for inference.

- model_algo-1: Contains the serialized index for computing the nearest neighbors.
- model_algo-1.labels: Contains serialized labels (np.float32 binary format) for computing the predicted label based on the query result from the index.
- model_algo-1.json: Contains the JSON-formatted model metadata which stores the `k` and `predictor_type` hyper-parameters from training for inference along with other relevant state.

With the current implementation of k-NN, you can modify the metadata file to change the way predictions are computed. For example, you can change `k` to 10 or change `predictor_type` to *regressor*.

```
{
  "k": 5,
  "predictor_type": "classifier",
  "dimension_reduction": {"type": "sign", "seed": 3, "target_dim": 10, "input_dim": 20},
  "normalize": False,
```

```
    "version": "1.0"
}
```

EC2 Instance Recommendation for the K-nn Algorithm

Instance Recommendation for Training with the K-nn Algorithm

To start, try running training on a CPU, using, for example, an ml.m5.2xlarge instance, or on a GPU using, for example, an ml.p2.xlarge instance.

Instance Recommendation for Inference with the K-nn Algorithm

Inference requests from CPUs generally have a lower average latency than requests from GPUs because there is a tax on CPU-to-GPU communication when you use GPU hardware. However, GPUs generally have higher throughput for larger batches.

K-nn Hyperparameters

Parameter Name	Description
`feature_dim`	The number of features in the input data. **Required** Valid values: positive integer.
`k`	The number of nearest neighbors. **Required** Valid values: positive integer
`predictor_type`	The type of inference to use on the data labels. **Required** Valid values: *classifier* for classification or *regressor* for regression.
`sample_size`	The number of data points to be sampled from the training data set. **Required** Valid values: positive integer
`dimension_reduction_target`	The target dimension to reduce to. **Required** when you specify the `dimension_reduction_type` parameter. Valid values: positive integer greater than 0 and less than `feature_dim`.
`dimension_reduction_type`	The type of dimension reduction method. **Optional** Valid values: *sign* for random projection or *fjlt* for the fast Johnson-Lindenstrauss transform. Default value: No dimension reduction

Parameter Name	Description
`faiss_index_ivf_nlists`	The number of centroids to construct in the index when `index_type` is *faiss.IVFFlat* or *faiss.IVFPQ*. **Optional** Valid values: positive integer Default value: *auto*, which resolves to `sqrt(sample_size)`.
`faiss_index_pq_m`	The number of vector sub-components to construct in the index when `index_type` is set to *faiss.IVFPQ*. The FaceBook AI Similarity Search (FAISS) library requires that the value of `faiss_index_pq_m` is a divisor of the data dimension. If `faiss_index_pq_m` is not a divisor of the data dimension, we increase the data dimension to smallest integer divisible by `faiss_index_pq_m`. If no dimension reduction is applied, the algorithm adds a padding of zeros. If dimension reduction is applied, the algorithm increase the value of the `dimension_reduction_target` hyper-parameter. **Optional** Valid values: One of the following positive integers: 1, 2, 3, 4, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64, 96
`index_metric`	The metric to measure the distance between points when finding nearest neighbors. When training with `index_type` set to `faiss.IVFPQ`, the `INNER_PRODUCT` distance and `COSINE` similarity are not supported. **Optional** Valid values: *L2* for Euclidean-distance, *INNER_PRODUCT* for inner-product distance, *COSINE* for cosine similarity. Default value: *L2*
`index_type`	The type of index. **Optional** Valid values: *faiss.Flat*, *faiss.IVFFlat*, *faiss.IVFPQ*. Default values: *faiss.Flat*
`mini_batch_size`	The number of observations per mini-batch for the data iterator. **Optional** Valid values: positive integer Default value: 5000

Tune a K-nn Model

The Amazon SageMaker k-nearest neighbors algorithm is a supervised algorithm. The algorithm consumes a test data set and emits a metric about the accuracy for a classification task or about the mean squared error for a regression task. These accuracy metrics compare the model predictions for

their respective task to the ground truth provided by the empirical test data. To find the best model that reports the highest accuracy or lowest error on the test dataset, run a hyperparameter tuning job for k-NN.

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric appropriate for the prediction task of the algorithm. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric. The hyperparameters are used only to help estimate model parameters and are not used by the trained model to make predictions.

For more information about model tuning, see .

Metrics Computed by the K-nn Algorithm

The k-nearest neighbors algorithm computes one of two metrics in the following table during training depending on the type of task specified by the `predictor_type` hyper-parameter.

- *classifier* specifies a classification task and computes `test:accuracy`
- *regressor* specifies a regression task and computes `test:mse`.

Choose the `predictor_type` value appropriate for the type of task undertaken to calculate the relevant objective metric when tuning a model.

Metric Name	Description	Optimization Direction
test:accuracy	When `predictor_type` is set to *classifier*, k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The accuracy reported ranges from 0.0 (0%) to 1.0 (100%).	Maximize
test:mse	When `predictor_type` is set to *regressor*, k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The mean squared error is computed by comparing the two labels.	Minimize

Tunable K-nn Hyperparameters

Tune the Amazon SageMaker k-nearest neighbor model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
k	IntegerParameterRanges	MinValue: 1, MaxValue: 1024
sample_size	IntegerParameterRanges	MinValue: 256, MaxValue: 20000000

Data Formats for K-nn Training Input

All Amazon SageMaker built-in algorithms adhere to the common input training formats described in Common Data Formats - Training. This topic contains a list of the available input formats for the Amazon SageMaker k-nearest-neighbor algorithm.

CSV Data Format

content-type: text/csv; label_size=1

```
4,1.2,1.3,9.6,20.3
```

The first `label_size` columns are interpreted as the label vector for that row.

RECORDIO Data Format

content-type: application/x-recordio-protobuf

```
[
    Record = {
        features = {
            'values': {
                values: [1.2, 1.3, 9.6, 20.3]  # float32
            }
        },
        label = {
            'values': {
                values: [4]  # float32
            }
        }
    }
]


}
```

K-nn Request and Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker k-nearest-neighbor algorithm.

INPUT: CSV Request Format

content-type: text/csv

```
1.2,1.3,9.6,20.3
```

This accepts a `label_size` or encoding parameter. It assumes a `label_size` of 0 and a utf-8 encoding.

INPUT: JSON Request Format

content-type: application/json

```
{
  "instances": [
    {"data": {"features": {"values": [-3, -1, -4, 2]}}},
    {"features": [3.0, 0.1, 0.04, 0.002]}]
```

```
}
```

INPUT: JSONLINES Request Format

content-type: application/jsonlines

```
{"features": [1.5, 16.0, 14.0, 23.0]}
{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}}
```

INPUT: RECORDIO Request Format

content-type: application/x-recordio-protobuf

```
[
    Record = {
        features = {
            'values': {
                values: [-3, -1, -4, 2]   # float32
            }
        },
        label = {}
    },
    Record = {
        features = {
            'values': {
                values: [3.0, 0.1, 0.04, 0.002]   # float32
            }
        },
        label = {}
    },
]
```

OUTPUT: JSON Response Format

accept: application/json

```
{
  "predictions": [
    {"predicted_label": 0.0},
    {"predicted_label": 2.0}
  ]
}
```

OUTPUT: JSONLINES Response Format

accept: application/jsonlines

```
{"predicted_label": 0.0}
{"predicted_label": 2.0}
```

OUTPUT: VERBOSE JSON Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/json; verbose=true

```
{
  "predictions": [
```

```
    {
        "predicted_label": 0.0,
        "distances": [3.11792408, 3.89746071, 6.32548437],
        "labels": [0.0, 1.0, 0.0]
    },
    {
        "predicted_label": 2.0,
        "distances": [1.08470316, 3.04917915, 5.25393973],
        "labels": [2.0, 2.0, 0.0]
    }
  ]
}
```

OUTPUT: RECORDIO-PROTOBUF Response Format

content-type: application/x-recordio-protobuf

```
[
    Record = {
        features = {},
        label = {
            'predicted_label': {
                values: [0.0]  # float32
            }
        }
    },
    Record = {
        features = {},
        label = {
            'predicted_label': {
                values: [2.0]  # float32
            }
        }
    }
]
```

OUTPUT: VERBOSE RECORDIO-PROTOBUF Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/x-recordio-protobuf; verbose=true

```
[
    Record = {
        features = {},
        label = {
            'predicted_label': {
                values: [0.0]  # float32
            },
            'distances': {
                values: [3.11792408, 3.89746071, 6.32548437]  # float32
            },
            'labels': {
                values: [0.0, 1.0, 0.0]  # float32
            }
        }
    },
    Record = {
        features = {},
        label = {
            'predicted_label': {
                values: [0.0]  # float32
```

```
            },
            'distances': {
                values: [1.08470316, 3.04917915, 5.25393973]  # float32
            },
            'labels': {
                values: [2.0, 2.0, 0.0]  # float32
            }
        }
    }
]
```

SAMPLE OUTPUT for the K-nn Algorithm

For regressor tasks:

```
[06/08/2018 20:15:33 INFO 140026520049408] #test_score (algo-1) : ('mse',
 0.013333333333333334)
```

For classifier tasks:

```
[06/08/2018 20:15:46 INFO 140285487171328] #test_score (algo-1) : ('accuracy',
 0.98666666666666669)
```

Latent Dirichlet Allocation (LDA) Algorithm

The Amazon SageMaker Latent Dirichlet Allocation (LDA) algorithm is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of distinct categories. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics. Since the method is unsupervised, the topics are not specified up front, and are not guaranteed to align with how a human may naturally categorize documents. The topics are learned as a probability distribution over the words that occur in each document. Each document, in turn, is described as a mixture of topics.

The exact content of two documents with similar topic mixtures will not be the same. But overall, you would expect these documents to more frequently use a shared subset of words, than when compared with a document from a different topic mixture. This allows LDA to discover these word groups and use them to form topics. As an extremely simple example, given a set of documents where the only words that occur within them are: *eat*, *sleep*, *play*, *meow*, and *bark*, LDA might produce topics like the following:

Topic	*eat*	*sleep*	*play*	*meow*	*bark*
Topic 1	0.1	0.3	0.2	0.4	0.0
Topic 2	0.2	0.1	0.4	0.0	0.3

You can infer that documents that are more likely to fall into Topic 1 are about cats (who are more likely to *meow* and *sleep*), and documents that fall into Topic 2 are about dogs (who prefer to *play* and *bark*). These topics can be found even though the words dog and cat never appear in any of the texts.

Topics

- Input/Output Interface for the LDA Algorithm (p. 158)
- EC2 Instance Recommendation for the LDA Algorithm (p. 158)
- LDA Sample LDA Notebooks (p. 158)
- How LDA Works (p. 158)

Input/Output Interface for the LDA Algorithm

LDA expects data to be provided on the train channel, and optionally supports a test channel, which is scored by the final model. LDA supports both `recordIO-wrapped-protobuf` (dense and sparse) and `CSV` file formats. For `CSV`, the data must be dense and have dimension equal to *number of records * vocabulary size*. LDA can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the `CSV` format.

For inference, `text/csv`, `application/json`, and `application/x-recordio-protobuf` content types are supported. Sparse data can also be passed for `application/json` and `application/x-recordio-protobuf`. LDA inference returns `application/json` or `application/x-recordio-protobuf` *predictions*, which include the `topic_mixture` vector for each observation.

Please see the LDA Sample LDA Notebooks (p. 158) for more detail on training and inference formats.

EC2 Instance Recommendation for the LDA Algorithm

LDA currently only supports single-instance CPU training. CPU instances are recommended for hosting/inference.

LDA Sample LDA Notebooks

For a sample notebook that shows how to train the Amazon SageMaker Latent Dirichlet Allocation algorithm on a dataset and then how to deploy the trained model to perform inferences about the topic mixtures in input documents, see the An Introduction to SageMaker LDA. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How LDA Works

Amazon SageMaker LDA is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of different categories. These categories are themselves a probability distribution over the features. LDA is a generative probability model, which means it attempts to provide a model for the distribution of outputs and inputs based on latent variables. This is opposed to discriminative models, which attempt to learn how inputs map to outputs.

You can use LDA for a variety of tasks, from clustering customers based on product purchases to automatic harmonic analysis in music. However, it is most commonly associated with topic modeling in text corpuses. Observations are referred to as documents. The feature set is referred to as vocabulary. A feature is referred to as a word. And the resulting categories are referred to as topics.

> **Note**
> Lemmatization significantly increases algorithm performance and accuracy. Consider pre-processing any input text data.

An LDA model is defined by two parameters:

- α—A prior estimate on topic probability (in other words, the average frequency that each topic within a given document occurs).
- β—a collection of k topics where each topic is given a probability distribution over the vocabulary used in a document corpus, also called a "topic-word distribution."

LDA is a "bag-of-words" model, which means that the order of words does not matter. LDA is a generative model where each document is generated word-by-word by choosing a topic mixture θ ~ Dirichlet(α).

For each word in the document:

- Choose a topic z ~ Multinomial(θ)
- Choose the corresponding topic-word distribution β_z.
- Draw a word w ~ Multinomial(β_z).

When training the model, the goal is to find parameters α and β, which maximize the probability that the text corpus is generated by the model.

The most popular methods for estimating the LDA model use Gibbs sampling or Expectation Maximization (EM) techniques. The Amazon SageMaker LDA uses tensor spectral decomposition. This provides several advantages:

- **Theoretical guarantees on results**. The standard EM-method is guaranteed to converge only to local optima, which are often of poor quality.
- **Embarrassingly parallelizable**. The work can be trivially divided over input documents in both training and inference. The EM-method and Gibbs Sampling approaches can be parallelized, but not as easily.
- **Fast**. Although the EM-method has low iteration cost it is prone to slow convergence rates. Gibbs Sampling is also subject to slow convergence rates and also requires a large number of samples.

At a high-level, the tensor decomposition algorithm follows this process:

1. The goal is to calculate the spectral decomposition of a **V** x **V** x **V** tensor, which summarizes the moments of the documents in our corpus. **V** is vocabulary size (in other words, the number of distinct words in all of the documents). The spectral components of this tensor are the LDA parameters α and β, which maximize the overall likelihood of the document corpus. However, because vocabulary size tends to be large, this **V** x **V** x **V** tensor is prohibitively large to store in memory.

2. Instead, it uses a **V** x **V** moment matrix, which is the two-dimensional analog of the tensor from step 1, to find a whitening matrix of dimension **V** x **k**. This matrix can be used to convert the **V** x **V** moment matrix into a **k** x **k** identity matrix. **k** is the number of topics in the model.

3. This same whitening matrix can then be used to find a smaller **k** x **k** x **k** tensor. When spectrally decomposed, this tensor has components that have a simple relationship with the components of the **V** x **V** x **V** tensor.

4. *Alternating Least Squares* is used to decompose the smaller **k** x *k* x **k** tensor. This provides a substantial improvement in memory consumption and speed. The parameters α and β can be found by "unwhitening" these outputs in the spectral decomposition.

After the LDA model's parameters have been found, you can find the topic mixtures for each document. You use stochastic gradient descent to maximize the likelihood function of observing a given topic mixture corresponding to these data.

Topic quality can be improved by increasing the number of topics to look for in training and then filtering out poor quality ones. This is in fact done automatically in Amazon SageMaker LDA: 25% more topics are computed and only the ones with largest associated Dirichlet priors are returned. To perform further topic filtering and analysis, you can increase the topic count and modify the resulting LDA model as follows:

```
> import mxnet as mx
> alpha, beta = mx.ndarray.load('model.tar.gz')
```

```
> # modify alpha and beta
> mx.nd.save('new_model.tar.gz', [new_alpha, new_beta])
> # upload to S3 and create new SageMaker model using the console
```

For more information about algorithms for LDA and the Amazon SageMaker implementation, see the following:

- Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. *Tensor Decompositions for Learning Latent Variable Models*, Journal of Machine Learning Research, 15:2773–2832, 2014.
- David M Blei, Andrew Y Ng, and Michael I Jordan. *Latent Dirichlet Allocation*. Journal of Machine Learning Research, 3(Jan):993–1022, 2003.
- Thomas L Griffiths and Mark Steyvers. *Finding Scientific Topics*. Proceedings of the National Academy of Sciences, 101(suppl 1):5228–5235, 2004.
- Tamara G Kolda and Brett W Bader. *Tensor Decompositions and Applications*. SIAM Review, 51(3):455–500, 2009.

LDA Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the LDA training algorithm provided by Amazon SageMaker. For more information, see How LDA Works (p. 158).

Parameter Name	Description
num_topics	The number of topics for LDA to find within the data. **Required** Valid values: positive integer
feature_dim	The size of the vocabulary of the input document corpus. **Required** Valid values: positive integer
mini_batch_size	The total number of documents in the input document corpus. **Required** Valid values: positive integer
alpha0	Initial guess for the concentration parameter: the sum of the elements of the Dirichlet prior. Small values are more likely to generate sparse topic mixtures and large values (greater than 1.0) produce more uniform mixtures. **Optional** Valid values: Positive float Default value: 0.1
max_restarts	The number of restarts to perform during the Alternating Least Squares (ALS) spectral decomposition phase of the algorithm.

Parameter Name	Description
	Can be used to find better quality local minima at the expense of additional computation, but typically should not be adjusted. **Optional** Valid values: Positive integer Default value: 10
`max_iterations`	The maximum number of iterations to perform during the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted. **Optional** Valid values: Positive integer Default value: 1000
`tol`	Target error tolerance for the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted. **Optional** Valid values: Positive float Default value: 1e-8

Tune an LDA Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

LDA is an unsupervised topic modeling algorithm that attempts to describe a set of observations (documents) as a mixture of different categories (topics). The "per-word log-likelihood" (PWLL) metric measures the likelihood that a learned set of topics (an LDA model) accurately describes a test document dataset. Larger values of PWLL indicate that the test data is more likely to be described by the LDA model.

For more information about model tuning, see .

Metrics Computed by the LDA Algorithm

The LDA algorithm reports on a single metric during training: `test:pwll`. When tuning a model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
`test:pwll`	Per-word log-likelihood on the test dataset. The likelihood that the test dataset is accurately described by the learned LDA model.	Maximize

Tunable LDA Hyperparameters

You can tune the following hyperparameters for the LDA algorithm. Both hyperparameters, `alpha0` and `num_topics`, can affect the LDA objective metric (`test:pwll`). If you don't already know the optimal values for these hyperparameters, which maximize per-word log-likelihood and produce an accurate LDA model, automatic model tuning can help find them.

Parameter Name	Parameter Type	Recommended Ranges
`alpha0`	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 10
`num_topics`	IntegerParameterRanges	MinValue: 1, MaxValue: 150

Linear Learner Algorithm

Linear models are supervised learning algorithms used for solving either classification or regression problems. For input, you give the model labeled examples (*x*, *y*). *x* is a high-dimensional vector and *y* is a numeric label. For binary classification problems, the label must be either 0 or 1. For multiclass classification problems, the labels must be from 0 to `num_classes` - 1. For regression problems, *y* is a real number. The algorithm learns a linear function, or, for classification problems, a linear threshold function, and maps a vector *x* to an approximation of the label *y*.

The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. With the Amazon SageMaker algorithm, you can simultaneously explore different training objectives and choose the best solution from a validation set. You can also explore a large number of models and choose the best. The best model optimizes either of the following:

- Continuous objectives, such as mean square error, cross entropy loss, absolute error.
- Discrete objectives suited for classification, such as F1 measure, precision, recall, or accuracy.

Compared with methods that provide a solution for only continuous objectives, the Amazon SageMaker linear learner algorithm provides a significant increase in speed over naive hyperparameter optimization techniques. It is also more convenient.

The linear learner algorithm requires a data matrix, with rows representing the observations, and columns representing the dimensions of the features. It also requires an additional column that contains the labels that match the data points. At a minimum, Amazon SageMaker linear learner requires you to specify input and output data locations, and objective type (classification or regression) as arguments. The feature dimension is also required. For more information, see CreateTrainingJob (p. 667). You can specify additional parameters in the `HyperParameters` string map of the request body. These parameters control the optimization procedure, or specifics of the objective function that you train on. For example, the number of epochs, regularization, and loss type.

Topics

- Linear Learner Response Formats (p. 175)

Input/Output Interface for the Linear Learner Algorithm

The Amazon SageMaker linear learner algorithm supports three data channels: train, validation (optional), and test (optional). If you provide validation data, it should be `FullyReplicated`. The algorithm logs validation loss at every epoch, and uses a sample of the validation data to calibrate and select the best model. If you don't provide validation data, the algorithm uses a sample of the training data to calibrate and select the model. If you provide test data, the algorithm logs include the test score for the final model.

For training, the linear learner algorithm supports both `recordIO-wrapped protobuf` and `CSV` formats. For the `application/x-recordio-protobuf` input type, only Float32 tensors are supported. For the `text/csv` input type, the first column is assumed to be the label, which is the target variable for prediction. You can use either File mode or Pipe mode to train linear learner models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, the linear learner algorithm supports the `application/json`, `application/x-recordio-protobuf`, and `text/csv` formats. When you make predictions on new data, the format of the response depends on the type of model. **For regression** (`predictor_type='regressor'`), the `score` is the prediction produced by the model. **For classification** (`predictor_type='binary_classifier'` or `predictor_type='multiclass_classifier'`), the model returns a `score` and also a `predicted_label`. The `predicted_label` is the class predicted by the model and the `score` measures the strength of that prediction.

- **For binary classification**, `predicted_label` is `0` or `1`, and score is a single floating point number correspond class.
- **For multiclass classification**, the `predicted_class` will be an integer from `0` to `num_classes-1`, and the score will be a list of one floating point number per class.

To interpret the `score` in classification problems, you have to consider the loss function used. If the `loss` hyperparameter value is `logistic` for binary classification or `softmax_loss` for multiclass classification, then the `score` can be interpreted as the probability of the corresponding class. These are the loss values used by the linear learner when the `loss` value is `auto` default value. But if the loss is set to `hinge_loss`, then the score cannot be interpreted as a probability. This is because hinge loss corresponds to a Support Vector Classifier, which does not produce probability estimates.

For more information on input and output file formats, see Linear Learner Response Formats (p. 175). For more information on inference formats, and the Linear Learner Sample Notebooks (p. 163).

EC2 Instance Recommendation for the Linear Learner Algorithm

You can train the linear learner algorithm on single- or multi-machine CPU and GPU instances. During testing, we have not found substantial evidence that multi-GPU computers are faster than single-GPU computers. Results can vary, depending on your specific use case.

Linear Learner Sample Notebooks

For a sample notebook that uses the Amazon SageMaker linear learner algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to Linear Learner with MNIST. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all of the Amazon SageMaker samples. The topic modeling example notebooks using the linear learning algorithm are

located in the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab and choose **Create copy**.

How Linear Learner Works

There are three steps involved in the implementation of the linear learner algorithm: preprocess, train, and validate.

Step 1: Preprocess

Normalization, or feature scaling, is an important preprocessing step for certain loss functions that ensures the model being trained on a dataset does not become dominated by the weight of a single feature. The Amazon SageMaker Linear Learner algorithm has a normalization option to assist with this preprocessing step. If normalization is turned on, the algorithm first goes over a small sample of the data to learn the mean value and standard deviation for each feature and for the label. Each of the features in the full dataset is then shifted to have mean of zero and scaled to have a unit standard deviation.

> **Note**
> For best results, ensure your data is shuffled before training. Training with unshuffled data may cause training to fail.

You can configure whether the linear learner algorithm normalizes the feature data and the labels using the `normalize_data` and `normalize_label`) hyperparameters respectively. Normalization is enabled by default for both features and labels for regression. Only the features can be normalized for binary classification and this is the default behavior.

Step 2: Train

With the linear learner algorithm, you train with a distributed implementation of stochastic gradient descent (SGD). You can control the optimization process by choosing the optimization algorithm. For example, you can choose to use Adam, AdaGrad, stochastic gradient descent, or other optimization algorithms. You also specify their hyperparameters, such as momentum, learning rate, and the learning rate schedule. If you aren't sure which algorithm or hyperparameter value to use, choose a default that works for the majority of datasets.

During training, you simultaneously optimize multiple models, each with slightly different objectives. For example, you vary L1 or L2 regularization and try out different optimizer settings.

Step 3: Validate and Set the Threshold

When training multiple models in parallel, the models are evaluated against a validation set to select the most optimal model once training is complete. For regression, the most optimal model is the one that achieves the best loss on the validation set. For classification, a sample of the validation set is used to calibrate the classification threshold. The most optimal model selected is the one that achieves the best binary classification selection criteria on the validation set. Examples of such such criteria include the F1 measure, accuracy, and cross-entropy loss.

> **Note**
> If the algorithm is not provided a validation set, then evaluating and selecting the most optimal model is not possible. To take advantage of parallel training and model selection ensure you provide a validation set to the algorithm.

Linear Learner Hyperparameters

The following table contains the hyperparameters for the learner learner algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The

required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order.

Parameter Name	Description
feature_dim	The number of features in the input data. **Required** Valid values: Positive integer
num_classes	The number of classes for the response variable. The algorithm assumes that classes are labeled 0, ..., num_classes - 1. **Required** when predictor_type is multiclass_classifier. Otherwise, the algorithm ignores it. Valid values: Integers from 3 to 1,000,000
predictor_type	Specifies the type of target variable as a binary classification, multiclass classification, or regression. **Required** Valid values: binary_classifier, multiclass_classifier, or regressor
accuracy_top_k	When computing the top-k accuracy metric for multiclass classification, the value of *k*. If the model assigns one of the top-k scores to the true label, an example is scored as correct. **Optional** Valid values: Positive integers Default value: 3
balance_multiclass_weights	Specifies whether to use class weights, which give each class equal importance in the loss function. Used only when the predictor_type is multiclass_classifier. **Optional** Valid values: true, false Default value: false
beta_1	The exponential decay rate for first-moment estimates. Applies only when the optimizer value is adam. **Optional** Valid values: auto or floating-point value between 0 and 1.0 Default value: auto
beta_2	The exponential decay rate for second-moment estimates. Applies only when the optimizer value is adam. **Optional**

Parameter Name	Description
	Valid values: `auto` or floating-point integer between 0 and 1.0 Default value: `auto`
`bias_lr_mult`	Allows a different learning rate for the bias term. The actual learning rate for the bias is `learning_rate` * `bias_lr_mult`. **Optional** Valid values: `auto` or positive floating-point integer Default value: `auto`
`bias_wd_mult`	Allows different regularization for the bias term. The actual L2 regularization weight for the bias is `wd` * `bias_wd_mult`. By default, there is no regularization on the bias term. **Optional** Valid values: `auto` or non-negative floating-point integer Default value: `auto`
`binary_classifier_model_selection_criteria`	When `predictor_type` is set to `binary_classifier`, the model evaluation criteria for the validation dataset (or for the training dataset if you don't provide a validation dataset). Criteria include: • `accuracy`—The model with the highest accuracy. • `f_beta`—The model with the highest F1 score. The default is F1. • `precision_at_target_recall`—The model with the highest precision at a given recall target. • `recall_at_target_precision`—The model with the highest recall at a given precision target. • `loss_function`—The model with the lowest value of the loss function used in training. **Optional** Valid values: `accuracy`, `f_beta`, `precision_at_target_recall`, `recall_at_target_precision`, or `loss_function` Default value: `accuracy`

Parameter Name	Description
`early_stopping_patience`	If no improvement is made in the relevant metric, the number of epochs to wait before ending training. If you have provided a value for `binary_classifier_model_selection_criteria`. the metric is that value. Otherwise, the metric is the same as the value specified for the `loss` hyperparameter. The metric is evaluated on the validation data. If you haven't provided validation data, the metric is always the same as the value specified for the `loss` hyperparameter and is evaluated on the training data. To disable early stopping, set `early_stopping_patience` to a value greater than the value specified for `epochs`. **Optional** Valid values: Positive integer Default value: 3
`early_stopping_tolerance`	The relative tolerance to measure an improvement in loss. If the ratio of the improvement in loss divided by the previous best loss is smaller than this value, early stopping considers the improvement to be zero. **Optional** Valid values: Positive floating-point integer Default value: 0.001
`epochs`	The maximum number of passes over the training data. **Optional** Valid values: Positive integer Default value: 15
`f_beta`	The value of beta to use when calculating F score metrics for binary or multiclass classification. Also used if the value specified for `binary_classifier_model_selection_criteria` is `f_beta`. **Optional** Valid values: Positive floating-point integers Default value: 1.0
`huber_delta`	The parameter for Huber loss. During training and metric evaluation, compute L2 loss for errors smaller than delta and L1 loss for errors larger than delta. **Optional** Valid values: Positive floating-point integer Default value: 1.0

Parameter Name	Description
`init_bias`	Initial weight for the bias term. **Optional** Valid values: Floating-point integer Default value: 0
`init_method`	Sets the initial distribution function used for model weights. Functions include: • `uniform`—Uniformly distributed between (-scale, +scale) • `normal`—Normal distribution, with mean 0 and sigma **Optional** Valid values: `uniform` or `normal` Default value: `uniform`
`init_scale`	Scales an initial uniform distribution for model weights. Applies only when the `init_method` hyperparameter is set to `uniform`. **Optional** Valid values: Positive floating-point integer Default value: 0.07
`init_sigma`	The initial standard deviation for the normal distribution. Applies only when the `init_method` hyperparameter is set to `normal`. **Optional** Valid values: Positive floating-point integer Default value: 0.01
`l1`	The L1 regularization parameter. If you don't want to use L1 regularization, set the value to 0. **Optional** Valid values: `auto` or non-negative float Default value: `auto`
`learning_rate`	The step size used by the optimizer for parameter updates. **Optional** Valid values: `auto` or positive floating-point integer Default value: `auto`, whose value depends on the optimizer chosen.

Parameter Name	Description
`loss`	Specifies the loss function. The available loss functions and their default values depend on the value of `predictor_type`: - If the `predictor_type` is set to `regressor`, the available options are `auto`, `squared_loss`, `absolute_loss`, `eps_insensitive_squared_loss`, `eps_insensitive_absolute_loss`, `quantile_loss`, and `huber_loss`. The default value for `auto` is `squared_loss`. - If the `predictor_type` is set to `binary_classifier`, the available options are `auto`,`logistic`, and `hinge_loss`. The default value for `auto` is `logistic`. - If the `predictor_type` is set to `multiclass_classifier`, the available options are `auto` and `softmax_loss`. The default value for `auto` is `softmax_loss`. Valid values: `auto`, `logistic`, `squared_loss`, `absolute_loss`, `hinge_loss`, `eps_insensitive_squared_loss`, `eps_insensitive_absolute_loss`, `quantile_loss`, or `huber_loss` **Optional** Default value: `auto`
`loss_insensitivity`	The parameter for the epsilon-insensitive loss type. During training and metric evaluation, any error smaller than this value is considered to be zero. **Optional** Valid values: Positive floating-point integer Default value: 0.01
`lr_scheduler_factor`	For every `lr_scheduler_step` hyperparameter, the learning rate decreases by this quantity. Applies only when the `use_lr_scheduler` hyperparameter is set to `true`. **Optional** Valid values: `auto` or positive floating-point integer between 0 and 1 Default value: `auto`
`lr_scheduler_minimum_lr`	The learning rate never decreases to a value lower than the value set for `lr_scheduler_minimum_lr`. Applies only when the `use_lr_scheduler` hyperparameter is set to `true`. **Optional** Valid values: `auto` or positive floating-point integer Default values: `auto`

Parameter Name	Description
lr_scheduler_step	The number of steps between decreases of the learning rate. Applies only when the use_lr_scheduler hyperparameter is set to true. **Optional** Valid values: auto or positive integer Default value: auto
margin	The margin for the hinge_loss function. **Optional** Valid values: Positive floating-point integer Default value: 1.0
mini_batch_size	The number of observations per mini-batch for the data iterator. **Optional** Valid values: Positive integer Default value: 1000
momentum	The momentum of the sgd optimizer. **Optional** Valid values: auto or a floating-point integer between 0 and 1.0 Default value: auto
normalize_data	Normalizes the feature data before training. Data normalization shifts the data for each feature to have a mean of zero and scales it to have unit standard deviation. **Optional** Valid values: auto, true, or false Default value: true
normalize_label	Normalizes the label. Label normalization shifts the label to have a mean of zero and scales it to have unit standard deviation. The auto default value normalizes the label for regression problems but does not for classification problems. If you set the normalize_label hyperparameter to true for classification problems, the algorithm ignores it. **Optional** Valid values: auto, true, or false Default value: auto

Parameter Name	Description
num_calibration_samples	The number of observations from the validation dataset to use for model calibration (when finding the best threshold). **Optional** Valid values: `auto` or positive integer Default value: `auto`
num_models	The number of models to train in parallel. For the default, `auto`, the algorithm decides the number of parallel models to train. One model is trained according to the given training parameter (regularization, optimizer, loss), and the rest by close parameters. **Optional** Valid values: `auto` or positive integer Default values: `auto`
num_point_for_scaler	The number of data points to use for calculating normalization or unbiasing of terms. **Optional** Valid values: Positive integer Default value: 10,000
optimizer	The optimization algorithm to use. **Optional** Valid values: `auto`—The default value. `sgd`—Stochastic gradient descent. `adam`—Adaptive momentum estimation. `rmsprop`—A gradient-based optimization technique that uses a moving average of squared gradients to normalize the gradient. Default value: `auto`. The default setting for `auto` is `adam`.
positive_example_weight_mult	The weight assigned to positive examples when training a binary classifier. The weight of negative examples is fixed at 1. If you want the algorithm to choose a weight so that errors in classifying negative *vs.* positive examples have equal impact on training loss, specify `balanced`. If you want the algorithm to choose the weight that optimizes performance, specify `auto`. **Optional** Valid values: `balanced`, `auto`, or a positive floating-point integer Default value: 1.0

Parameter Name	Description
`quantile`	The quantile for quantile loss. For quantile q, the model attempts to produce predictions so that the value of `true_label` is greater than the prediction with probability q. **Optional** Valid values: Floating-point integer between 0 and 1 Default value: 0.5
`target_precision`	The target precision. If `binary_classifier_model_selection_criteria` is `recall_at_target_precision`, then precision is held at this value while recall is maximized. **Optional** Valid values: Floating-point integer between 0 and 1.0 Default value: 0.8
`target_recall`	The target recall. If `binary_classifier_model_selection_criteria` is `precision_at_target_recall`, then recall is held at this value while precision is maximized. **Optional** Valid values: Floating-point integer between 0 and 1.0 Default value: 0.8
`unbias_data`	Unbiases the features before training so that the mean is 0. By default. data is unbiased if the `use_bias` hyperparameter is set to `true`. **Optional** Valid values: `auto`, `true`, or `false` Default value: `auto`
`unbias_label`	Unbiases labels before training so that the mean is 0. Applies to regression only if the `use_bias` hyperparameter is set to `true`. **Optional** Valid values: `auto`, `true`, or `false` Default value: `auto`
`use_bias`	Specifies whether the model should include a bias term, which is the intercept term in the linear equation. **Optional** Valid values: `true` or `false` Default value: `true`

Parameter Name	Description
use_lr_scheduler	Whether to use a scheduler for the learning rate. If you want to use a scheduler, specify `true`. **Optional** Valid values: `true` or `false` Default value: `true`
wd	The weight decay parameter, also known as the L2 regularization parameter. If you don't want to use L2 regularization, set the value to 0. **Optional** Valid values:`auto` or non-negative floating-point integer Default value: `auto`

Tune a Linear Learner Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The linear learner algorithm also has an internal mechanism for tuning hyperparameters separate from the automatic model tuning feature described here. By default, the linear learner algorithm tunes hyperparameters by training multiple models in parallel. When you use automatic model tuning, the linear learner internal tuning mechanism is turned off automatically. This sets the number of parallel models, `num_models`, to 1. The algorithm ignores any value that you set for `num_models`.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the Linear Learner Algorithm

The linear learner algorithm reports the metrics in the following table, which are computed during training. Choose one of them as the objective metric. To avoid overfitting, we recommend tuning the model against a validation metric instead of a training metric.

Metric Name	Description	Optimization Direction
test:objective_loss	The mean value of the objective loss function on the test dataset after the model is trained. By default, the loss is logistic loss for binary classification and squared loss for regression. To set the loss to other types, use the `loss` hyperparameter.	Minimize
test:binary_classification_ accuracy	The accuracy of the final model on the test dataset.	Maximize
test:binary_f_beta	The F_beta score of the final model on the test dataset. By default, it is the F1 score, which is the harmonic mean of precision and recall.	Maximize

Metric Name	Description	Optimization Direction
`test:precision`	The precision of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the `binary_classifier_model_selection` hyperparameter to `precision_at_target_recall` and setting the value for the `target_recall` hyperparameter.	Maximize
`test:recall`	The recall of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the `binary_classifier_model_selection` hyperparameter to `recall_at_target_precision` and setting the value for the `target_precison` hyperparameter.	Maximize
`validation:objective_loss`	The mean value of the objective loss function on the validation dataset every epoch. By default, the loss is logistic loss for binary classification and squared loss for regression. To set loss to other types, use the `loss` hyperparameter.	Minimize
`validation:binary_classification_accuracy`	The accuracy of the final model on the validation dataset.	Maximize
`validation:binary_f_beta`	The F_beta score of the final model on the validation dataset. By default, the F_beta score is the F1 score, which is the harmonic mean of the `validation:precision` and `validation:recall` metrics.	Maximize
`validation:precision`	The precision of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the `binary_classifier_model_selection` hyperparameter to `precision_at_target_recall` and setting the value for the `target_recall` hyperparameter.	Maximize
`validation:recall`	The recall of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the `binary_classifier_model_selection` hyperparameter to `recall_at_target_precision` and setting the value for the `target_precison` hyperparameter.	Maximize

Tuning Linear Learner Hyperparameters

You can tune a linear learner model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
wd	ContinuousParameterRanges	MinValue: 1e-7, MaxValue: 1
l1	ContinuousParameterRanges	MinValue: 1e-7, MaxValue: 1
learning_rate	ContinuousParameterRanges	MinValue: 1e-5, MaxValue: 1
mini_batch_size	IntegerParameterRanges	MinValue: 100, MaxValue: 5000
use_bias	CategoricalParameterRanges	[True, False]
positive_example_weight_mult	ContinuousParameterRanges	MinValue: 1e-5, MaxValue: 1e5

Linear Learner Response Formats

JSON Response Format

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. The following are the available output formats for the Amazon SageMaker linear learner algorithm.

Binary Classification

```
let response =   {
    "predictions":    [
        {
            "score": 0.4,
            "predicted_label": 0
        }
    ]
}
```

Multiclass Classification

```
let response =   {
    "predictions":    [
        {
            "score": [0.1, 0.2, 0.4, 0.3],
            "predicted_label": 2
        }
    ]
}
```

Regression

```
let response =   {
    "predictions":    [
        {
            "score": 0.4
        }
    ]
}
```

JSONLINES Response Format

Binary Classification

```
{"score": 0.4, "predicted_label": 0}
```

Multiclass Classification

```
{"score": [0.1, 0.2, 0.4, 0.3], "predicted_label": 2}
```

Regression

```
{"score": 0.4}
```

RECORDIO Response Format

Binary Classification

```
[
    Record = {
        features = {},
        label = {
            'score': {
                keys: [],
                values: [0.4]  # float32
            },
            'predicted_label': {
                keys: [],
                values: [0.0]  # float32
            }
        }
    }
]
```

Multiclass Classification

```
[
    Record = {
    "features": [],
    "label":     {
        "score":   {
                "values":   [0.1, 0.2, 0.3, 0.4]
        },
        "predicted_label":  {
                "values":   [3]
        }
     },
    "uid":  "abc123",
    "metadata": "{created_at: '2017-06-03'}"
  }
]
```

Regression

```
[
    Record = {
        features = {},
        label = {
            'score': {
                keys: [],
```

```
                values: [0.4]   # float32
            }
        }
    }
]
```

Neural Topic Model (NTM) Algorithm

Amazon SageMaker NTM is an unsupervised learning algorithm that is used to organize a corpus of documents into *topics* that contain word groupings based on their statistical distribution. Documents that contain frequent occurrences of words such as "bike", "car", "train", "mileage", and "speed" are likely to share a topic on "transportation" for example. Topic modeling can be used to classify or summarize documents based on the topics detected or to retrieve information or recommend content based on topic similarities. The topics from documents that NTM learns are characterized as a *latent representation* because the topics are inferred from the observed word distributions in the corpus. The semantics of topics are usually inferred by examining the top ranking words they contain. Because the method is unsupervised, only the number of topics, not the topics themselves, are prespecified. In addition, the topics are not guaranteed to align with how a human might naturally categorize documents.

Topic modeling provides a way to visualize the contents of a large document corpus in terms of the learned topics. Documents relevant to each topic might be indexed or searched for based on their soft topic labels. The latent representations of documents might also be used to find similar documents in the topic space. You can also use the latent representations of documents that the topic model learns for input to another supervised algorithm such as a document classifier. Because the latent representations of documents are expected to capture the semantics of the underlying documents, algorithms based in part on these representations are expected to perform better than those based on lexical features alone.

Although you can use both the Amazon SageMaker NTM and LDA algorithms for topic modeling, they are distinct algorithms and can be expected to produce different results on the same input data.

For more information on the mathematics behind NTM, see Neural Variational Inference for Text Processing.

Topics

Input/Output Interface for the NTM Algorithm

Amazon SageMaker Neural Topic Model supports four data channels: train, validation, test, and auxiliary. The validation, test, and auxiliary data channels are optional. If you specify any of these optional channels, set the value of the `S3DataDistributionType` parameter for them to `FullyReplicated`. If you provide validation data, the loss on this data is logged at every epoch, and the model stops training as soon as it detects that the validation loss is not improving. If you don't provide validation data, the algorithm stops early based on the training data, but this can be less efficient. If you provide test data, the algorithm reports the test loss from the final model.

The train, validation, and test data channels for NTM support both `recordIO-wrapped-protobuf` (dense and sparse) and `CSV` file formats. For `CSV` format, each row must be represented densely with zero counts for words not present in the corresponding document, and have dimension equal to: (number of records) * (vocabulary size). You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`. The auxiliary channel is used to

supply a text file that contains vocabulary. By supplying the vocabulary file, users are able to see the top words for each of the topics printed in the log instead of their integer IDs. Having the vocabulary file also allows NTM to compute the Word Embedding Topic Coherence (WETC) scores, a new metric displayed in the log that captures similarity among the top words in each topic effectively. The `ContentType` for the auxiliary channel is `text/plain`, with each line containing a single word, in the order corresponding to the integer IDs provided in the data. The vocabulary file must be named `vocab.txt` and currently only UTF-8 encoding is supported.

For inference, `text/csv`, `application/json`, `application/jsonlines`, and `application/x-recordio-protobuf` content types are supported. Sparse data can also be passed for `application/json` and `application/x-recordio-protobuf`. NTM inference returns `application/json` or `application/x-recordio-protobuf` *predictions*, which include the `topic_weights` vector for each observation.

See the blog post and the companion notebook for more details on using the auxiliary channel and the WETC scores. For more information on how to compute the WETC score, see Coherence-Aware Neural Topic Modeling. We used the pairwise WETC described in this paper for the Amazon SageMaker Neural Topic Model.

For more information on input and output file formats, see NTM Response Formats (p. 182) for inference and the NTM Sample Notebooks (p. 178).

EC2 Instance Recommendation for the NTM Algorithm

NTM training supports both GPU and CPU instance types. We recommend GPU instances, but for certain workloads, CPU instances may result in lower training costs. CPU instances should be sufficient for inference.

NTM Sample Notebooks

For a sample notebook that uses the Amazon SageMaker NTM algorithm to uncover topics in documents from a synthetic data source where the topic distributions are known, see the Introduction to Basic Functionality of NTM. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

NTM Hyperparameters

Parameter Name	Description
`feature_dim`	The vocabulary size of the dataset. **Required** Valid values: Positive integer (min: 1, max: 1,000,000)
`num_topics`	The number of required topics. **Required** Valid values: Positive integer (min: 2, max: 1000)
`batch_norm`	Whether to use batch normalization during training. **Optional**

Parameter Name	Description
	Valid values: *true* or *false*
	Default value: *false*
clip_gradient	The maximum magnitude for each gradient component.
	Optional
	Valid values: Float (min: 1e-3)
	Default value: Infinity
encoder_layers	The number of layers in the encoder and the output size of each layer. When set to *auto*, the algorithm uses two layers of sizes 3 x `num_topics` and 2 x `num_topics` respectively.
	Optional
	Valid values: Comma-separated list of positive integers or *auto*
	Default value: *auto*
encoder_layers_activation	The activation function to use in the encoder layers.
	Optional
	Valid values:
	`sigmoid`: Sigmoid function `tanh`: Hyperbolic tangent `relu`: Rectified linear unit
	Default value: `sigmoid`
epochs	The maximum number of passes over the training data.
	Optional
	Valid values: Positive integer (min: 1)
	Default value: 50
learning_rate	The learning rate for the optimizer.
	Optional
	Valid values: Float (min: 1e-6, max: 1.0)
	Default value: 0.001
mini_batch_size	The number of examples in each mini batch.
	Optional
	Valid values: Positive integer (min: 1, max: 10000)
	Default value: 256

Parameter Name	Description
num_patience_epochs	The number of successive epochs over which early stopping criterion is evaluated. Early stopping is triggered when the change in the loss function drops below the specified `tolerance` within the last `num_patience_epochs` number of epochs. To disable early stopping, set `num_patience_epochs` to a value larger than `epochs`. **Optional** Valid values: Positive integer (min: 1) Default value: 3
optimizer	The optimizer to use for training. **Optional** Valid values: • `sgd`: Stochastic gradient descent • `adam`: Adaptive momentum estimation • `adagrad`: Adaptive gradient algorithm • `adadelta`: An adaptive learning rate algorithm • `rmsprop`: Root mean square propagation Default value: `adadelta`
rescale_gradient	The rescale factor for gradient. **Optional** Valid values: float (min: 1e-3, max: 1.0) Default value: 1.0
sub_sample	The fraction of the training data to sample for training per epoch. **Optional** Valid values: Float (min: 0.0, max: 1.0) Default value: 1.0
tolerance	The maximum relative change in the loss function. Early stopping is triggered when change in the loss function drops below this value within the last `num_patience_epochs` number of epochs. **Optional** Valid values: Float (min: 1e-6, max: 0.1) Default value: 0.001

Parameter Name	Description
`weight_decay`	The weight decay coefficient. Adds L2 regularization. **Optional** Valid values: Float (min: 0.0, max: 1.0) Default value: 0.0

Tune an NTM Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

Amazon SageMaker NTM is an unsupervised learning algorithm that learns latent representations of large collections of discrete data, such as a corpus of documents. Latent representations use inferred variables that are not directly measured to model the observations in a dataset. Automatic model tuning on NTM helps you find the model that minimizes loss over the training or validation data. *Training loss* measures how well the model fits the training data. *Validation loss* measures how well the model can generalize to data that it is not trained on. Low training loss indicates that a model is a good fit to the training data. Low validation loss indicates that a model has not overfit the training data and so should be able to model documents on which is has not been trained successfully. Usually, it's preferable to have both losses be small. However, minimizing training loss too much might result in overfitting and increase validation loss, which would reduce the generality of the model.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the NTM Algorithm

The NTM algorithm reports a single metric that is computed during training: `validation:total_loss`. The total loss is the sum of the reconstruction loss and Kullback-Leibler divergence. When tuning hyperparameter values, choose this metric as the objective.

Metric Name	Description	Optimization Direction
`validation:total_loss`	Total Loss on validation set	Minimize

Tunable NTM Hyperparameters

You can tune the following hyperparameters for the NTM algorithm. Usually setting low `mini_batch_size` and small `learning_rate` values results in lower validation losses, although it might take longer to train. Low validation losses don't necessarily produce more coherent topics as interpreted by humans. The effect of other hyperparameters on training and validation loss can vary from dataset to dataset. To see which values are compatible, see NTM Hyperparameters (p. 178).

Parameter Name	Parameter Type	Recommended Ranges
`encoder_layers_activation`	CategoricalParameterRanges	['sigmoid', 'tanh', 'relu']
`learning_rate`	ContinuousParameterRange	MinValue: 1e-4, MaxValue: 0.1

Parameter Name	Parameter Type	Recommended Ranges
`mini_batch_size`	IntegerParameterRanges	MinValue: 16, MaxValue:2048
`optimizer`	CategoricalParameterRanges	['sgd', 'adam', 'adadelta']
`rescale_gradient`	ContinuousParameterRange	MinValue: 0.1, MaxValue: 1.0
`weight_decay`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0

NTM Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker NTM algorithm.

JSON Response Format

```
{
    "predictions":    [
        {"topic_weights": [0.02, 0.1, 0,...]},
        {"topic_weights": [0.25, 0.067, 0,...]}
    ]
}
```

JSONLINES Response Format

```
{"topic_weights": [0.02, 0.1, 0,...]}
{"topic_weights": [0.25, 0.067, 0,...]}
```

RECORDIO Response Format

```
[
    Record = {
        features = {},
        label = {
            'topic_weights': {
                keys: [],
                values: [0.25, 0.067, 0, ...]  # float32
            }
        }
    },
    Record = {
        features = {},
        label = {
            'topic_weights': {
                keys: [],
                values: [0.25, 0.067, 0, ...]  # float32
            }
        }
    }
]
```

Object2Vec Algorithm

The Amazon SageMaker Object2Vec algorithm is a general-purpose neural embedding algorithm that is highly customizable. It can learn low-dimensional dense embeddings of high-dimensional objects. The embeddings are learned in a way that preserves the semantics of the relationship between pairs of objects in the original space in the embedding space. You can use the learned embeddings to efficiently compute nearest neighbors of objects and to visualize natural clusters of related objects in low-dimensional space, for example. You can also use the embeddings as features of the corresponding objects in downstream supervised tasks, such as classification or regression.

Object2Vec generalizes the well-known Word2Vec embedding technique for words that is optimized in the Amazon SageMaker BlazingText Algorithm (p. 74). For a blog post that discusses how to apply Object2Vec to some practical use cases, see Introduction to Amazon SageMaker Object2Vec.

Topics

I/O Interface for the Object2Vec Algorithm

You can use Object2Vec on many input data types, including the following examples.

Input Data Type	Example
Sentence-sentence pairs	"A soccer game with multiple males playing." and "Some men are playing a sport."
Labels-sequence pairs	The genre tags of the movie "Titanic", such as "Romance" and "Drama", and its short description: "James Cameron's Titanic is an epic, action-packed romance set against the ill-fated maiden voyage of the R.M.S. Titanic. She was the most luxurious liner of her era, a ship of dreams, which ultimately carried over 1,500 people to their death in the ice cold waters of the North Atlantic in the early hours of April 15, 1912."
Customer-customer pairs	The customer ID of Jane and customer ID of Jackie.
Product-product pairs	The product ID of football and product ID of basketball.
Item review user-item pairs	A user's ID and the items she has bought, such as apple, pear, and orange.

To transform the input data into the supported formats, you must preprocess it. Currently, Object2Vec natively supports two types of input:

- A discrete token, which is represented as a list of a single `integer-id`. For example, `[10]`.

- A sequences of discrete tokens, which is represented as a list of `integer-ids`. For example, `[0,12,10,13]`.

The object in each pair can be asymmetric. For example, the pairs can be (token, sequence) or (token, token) or (sequence, sequence). For token inputs, the algorithm supports simple embeddings as compatible encoders. For sequences of token vectors, the algorithm supports the following as encoders:

- Average-pooled embeddings
- Hierarchical convolutional neural networks (CNNs),
- Multi-layered bidirectional long short-term memory (BiLSTMs)

The input label for each pair can be one of the following:

- A categorical label that expresses the relationship between the objects in the pair
- A score that expresses the strength of the similarity between the two objects

For categorical labels used in classification, the algorithm supports the cross-entropy loss function. For ratings/score-based labels used in regression, the algorithm supports the mean squared error (MSE) loss function. Specify these loss functions with the `output_layer` hyperparameter when you create the model training job.

EC2 Instance Recommendation for the Object2Vec Algorithm

The type of Amazon Elastic Compute Cloud (Amazon EC2) instance that you use depends on whether you are training or running inferences.

Instance Recommendation for Training

When training a model using the Object2Vec algorithm on a CPU, start with an ml.m5.2xlarge instance. For training on a GPU, start with an ml.p2.xlarge instance. If the training takes too long on this instance, you can use a larger instance, such as an ml.m5.4xlarge or an ml.m5.12xlarge instance Currently, the Object2Vec algorithm can train only on a single machine. However, it does offer support for multiple GPUs.

Instance Recommendation for Inference

For inference with a trained Object2Vec model that has a deep neural network, we recommend using ml.p3.2xlarge GPU instance. Due to GPU memory scarcity, the `INFERENCE_PREFERRED_MODE` environment variable can be specified to optimize on whether the the section called "GPU optimization: Classification or Regression" (p. 196) or the section called "GPU optimization: Encoder Embeddings" (p. 198) inference network is loaded into GPU.

Object2Vec Sample Notebooks

For a sample notebook that uses the Amazon SageMaker Object2Vec algorithm to encode sequences into fixed-length embeddings, see Using Object2Vec to Encode Sentences into Fixed Length Embeddings. For a sample notebook that uses the Object2Vec algorithm in a multi-label prediction setting to predict the genre of a movie from its plot description, see Movie genre prediction with Object2Vec Algorithm. For a sample notebook that uses the Object2Vec algorithm to make movie recommendations, see An Introduction to SageMaker ObjectToVec model for MovieLens recommendation. For a sample notebook that uses the Object2Vec algorithm to learn document embeddings, see Using Object2Vec to learn document embeddings. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). After you have created a notebook instance and opened it, choose

SageMaker Examples to see a list of Amazon SageMaker samples. To open a notebook, choose its **Use** tab and choose **Create copy**.

How Object2Vec Works

When using the Amazon SageMaker Object2Vec algorithm, you follow the standard workflow: process the data, train the model, and produce inferences.

Topics

Step 1: Process Data

During preprocessing, convert the data to the JSON Lines text file format specified in Data Formats for Object2Vec Training (p. 196) . To get the highest accuracy during training, also randomly shuffle the data before feeding it into the model. How you generate random permutations depends on the language. For python, you could use `np.randon.shuffle`; for Unix, `shuf`.

Step 2: Train a Model

The Amazon SageMaker Object2Vec algorithm has the following main components:

- **Two input channels** – The input channels take a pair of objects of the same or different types as inputs, and pass them to independent and customizable encoders.
- **Two encoders** – The two encoders, enc0 and enc1, convert each object into a fixed-length embedding vector. The encoded embeddings of the objects in the pair are then passed into a comparator.
- **A comparator** – The comparator compares the embeddings in different ways and outputs scores that indicate the strength of the relationship between the paired objects. In the output score for a sentence pair. For example, 1 indicates a strong relationship between a sentence pair, and 0 represents a weak relationship.

During training, the algorithm accepts pairs of objects and their relationship labels or scores as inputs. The objects in each pair can be of different types, as described earlier. If the inputs to both encoders are composed of the same token-level units, you can use a shared token embedding layer by setting the `tied_token_embedding_weight` hyperparameter to `True` when you create the training job. This is possible, for example, when comparing sentences that both have word token-level units. To generate negative samples at a specified rate, set the `negative_sampling_rate` hyperparameter to the desired ratio of negative to positive samples. This hyperparameter expedites learning how to discriminate between the positive samples observed in the training data and the negative samples that are not likely to be observed.

Pairs of objects are passed through independent, customizable encoders that are compatible with the input types of corresponding objects. The encoders convert each object in a pair into a fixed-length embedding vector of equal length. The pair of vectors are passed to a comparator operator, which assembles the vectors into a single vector using the value specified in the he `comparator_list` hyperparameter. The assembled vector then passes through a multilayer perceptron (MLP) layer, which produces an output that the loss function compares with the labels that you provided. This comparison evaluates the strength of the relationship between the objects in the pair as predicted by the model. The following figure shows this workflow.
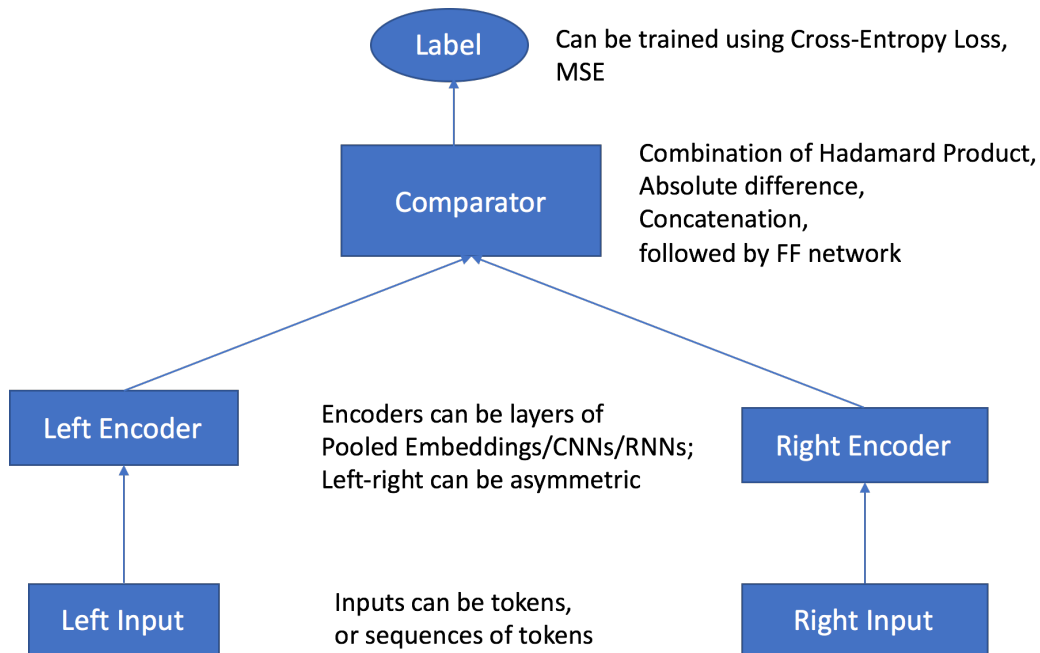
Architecture of the Object2Vec Algorithm from Data Inputs to Scores

Step 3: Produce Inferences

After the model is trained, you can use the trained encoder to preprocess input objects or to perform two types of inference:

- To convert singleton input objects into fixed-length embeddings using the corresponding encoder
- To predict the relationship label or score between a pair of input objects

The inference server automatically figures out which of the types is requested based on the input data. To get the embeddings as output, provide only one input. To predict the relationship label or score, provide both inputs in the pair.

Object2Vec Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Object2Vec training algorithm.

Parameter Name	Description
enc0_max_seq_len	The maximum sequence length for the enc0 encoder. **Required** Valid values: 1 ≤ integer ≤ 5000
enc0_vocab_size	The vocabulary size of enc0 tokens. **Required** Valid values: 2 ≤ integer ≤ 3000000

Parameter Name	Description
`bucket_width`	The allowed difference between data sequence length when bucketing is enabled. To enable bucketing, specify a non-zero value for this parameter. **Optional** Valid values: 0 ≤ integer ≤ 100 Default value: 0 (no bucketing)
`comparator_list`	A list used to customize the way in which two embeddings are compared. The Object2Vec comparator operator layer takes the encodings from both encoders as inputs and outputs a single vector. This vector is a concatenation of subvectors. The string values passed to the `comparator_list` and the order in which they are passed determine how these subvectors are assembled. For example, if `comparator_list="hadamard, concat"`, then the comparator operator constructs the vector by concatenating the Hadamard product of two encodings and the concatenation of two encodings. If, on the other hand, `comparator_list="hadamard"`, then the comparator operator constructs the vector as the hadamard product of only two encodings. **Optional** Valid values: A string that contains any combination of the names of the three binary operators: `hadamard`, `concat`, or `abs_diff`. The Object2Vec algorithm currently requires that the two vector encodings have the same dimension. These operators produce the subvectors as follows: • `hadamard`: Constructs a vector as the Hadamard (element-wise) product of two encodings. • `concat`: Constructs a vector as the concatenation of two encodings. • `abs_diff`: Constructs a vector as the absolute difference between two encodings. Default value: `"hadamard, concat, abs_diff"`
`dropout`	The dropout probability for network layers. *Dropout* is a form of regularization used in neural networks that reduces overfitting by trimming codependent neurons. **Optional** Valid values: 0.0 ≤ float ≤ 1.0 Default value: 0.0

Parameter Name	Description
early_stopping_patience	The number of consecutive epochs without improvement allowed before early stopping is applied. Improvement is defined by with the early_stopping_tolerance hyperparameter. **Optional** Valid values: 1 ≤ integer ≤ 5 Default value: 3
early_stopping_tolerance	The reduction in the loss function that an algorithm must achieve between consecutive epochs to avoid early stopping after the number of consecutive epochs specified in the early_stopping_patience hyperparameter concludes. **Optional** Valid values: 0.000001 ≤ float ≤ 0.1 Default value: 0.01
enc_dim	The dimension of the output of the embedding layer. **Optional** Valid values: 4 ≤ integer ≤ 10000 Default value: 4096
enc0_network	The network model for the enc0 encoder. **Optional** Valid values: hcnn, bilstm, or pooled_embedding • hcnn: A hierarchical convolutional neural network. • bilstm: A bidirectional long short-term memory network (biLSTM), in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks. • pooled_embedding: Averages the embeddings of all of the tokens in the input. Default value: hcnn
enc0_cnn_filter_width	The filter width of the convolutional neural network (CNN) enc0 encoder. **Conditional** Valid values: 1 ≤ integer ≤ 9 Default value: 3

Parameter Name	Description
enc0_freeze_pretrained_embedding	Whether to freeze enc0 pretrained embedding weights. **Conditional** Valid values: `True` or `False` Default value: `True`
enc0_layers	The number of layers in the enc0 encoder. **Conditional** Valid values: `auto` or 1 ≤ integer ≤ 4 • For `hcnn`, `auto` means 4. • For `bilstm`, `auto` means 1. • For `pooled_embedding`, `auto` ignores the number of layers. Default value: `auto`
enc0_pretrained_embedding_file	The filename of the pretrained enc0 token embedding file in the auxiliary data channel. **Conditional** Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._] Default value: "" (empty string)
enc0_token_embedding_dim	The output dimension of the enc0 token embedding layer. **Conditional** Valid values: 2 ≤ integer ≤ 1000 Default value: 300
enc0_vocab_file	The vocabulary file for mapping pretrained enc0 token embedding vectors to numerical vocabulary IDs. **Conditional** Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._] Default value: "" (empty string)

Parameter Name	Description
enc1_network	The network model for the enc1 encoder. If you want the enc1 encoder to use the same network model as enc0, including the hyperparameter values, set the value to `enc0`. **Note** Even when the enc0 and enc1 encoder networks have symmetric architectures, you can't shared parameter values for these networks. **Optional** Valid values: `enc0`, `hcnn`, `bilstm`, or `pooled_embedding` • `enc0`: The network model for the enc0 encoder. • `hcnn`: A hierarchical convolutional neural network. • `bilstm`: A bidirectional LSTM, in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks. • `pooled_embedding`: The averages of the embeddings of all of the tokens in the input. Default value: `enc0`
enc1_cnn_filter_width	The filter width of the CNN enc1 encoder. **Conditional** Valid values: 1 ≤ integer ≤ 9 Default value: 3
enc1_freeze_pretrained_embedding	Whether to freeze enc1 pretrained embedding weights. **Conditional** Valid values: `True` or `False` Default value: `True`
enc1_layers	The number of layers in the enc1 encoder. **Conditional** Valid values: `auto` or 1 ≤ integer ≤ 4 • For `hcnn`, `auto` means 4. • For `bilstm`, `auto` means 1. • For `pooled_embedding`, `auto` ignores the number of layers. Default value: `auto`
enc1_max_seq_len	The maximum sequence length for the enc1 encoder. **Conditional** Valid values: 1 ≤ integer ≤ 5000

Parameter Name	Description
`enc1_pretrained_embedding_file`	The name of the enc1 pretrained token embedding file in the auxiliary data channel. **Conditional** Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._] Default value: "" (empty string)
`enc1_token_embedding_dim`	The output dimension of the enc1 token embedding layer. **Conditional** Valid values: 2 ≤ integer ≤ 1000 Default value: 300
`enc1_vocab_file`	The vocabulary file for mapping pretrained enc1 token embeddings to vocabulary IDs. **Conditional** Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._] Default value: "" (empty string)
`enc1_vocab_size`	The vocabulary size of enc0 tokens. **Conditional** Valid values: 2 ≤ integer ≤ 3000000
`epochs`	The number of epochs to run for training. **Optional** Valid values: 1 ≤ integer ≤ 100 Default value: 30
`learning_rate`	The learning rate for training. **Optional** Valid values: 1.0E-6 ≤ float ≤ 1.0 Default value: 0.0004
`mini_batch_size`	The batch size that the dataset is split into for an `optimizer` during training. **Optional** Valid values: 1 ≤ integer ≤ 10000 Default value: 32

Parameter Name	Description
mlp_activation	The type of activation function for the multilayer perceptron (MLP) layer. **Optional** Valid values: `tanh`, `relu`, or `linear` • `tanh`: Hyperbolic tangent • `relu`: Rectified linear unit (ReLU) • `linear`: Linear function Default value: `linear`
mlp_dim	The dimension of the output from MLP layers. **Optional** Valid values: 2 ≤ integer ≤ 10000 Default value: 512
mlp_layers	The number of MLP layers in the network. **Optional** Valid values: 0 ≤ integer ≤ 10 Default value: 2
negative_sampling_rate	The ratio of negative samples, generated to assist in training the algorithm, to positive samples that are provided by users. Negative samples represent data that is unlikely to occur in reality and are labelled negatively for training. They facilitate training a model to discriminate between the positive samples observed and the negative samples that are not. To specify the ratio of negative samples to positive samples used for training, set the value to a positive integer. For example, if you train the algorithm on input data in which all of the samples are positive and set `negative_sampling_rate` to 2, the Object2Vec algorithm internally generates two negative samples per positive sample. If you don't want to generate or use negative samples during training, set the value to 0. **Optional** Valid values: 0 ≤ integer Default value: 0 (off)
num_classes	The number of classes for classification training. Amazon SageMaker ignores this hyperparameter for regression problems. **Optional** Valid values: 2 ≤ integer ≤ 30 Default value: 2

Parameter Name	Description
optimizer	The optimizer type. **Optional** Valid values: `adadelta`, `adagrad`, `adam`, `sgd`, or `rmsprop`. `adadelta`: A per-dimension learning rate method for gradient descent `adagrad`: The adaptive gradient algorithm `adam`: The adaptive moment estimation algorithm `sgd`: Stochastic gradient descent `rmsprop`: Root mean square propagation Default value: `adam`
output_layer	The type of output layer where you specify that the task is regression or classification. **Optional** Valid values: `softmax` or `mean_squared_error` `softmax`: The Softmax function used for classification. `mean_squared_error`: The MSE used for regression. Default value: `softmax`
tied_token_embedding_weight	Whether to use a shared embedding layer for both encoders. If the inputs to both encoders use the same token-level units, use a shared token embedding layer. For example, for a collection of documents, if one encoder encodes sentences and another encodes whole documents, you can use a shared token embedding layer. That's because both sentences and documents are composed of word tokens from the same vocabulary. **Optional** Valid values: `True` or `False` Default value: `False`

Parameter Name	Description
token_embedding_storage_type	The mode of gradient update used during training: when the `dense` mode is used, the optimizer calculates the full gradient matrix for the token embedding layer even if most rows of the gradient are zero-valued. When `sparse` mode is used, the optimizer only stores rows of the gradient that are actually being used in the mini-batch. If you want the algorithm to perform lazy gradient updates, which calculate the gradients only in the non-zero rows and which speed up training, specify `row_sparse`. Setting the value to `row_sparse` constrains the values available for other hyperparameters, as follows: • The `optimizer` hyperparameter must be set to `adam`, `adagrad`, or `sgd`. Otherwise, the algorithm throws a `CustomerValueError`. • The algorithm automatically disables bucketing, setting the `bucket_width` hyperparameter to 0. **Optional** Valid values: `dense` or `row_sparse` Default value: `dense`
weight_decay	The weight decay parameter used for optimization. **Optional** Valid values: 0 ≤ float ≤ 10000 Default value: 0 (no decay)

Tune an Object2Vec Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. For the objective metric, you use one of the metrics that the algorithm computes. Automatic model tuning searches the chosen hyperparameters to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm has both classification and regression metrics. The `output_layer` type determines which metric you can use for automatic model tuning.

Regressor Metrics Computed by the Object2Vec Algorithm

The algorithm reports a mean squared error regressor metric, which is computed during testing and validation. When tuning the model for regression tasks, choose this metric as the objective.

Metric Name	Description	Optimization Direction
`test:mean_squared_error`	Root Mean Square Error	Minimize
`validation:mean_squared_error`	Root Mean Square Error	Minimize

Classification Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm reports accuracy and cross-entropy classification metrics, which are computed during test and validation. When tuning the model for classification tasks, choose one of these as the objective.

Metric Name	Description	Optimization Direction
`test:accuracy`	Accuracy	Maximize
`test:cross_entropy`	Cross-entropy	Minimize
`validation:accuracy`	Accuracy	Maximize
`validation:cross_entropy`	Cross-entropy	Minimize

Tunable Object2Vec Hyperparameters

You can tune the following hyperparameters for the Object2Vec algorithm.

Hyperparameter Name	Hyperparameter Type	Recommended Ranges and Values	
`dropout`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0	
`early_stopping_patience`	IntegerParameterRange	MinValue: 1, MaxValue: 5	
`early_stopping_tolerance`	ContinuousParameterRange	MinValue: 0.001, MaxValue: 0.1	
`enc_dim`	IntegerParameterRange	MinValue: 4, MaxValue: 4096	
`enc0_cnn_filter_width`	IntegerParameterRange	MinValue: 1, MaxValue: 5	
`enc0_layers`	IntegerParameterRange	MinValue: 1, MaxValue: 4	
`enc0_token_embedding_dim`	IntegerParameterRange	MinValue: 5, MaxValue: 300	
`enc1_cnn_filter_width`	IntegerParameterRange	MinValue: 1, MaxValue: 5	
`enc1_layers`	IntegerParameterRange	MinValue: 1, MaxValue: 4	

Hyperparameter Name	Hyperparameter Type	Recommended Ranges and Values	
`enc1_token_embedding_dim`	IntegerParameterRange	MinValue: 5, MaxValue: 300	
`epochs`	IntegerParameterRange	MinValue: 4, MaxValue: 20	
`learning_rate`	ContinuousParameterRange	MinValue: 1e-6, MaxValue: 1.0	
`mini_batch_size`	IntegerParameterRange	MinValue: 1, MaxValue: 8192	
`mlp_activation`	CategoricalParameterRanges	[`tanh, relu, linear`]	
`mlp_dim`	IntegerParameterRange	MinValue: 16, MaxValue: 1024	
`mlp_layers`	IntegerParameterRange	MinValue: 1, MaxValue: 4	
`optimizer`	CategoricalParameterRanges	[`adagrad, adam, rmsprop, sgd, adadelta`]	
`weight_decay`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0	

Data Formats for Object2Vec Training

Input: JSON Lines Request Format

Content-type: application/jsonlines

```
{"label": 0, "in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15,
 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}
{"label": 1, "in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9,
 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}
{"label": 1, "in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

The "in0" and "in1" are the inputs for encoder0 and encoder1, respectively. The same format is valid for both classification and regression problems. For regression, the field `"label"` can accept real valued inputs.

Data Formats for Object2Vec Inference

GPU optimization: Classification or Regression

Due to GPU memory scarcity, the `INFERENCE_PREFERRED_MODE` environment variable can be specified to optimize on whether the classification/regression or the the section called "Output: Encoder Embeddings" (p. 198) inference network is loaded into GPU. If the majority of your inference is for classification or regression, specify `INFERENCE_PREFERRED_MODE=classification`. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for classification/regression inference:

```
transformer = o2v.transformer(instance_count=4,
                              instance_type="ml.p2.xlarge",
                              max_concurrent_transforms=2,
                              max_payload=1,  # 1MB
                              strategy='MultiRecord',
                              env={'INFERENCE_PREFERRED_MODE': 'classification'},  # only
 useful with GPU
                              output_path=output_s3_path)
```

Input: Classification or Regression Request Format

Content-type: application/json

```
{
  "instances" : [
    {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821,
 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]},
    {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4],
 "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]},
    {"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
  ]
}
```

Content-type: application/jsonlines

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4],
 "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1":
 [22, 32, 13, 25, 1016, 573, 3252, 4]}
{"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

For classification problems, the length of the scores vector corresponds to `num_classes`. For regression problems, the length is 1.

Ouput: Classification or Regression Response Format

Accept: application/json

```
{
    "predictions": [
        {
            "scores": [
                0.6533935070037842,
                0.07582679390907288,
                0.2707797586917877
            ]
        },
        {
            "scores": [
                0.026291321963071823,
                0.6577019095420837,
                0.31600672006607056
            ]
        }
    ]
}
```

Accept: application/jsonlines

```
{"scores":[0.195667684078216,0.395351558923721,0.408980727195739]}
```

```
{"scores":[0.251988261938095,0.258233487606048,0.489778339862823]}
{"scores":[0.280087798833847,0.368331134319305,0.351581096649169]}
```

In both the classification and regression formats, the scores apply to individual labels.

Encoder Embeddings for Object2Vec

GPU optimization: Encoder Embeddings

Due to GPU memory scarcity, the `INFERENCE_PREFERRED_MODE` environment variable can be specified to optimize on whether the or the encoder embedding inference network is loaded into GPU. If the majority of your inference is for encoder embeddings, specify `INFERENCE_PREFERRED_MODE=embedding`. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for encoder embedding inference:

```
transformer = o2v.transformer(instance_count=4,
                              instance_type="ml.p2.xlarge",
                              max_concurrent_transforms=2,
                              max_payload=1,   # 1MB
                              strategy='MultiRecord',
                              env={'INFERENCE_PREFERRED_MODE': 'embedding'},   # only useful
 with GPU
                              output_path=output_s3_path)
```

Input: Encoder Embeddings

Content-type: application/json

```
{
  "instances" : [
    {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821,
 4]},
    {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]},
    {"in0": [774, 14, 21, 206]}
  ]
}
```

Content-type: application/jsonlines

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]}
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]}
{"in0": [774, 14, 21, 206]}
```

In both of these formats, you specify only one input type: "in0" or "in1." The inference service then invokes the corresponding encoder and outputs the embeddings for each of the instances.

Output: Encoder Embeddings

Content-type: application/json

```
{
  "predictions": [
    {"embeddings":
[0.057368703186511,0.030703511089086,0.099890425801277,0.063688032329082,0.026327300816774,0.0036375711
    {"embeddings":
[0.150190666317939,0.05145975202322,0.098204270005226,0.064249359071254,0.056249320507049,0.01513972133
  ]
```

```
}
```

Content-type: application/jsonlines

```
{"embeddings":
[0.057368703186511,0.030703511089086,0.099890425801277,0.063688032329082,0.026327300816774,0.0036375711
{"embeddings":
[0.150190666317939,0.05145975202322,0.098204270005226,0.064249359071254,0.056249320507049,0.01513972133
```

The vector length of the embeddings output by the inference service is equal to the value of one of the following hyperparameters that you specify at training time: `enc0_token_embedding_dim`, `enc1_token_embedding_dim`, or `enc_dim`.

Object Detection Algorithm

The Amazon SageMaker Object Detection algorithm detects and classifies objects in images using a single deep neural network. It is a supervised learning algorithm that takes images as input and identifies all instances of objects within the image scene. The object is categorized into one of the classes in a specified collection with a confidence score that it belongs to the class. Its location and scale in the image are indicated by a rectangular bounding box. It uses the Single Shot multibox Detector (SSD) framework and supports two base networks: VGG and ResNet. The network can be trained from scratch, or trained with models that have been pre-trained on the ImageNet dataset.

Topics

Input/Output Interface for the Object Detection Algorithm

The Amazon SageMaker Object Detection algorithm supports both RecordIO (`application/x-recordio`) and image (`image/png`, `image/jpeg`, and `application/x-image`) content types for training in file mode and supports RecordIO (`application/x-recordio`) for training in pipe mode. However you can also train in pipe mode using the image files (`image/png`, `image/jpeg`, and `application/x-image`), without creating RecordIO files, by using the augmented manifest format. The recommended input format for the Amazon SageMaker object detection algorithms is Apache MXNet RecordIO. However, you can also use raw images in .jpg or .png format. The algorithm supports only `application/x-image` for inference.

> **Note**
> To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

See the Object Detection Sample Notebooks (p. 214) for more details on data formats.

Train with the RecordIO Format

If you use the RecordIO format for training, specify both train and validation channels as values for the `InputDataConfig` parameter of the CreateTrainingJob (p. 667) request. Specify one RecordIO (.rec) file in the train channel and one RecordIO file in the validation channel. Set the content type for

both channels to `application/x-recordio`. An example of how to generate RecordIO file can be found in the object detection sample notebook. You can also use tools from the MXNet example to generate RecordIO files for popular datasets like the PASCAL Visual Object Classes and Common Objects in Context (COCO).

Train with the Image Format

If you use the image format for training, specify `train`, `validation`, `train_annotation`, and `validation_annotation` channels as values for the `InputDataConfig` parameter of CreateTrainingJob (p. 667) request. Specify the individual image data (.jpg or .png) files for the train and validation channels. For annotation data, you can use the JSON format. Specify the corresponding .json files in the `train_annotation` and `validation_annotation` channels. Set the content type for all four channels to `image/png` or `image/jpeg` based on the image type. You can also use the content type `application/x-image` when your dataset contains both .jpg and .png images. The following is an example of a .json file.

```
{
   "file": "your_image_directory/sample_image1.jpg",
   "image_size": [
      {
         "width": 500,
         "height": 400,
         "depth": 3
      }
   ],
   "annotations": [
      {
         "class_id": 0,
         "left": 111,
         "top": 134,
         "width": 61,
         "height": 128
      },
      {
         "class_id": 0,
         "left": 161,
         "top": 250,
         "width": 79,
         "height": 143
      },
      {
         "class_id": 1,
         "left": 101,
         "top": 185,
         "width": 42,
         "height": 130
      }
   ],
   "categories": [
      {
         "class_id": 0,
         "name": "dog"
      },
      {
         "class_id": 1,
         "name": "cat"
      }
   ]
}
```

Each image needs a .json file for annotation, and the .json file should have the same name as the corresponding image. The name of above .json file should be "sample_image1.json". There are four

properties in the annotation .json file. The property "file" specifies the relative path of the image file. For example, if your training images and corresponding .json files are stored in s3://*your_bucket*/ train/sample_image and s3://*your_bucket*/train_annotation, specify the path for your train and train_annotation channels as s3://*your_bucket*/train and s3://*your_bucket*/train_annotation, respectively.

In the .json file, the relative path for an image named sample_image1.jpg should be sample_image/ sample_image1.jpg. The `image_size` property specifies the overall image dimensions. The SageMaker object detection algorithm currently only supports 3-channel images. The `annotations` property specifies the categories and bounding boxes for objects within the image. Each object is annotated by a `class_id` index and by four bounding box coordinates (`left`, `top`, `width`, `height`). The `left` (x-coordinate) and `top` (y-coordinate) values represent the upper-left corner of the bounding box. The `width` (x-coordinate) and `height` (y-coordinate) values represent the dimensions of the bounding box. The origin (0, 0) is the upper-left corner of the entire image. If you have multiple objects within one image, all the annotations should be included in a single .json file. The `categories` property stores the mapping between the class index and class name. The class indices should be numbered successively and the numbering should start with 0. The `categories` property is optional for the annotation .json file

Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the `InputDataConfig` parameter of the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- `AlgorithmSpecification` - Identifies the training algorithm to use.
- `HyperParameters` - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- `InputDataConfig` - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- `OutputDataConfig` - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.
- `ResourceConfig` - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- `EnableManagedSpotTraining` - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.
- `RoleARN` - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- `StoppingCondition` - To help cap training costs, use `MaxRuntimeInSeconds` to set a time limit for training. Use `MaxWaitTimeInSeconds` to specify how long you are willing to to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see How It Works.

Request Syntax

```
{
   "AlgorithmSpecification": {
      "AlgorithmName": "string",
      "MetricDefinitions": [
         {
            "Name": "string",
            "Regex": "string"
         }
      ],
      "TrainingImage": "string",
      "TrainingInputMode": "string"
   },
   "CheckpointConfig": {
      "LocalPath": "string",
      "S3Uri": "string"
   },
   "EnableInterContainerTrafficEncryption": boolean,
   "EnableManagedSpotTraining": boolean,
   "EnableNetworkIsolation": boolean,
   "HyperParameters": {
      "string" : "string"
   },
   "InputDataConfig": [
      {
         "ChannelName": "string",
         "CompressionType": "string",
         "ContentType": "string",
         "DataSource": {
            "FileSystemDataSource": {
               "DirectoryPath": "string",
```

```
        "MaxWaitTimeInSeconds": number

    },

    "Tags": [

        {

            "Key": "string",

            "Value": "string"

        }

    ],

    "TrainingJobName": "string",

    "VpcConfig": {

        "SecurityGroupIds": [ "string" ],

        "Subnets": [ "string" ]

    }

}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 667)

> The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

> Type: AlgorithmSpecification (p. 863) object

> Required: Yes

CheckpointConfig (p. 667)

> Contains information about the output location for managed spot training checkpoint data.

> Type: CheckpointConfig (p. 880) object

> Required: No

EnableInterContainerTrafficEncryption (p. 667)

> To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.

> Type: Boolean

provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No

EnableNetworkIsolation (p. 667)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

> **Note**
> The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 667)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the `Length Constraint`.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

InputDataConfig (p. 667)

An array of `Channel` objects. Each channel is a named input source. `InputDataConfig` describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, `training_data` and `validation_data`. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

OutputDataConfig (p. 667)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 976) object

Required: Yes

ResourceConfig (p. 667)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose `File` as the `TrainingInputMode` in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 991) object

Required: Yes

RoleArn (p. 667)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

> **Note**
> To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StoppingCondition (p. 667)

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1004) object

Required: Yes

Tags (p. 667)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobName (p. 667)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

VpcConfig (p. 667)

A VpcConfig (p. 1039) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

Response Syntax

```
{
    "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 672)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(p.   ) request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in JSON Lines format in which each line represents one sample. The images are specified using the `'source-ref'` tag that points to the S3 location of the image. The annotations are provided under the `"AttributeNames"` parameter value as specified in the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- `AlgorithmSpecification` - Identifies the training algorithm to use.
- `HyperParameters` - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- `InputDataConfig` - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- `OutputDataConfig` - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.
- `ResourceConfig` - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.

- `EnableManagedSpotTraining` - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.

- `RoleARN` - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- `StoppingCondition` - To help cap training costs, use `MaxRuntimeInSeconds` to set a time limit for training. Use `MaxWaitTimeInSeconds` to specify how long you are willing to to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see How It Works.

Request Syntax

```
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
    },
    "EnableInterContainerTrafficEncryption": boolean,
    "EnableManagedSpotTraining": boolean,
    "EnableNetworkIsolation": boolean,
    "HyperParameters": {
        "string" : "string"
    },
    "InputDataConfig": [
        {
            "ChannelName": "string"
```

```
    "OutputDataConfig": {

        "KmsKeyId": "string",

        "S3OutputPath": "string"

    },

    "ResourceConfig": {

        "InstanceCount": number,

        "InstanceType": "string",

        "VolumeKmsKeyId": "string",

        "VolumeSizeInGB": number

    },

    "RoleArn": "string",

    "StoppingCondition": {

        "MaxRuntimeInSeconds": number,

        "MaxWaitTimeInSeconds": number

    },

    "Tags": [

        {

            "Key": "string",

            "Value": "string"

        }

    ],

    "TrainingJobName": "string",

    "VpcConfig": {

        "SecurityGroupIds": [ "string" ],

        "Subnets": [ "string" ]

    }

}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 667)

The registry path of the Docker image that contains the training algorithm and

Type: Boolean

Required: No

EnableManagedSpotTraining (p. 667)

To train models using managed spot training, choose `True`. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. this option is useful when training jobs can be interrupted and when there is flexibility when the training job is run.

The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No

EnableNetworkIsolation (p. 667)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

> **Note**
> The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 667)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the `Length Constraint`.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

InputDataConfig (p. 667)

An array of `Channel` objects. Each channel is a named input source. `InputDataConfig` describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, `training_data` and `validation_data`. The

configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

OutputDataConfig (p. 667)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 976) object

Required: Yes

ResourceConfig (p. 667)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose `File` as the `TrainingInputMode` in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 991) object

Required: Yes

RoleArn (p. 667)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

> **Note**
> To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StoppingCondition (p. 667)

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays

job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1004) object

Required: Yes

Tags (p. 667)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide.*

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobName (p. 667)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

VpcConfig (p. 667)

A VpcConfig (p. 1039) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

Response Syntax

```
{
    "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 672)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Errors

For information about the errors that are common to all actions, see Common
Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have
too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the
following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(p.   ) request. It can also contain additional metadata under the `metadata` tag, but these are
ignored by the algorithm. In the following example, the `"AttributeNames` are contained in the list
`["source-ref", "bounding-box"]`:

```
{"source-ref": "s3://your_bucket/image1.jpg", "bounding-box":{"image_size":[{ "width":
 500, "height": 400, "depth":3}], "annotations":[{"class_id": 0, "left": 111, "top":
 134, "width": 61, "height": 128}, {"class_id": 5, "left": 161, "top": 250, "width": 80,
 "height": 50}]}, "bounding-box-metadata":{"class-map":{"0": "dog", "5": "horse"}, "type":
 "groundtruth/object_detection"}}
{"source-ref": "s3://your_bucket/image2.jpg", "bounding-box":{"image_size":[{ "width":
 400, "height": 300, "depth":3}], "annotations":[{"class_id": 1, "left": 100, "top": 120,
 "width": 43, "height": 78}]}, "bounding-box-metadata":{"class-map":{"1": "cat"}, "type":
 "groundtruth/object_detection"}}
```

The order of `"AttributeNames"` in the input files matters when training the Object Detection
algorithm. It accepts piped data in a specific order, with `image` first, followed by `annotations`. So the
"AttributeNames" in this example are provided with `"source-ref"` first, followed by `"bounding-box"`.
When using Object Detection with Augmented Manifest, the value of parameter `RecordWrapperType`
must be set as `"RecordIO"`.

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with
an Augmented Manifest File (p. 308).

Incremental Training

You can also seed the training of a new model with the artifacts from a model that you trained previously with Amazon SageMaker. Incremental training saves training time when you want to train a new model with the same or similar data. Amazon SageMaker object detection models can be seeded only with another built-in object detection model trained in Amazon SageMaker.

To use a pretrained model, in the CreateTrainingJob (p. 667) request, specify the `ChannelName` as "model" in the `InputDataConfig` parameter. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the `base_network` and `num_classes` input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by Amazon SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the Amazon SageMaker object detection algorithm, see Amazon SageMaker Object Detection Incremental Training sample notebook. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 282).

EC2 Instance Recommendation for the Object Detection Algorithm

For object detection, we support the following GPU instances for training: `ml.p2.xlarge`, `ml.p2.8xlarge`, `ml.p2.16xlarge`, `ml.p3.2xlarge`, `ml.p3.8xlarge` and `ml.p3.16xlarge`. We recommend using GPU instances with more memory for training with large batch sizes. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training. However, both CPU (such as C5 and M5) and GPU (such as P2 and P3) instances can be used for the inference. All the supported instance types for inference are itemized on Amazon SageMaker ML Instance Types.

Object Detection Sample Notebooks

For a sample notebook that shows how to use the Amazon SageMaker Object Detection algorithm to train and host a model on the COCO dataset using the Single Shot multibox Detector algorithm, see Object Detection using the Image and JSON format. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Object Detection Works

The object detection algorithm identifies and locates all instances of objects in an image from a known collection of object categories. The algorithm takes an image as input and outputs the category that the object belongs to, along with a confidence score that it belongs to the category. The algorithm also predicts the object's location and scale with a rectangular bounding box. Amazon SageMaker Object Detection uses the Single Shot multibox Detector (SSD) algorithm that takes a convolutional neural network (CNN) pretrained for classification task as the base network. SSD uses the output of intermediate layers as features for detection.

Various CNNs such as VGG and ResNet have achieved great performance on the image classification task. Object detection in Amazon SageMaker supports both VGG-16 and ResNet-50 as a base network for SSD. The algorithm can be trained in full training mode or in transfer learning mode. In full training mode, the base network is initialized with random weights and then trained on user data. In transfer learning mode, the base network weights are loaded from pretrained models.

The object detection algorithm uses standard data augmentation operations, such as flip, rescale, and jitter, on the fly internally to help avoid overfitting.

Object Detection Hyperparameters

In the CreateTrainingJob (p. 667) request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters that are used to help estimate the parameters of the model from a training dataset. The following table lists the hyperparameters provided by Amazon SageMaker for training the object detection algorithm. For more information about how object training works, see How Object Detection Works (p. 214).

Parameter Name	Description
num_classes	The number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset. **Required** Valid values: positive integer
num_training_samples	The number of training examples in the input dataset. **Note** If there is a mismatch between this value and the number of samples in the training set, then the behavior of the `lr_scheduler_step` parameter will be undefined and distributed training accuracy may be affected. **Required** Valid values: positive integer
base_network	The base network architecture to use. **Optional** Valid values: 'vgg-16' or 'resnet-50' Default value: 'vgg-16'
early_stopping	`True` to use early stopping logic during training. `False` not to use it. **Optional** Valid values: `True` or `False` Default value: `False`
early_stopping_min_epochs	The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when `early_stopping = True`. **Optional** Valid values: positive integer Default value: 10

Parameter Name	Description
early_stopping_patience	The number of epochs to wait before ending training if no improvement, as defined by the `early_stopping_tolerance` hyperparameter, is made in the relevant metric. It is used only when `early_stopping` = `True`. **Optional** Valid values: positive integer Default value: 5
early_stopping_tolerance	The tolerance value that the relative improvement in `validation:mAP`, the mean average precision (mAP), is required to exceed to avoid early stopping. If the ratio of the change in the mAP divided by the previous best mAP is smaller than the `early_stopping_tolerance` value set, early stopping considers that there is no improvement. It is used only when `early_stopping` = `True`. **Optional** Valid values: 0 ≤ float ≤ 1 Default value: 0.0
image_shape	The image size for input images. We rescale the input image to a square image with this size. We recommend using 300 and 512 for better performance. **Optional** Valid values: positive integer ≥300 Default: 300
epochs	The number of training epochs. **Optional** Valid values: positive integer Default: 30

Parameter Name	Description
`freeze_layer_pattern`	The regular expression (regex) for freezing layers in the base network. For example, if we set `freeze_layer_pattern` = `"^(conv1_|conv2_).*"`, then any layers with a name that contains `"conv1_"` or `"conv2_"` are frozen, which means that the weights for these layers are not updated during training. The layer names can be found in the network symbol files vgg16-symbol.json and resnet-50-symbol.json. Freezing a layer means that its weights can not be modified further. This can reduce training time significantly in exchange for modest losses in accuracy. This technique is commonly used in transfer learning where the lower layers in the base network do not need to be retrained. **Optional** Valid values: string Default: No layers frozen.
`kv_store`	The weight update synchronization mode used for distributed training. The weights can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See the Distributed Training MXNet tutorial for details. **Note** This parameter is not applicable to single machine training. **Optional** Valid values: `'dist_sync'` or `'dist_async'` • `'dist_sync'`: The gradients are synchronized after every batch with all the workers. With `'dist_sync'`, batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then dist_sync behaves like a single machine with batch size n*b. • `'dist_async'`: Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed. Default: -

Parameter Name	Description
label_width	The force padding label width used to sync across training and validation data. For example, if one image in the data contains at most 10 objects, and each object's annotation is specified with 5 numbers, [class_id, left, top, width, height], then the label_width should be no smaller than (10*5 + header information length). The header information length is usually 2. We recommend using a slightly larger label_width for the training, such as 60 for this example. **Optional** Valid values: Positive integer large enough to accommodate the largest annotation information length in the data. Default: 350
learning_rate	The initial learning rate. **Optional** Valid values: float in (0, 1] Default: 0.001
lr_scheduler_factor	The ratio to reduce learning rate. Used in conjunction with the lr_scheduler_step parameter defined as lr_new = lr_old * lr_scheduler_factor. **Optional** Valid values: float in (0, 1) Default: 0.1
lr_scheduler_step	The epochs at which to reduce the learning rate. The learning rate is reduced by lr_scheduler_factor at epochs listed in a comma-delimited string: "epoch1, epoch2, ...". For example, if the value is set to "10, 20" and the lr_scheduler_factor is set to 1/2, then the learning rate is halved after 10th epoch and then halved again after 20th epoch. **Optional** Valid values: string Default: empty string

Parameter Name	Description
mini_batch_size	The batch size for training. In a single-machine multi-gpu setting, each GPU handles `mini_batch_size/num_gpu` training samples. For the multi-machine training in `dist_sync` mode, the actual batch size is `mini_batch_size*number` of machines. A large `mini_batch_size` usually leads to faster training, but it may cause out of memory problem. The memory usage is related to `mini_batch_size`, `image_shape`, and `base_network` architecture. For example, on a single p3.2xlarge instance, the largest `mini_batch_size` without an out of memory error is 32 with the base_network set to "resnet-50" and an `image_shape` of 300. With the same instance, you can use 64 as the `mini_batch_size` with the base network `vgg-16` and an `image_shape` of 300. **Optional** Valid values: positive integer Default: 32
momentum	The momentum for `sgd`. Ignored for other optimizers. **Optional** Valid values: float in (0, 1] Default: 0.9
nms_threshold	The non-maximum suppression threshold. **Optional** Valid values: float in (0, 1] Default: 0.45
optimizer	The optimizer types. For details on optimizer values, see MXNet's API. **Optional** Valid values: ['sgd', 'adam', 'rmsprop', 'adadelta'] Default: 'sgd'
overlap_threshold	The evaluation overlap threshold. **Optional** Valid values: float in (0, 1] Default: 0.5

Parameter Name	Description
use_pretrained_model	Indicates whether to use a pre-trained model for training. If set to 1, then the pre-trained model with corresponding architecture is loaded and used for training. Otherwise, the network is trained from scratch. **Optional** Valid values: 0 or 1 Default: 1
weight_decay	The weight decay coefficient for `sgd` and `rmsprop`. Ignored for other optimizers. **Optional** Valid values: float in (0, 1) Default: 0.0005

Tune an Object Detection Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see .

Metrics Computed by the Object Detection Algorithm

The object detection algorithm reports on a single metric during training: `validation:mAP`. When tuning a model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
validation:mAP	Mean Average Precision (mAP) computed on the validation set.	Maximize

Tunable Object Detection Hyperparameters

Tune the Amazon SageMaker object detection model with the following hyperparameters. The hyperparameters that have the greatest impact on the object detection objective metric are: `mini_batch_size`, `learning_rate`, and `optimizer`.

Parameter Name	Parameter Type	Recommended Ranges
learning_rate	ContinuousParameterRange	MinValue: 1e-6, MaxValue: 0.5
mini_batch_size	IntegerParameterRanges	MinValue: 8, MaxValue: 64

Parameter Name	Parameter Type	Recommended Ranges
`momentum`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.999
`optimizer`	CategoricalParameterRanges	['sgd', 'adam', 'rmsprop', 'adadelta']
`weight_decay`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.999

Object Detection Request and Response Formats

Request Format

Query a trained model by using the model's endpoint. The endpoint takes .jpg and .png image formats with `image/jpeg` and `image/png` content-types.

Response Formats

The response is the class index with a confidence score and bounding box coordinates for all objects within the image encoded in JSON format. The following is an example of response .json file:

```
{"prediction":[
  [4.0, 0.86419455409049988, 0.3088374733924866, 0.07030484080314636, 0.7110607028007507,
 0.9345266819000244],
  [0.0, 0.73376623392105103, 0.5714187026023865, 0.40427327156066895, 0.827075183391571,
 0.9712159633636475],
  [4.0, 0.32643985450267792, 0.3677481412887573, 0.034883320331573486, 0.6318609714508057,
 0.5967587828636169],
  [8.0, 0.22552496790885925, 0.6152569651603699, 0.5722782611846924, 0.882301390171051,
 0.8985623121261597],
  [3.0, 0.42260299175977707, 0.019305512309074402, 0.08386176824569702,
 0.39093565940856934, 0.9574796557426453]
]}
```

Each row in this .json file contains an array that represents a detected object. Each of these object arrays consists of a list of six numbers. The first number is the predicted class label. The second number is the associated confidence score for the detection. The last four numbers represent the bounding box coordinates [xmin, ymin, xmax, ymax]. These output bounding box corner indices are normalized by the overall image size. Note that this encoding is different than that use by the input .json format. For example, in the first entry of the detection result, 0.3088374733924866 is the left coordinate (x-coordinate of upper-left corner) of the bounding box as a ratio of the overall image width, 0.07030484080314636 is the top coordinate (y-coordinate of upper-left corner) of the bounding box as a ratio of the overall image height, 0.7110607028007507 is the right coordinate (x-coordinate of lower-right corner) of the bounding box as a ratio of the overall image width, and 0.9345266819000244 is the bottom coordinate (y-coordinate of lower-right corner) of the bounding box as a ratio of the overall image height.

To avoid unreliable detection results, you might want to filter out the detection results with low confidence scores. In the object detection sample notebook, we provide scripts to remove the low confidence detections. Scripts are also provided to plot the bounding boxes on the original image.

For batch transform, the response is in JSON format, where the format is identical to the JSON format described above. The detection results of each image is represented as a JSON file. For example:

```
{"prediction": [[label_id, confidence_score, xmin, ymin, xmax, ymax], [label_id,
 confidence_score, xmin, ymin, xmax, ymax]]}
```

For more details on training and inference, see the .

OUTPUT: JSON Response Format

accept: application/json;annotation=1

```
{
    "image_size": [
        {
            "width": 500,
            "height": 400,
            "depth": 3
        }
    ],
    "annotations": [
        {
            "class_id": 0,
            "score": 0.943,
            "left": 111,
            "top": 134,
            "width": 61,
            "height": 128
        },
        {
            "class_id": 0,
            "score": 0.0013,
            "left": 161,
            "top": 250,
            "width": 79,
            "height": 143
        },
        {
            "class_id": 1,
            "score": 0.0133,
            "left": 101,
            "top": 185,
            "width": 42,
            "height": 130
        }
    ]
}
```

Principal Component Analysis (PCA) Algorithm

PCA is an unsupervised machine learning algorithm that attempts to reduce the dimensionality (number of features) within a dataset while still retaining as much information as possible. This is done by finding a new set of features called *components*, which are composites of the original features that are uncorrelated with one another. They are also constrained so that the first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

In Amazon SageMaker, PCA operates in two modes, depending on the scenario:

- **regular**: For datasets with sparse data and a moderate number of observations and features.
- **randomized**: For datasets with both a large number of observations and features. This mode uses an approximation algorithm.

PCA uses tabular data.

The rows represent observations you want to embed in a lower dimensional space. The columns represent features that you want to find a reduced approximation for. The algorithm calculates the

covariance matrix (or an approximation thereof in a distributed manner), and then performs the singular value decomposition on this summary to produce the principal components.

Topics

Input/Output Interface for the PCA Algorithm

For training, PCA expects data provided in the train channel, and optionally supports a dataset passed to the test dataset, which is scored by the final algorithm. Both `recordIO-wrapped-protobuf` and `CSV` formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, PCA supports `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Results are returned in either `application/json` or `application/x-recordio-protobuf` format with a vector of "projections."

For more details on training and inference file formats, see the PCA Sample Notebooks (p. 223) and the PCA Response Formats (p. 226).

For more information on input and output file formats, see PCA Response Formats (p. 226) for inference and the PCA Sample Notebooks (p. 223).

EC2 Instance Recommendation for the PCA Algorithm

PCA supports both GPU and CPU computation. Which instance type is most performant depends heavily on the specifics of the input data.

PCA Sample Notebooks

For a sample notebook that shows how to use the Amazon SageMaker Principal Component Analysis algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to PCA with MNIST. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How PCA Works

Principal Component Analysis (PCA) is a learning algorithm that reduces the dimensionality (number of features) within a dataset while still retaining as much information as possible.

PCA reduces dimensionality by finding a new set of features called *components*, which are composites of the original features, but are uncorrelated with one another. The first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

It is an unsupervised dimensionality reduction algorithm. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

Given the input of a matrix with rows $x_1, \ldots, x_n$ each of dimension `1 * d`, the data is partitioned into mini-batches of rows and distributed among the training nodes (workers). Each worker then computes a summary of its data. The summaries of the different workers are then unified into a single solution at the end of the computation.

Modes

The Amazon SageMaker PCA algorithm uses either of two modes to calculate these summaries, depending on the situation:

- **regular**: for datasets with sparse data and a moderate number of observations and features.
- **randomized**: for datasets with both a large number of observations and features. This mode uses an approximation algorithm.

As the algorithm's last step, it performs the singular value decomposition on the unified solution, from which the principal components are then derived.

Mode 1: Regular

The workers jointly compute both $\sum x_i^T x_i$ and $\sum x_i$.

> **Note**
>
> Because x_i are `1 * d` row vectors, $x_i^T x_i$ is a matrix (not a scalar). Using row vectors within the code allows us to obtain efficient caching.

The covariance matrix is computed as $\sum x_i^T x_i - (1/n)(\sum x_i)^T \sum x_i$, and its top `num_components` singular vectors form the model.

> **Note**
>
> If `subtract_mean` is `False`, we avoid computing and subtracting $\sum x_i$.

Use this algorithm when the dimension `d` of the vectors is small enough so that d^2 can fit in memory.

Mode 2: Randomized

When the number of features in the input dataset is large, we use a method to approximate the covariance metric. For every mini-batch X_t of dimension `b * d`, we randomly initialize a `(num_components + extra_components) * b` matrix that we multiply by each mini-batch, to create a `(num_components + extra_components) * d` matrix. The sum of these matrices is computed by the workers, and the servers perform SVD on the final `(num_components + extra_components) * d` matrix. The top right `num_components` singular vectors of it are the approximation of the top singular vectors of the input matrix.

Let ℓ = `num_components + extra_components`. Given a mini-batch X_t of dimension `b * d`, the worker draws a random matrix H_t of dimension $\ell * b$. Depending on whether the environment uses a GPU or CPU and the dimension size, the matrix is either a random sign matrix where each entry is `+-1` or a *FJLT* (fast Johnson Lindenstrauss transform; for information, see FJLT Transforms and the follow-up papers). The worker then computes $H_t X_t$ and maintains $B = \sum H_t X_t$. The worker also maintains h^T , the sum of columns of $H_1, \ldots, H_T$ (`T` being the total number of mini-batches), and `s`, the sum of all input rows. After processing the entire shard of data, the worker sends the server `B`, `h`, `s`, and `n` (the number of input rows).

Denote the different inputs to the server as B^1, h^1, s^1, n^1 The server computes `B`, `h`, `s`, `n` the sums of the respective inputs. It then computes $C = B - (1/n)h^T s$, and finds its singular value decomposition. The top-right singular vectors and singular values of `C` are used as the approximate solution to the problem.

PCA Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific HyperParameters as string-to-string maps. The following table lists the hyperparameters for the PCA training algorithm provided by Amazon SageMaker. For more information about how PCA works, see How PCA Works (p. 223).

Parameter Name	Description
`feature_dim`	Input dimension. **Required** Valid values: positive integer
`mini_batch_size`	Number of rows in a mini-batch. **Required** Valid values: positive integer
`num_components`	The number of principal components to compute. **Required** Valid values: positive integer
`algorithm_mode`	Mode for computing the principal components. **Optional** Valid values: *regular* or *randomized* Default value: *regular*
`extra_components`	As the value increases, the solution becomes more accurate but the runtime and memory consumption increase linearly. The default, -1, means the maximum of 10 and `num_components`. Valid for *randomized* mode only. **Optional** Valid values: Non-negative integer or -1 Default value: -1
`subtract_mean`	Indicates whether the data should be unbiased both during training and at inference. **Optional** Valid values: One of *true* or *false* Default value: *true*

PCA Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker PCA algorithm.

JSON Response Format

Accept—application/json

```
{
    "projections": [
        {
            "projection": [1.0, 2.0, 3.0, 4.0, 5.0]
        },
        {
            "projection": [6.0, 7.0, 8.0, 9.0, 0.0]
        },
        ....
    ]
}
```

JSONLINES Response Format

Accept—application/jsonlines

```
{ "projection": [1.0, 2.0, 3.0, 4.0, 5.0] }
{ "projection": [6.0, 7.0, 8.0, 9.0, 0.0] }
```

RECORDIO Response Format

Accept—application/x-recordio-protobuf

```
[
    Record = {
        features = {},
        label = {
            'projection': {
                keys: [],
                values: [1.0, 2.0, 3.0, 4.0, 5.0]
            }
        }
    },
    Record = {
        features = {},
        label = {
            'projection': {
                keys: [],
                values: [1.0, 2.0, 3.0, 4.0, 5.0]
            }
        }
    }
]
```

Random Cut Forest (RCF) Algorithm

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a data set. These are observations which diverge from otherwise well-structured or

patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a data set can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

With each data point, RCF associates an anomaly score. Low score values indicate that the data point is considered "normal." High values indicate the presence of an anomaly in the data. The definitions of "low" and "high" depend on the application but common practice suggests that scores beyond three standard deviations from the mean score are considered anomalous.

While there are many applications of anomaly detection algorithms to one-dimensional time series data such as traffic volume analysis or sound volume spike detection, RCF is designed to work with arbitrary-dimensional input. Amazon SageMaker RCF scales well with respect to number of features, data set size, and number of instances.

Topics

- Input/Output Interface for the RCF Algorithm (p. 227)
- Instance Recommendations for the RCF Algorithm (p. 228)
- RCF Sample Notebooks (p. 228)
- How RCF Works (p. 228)
- RCF Hyperparameters (p. 231)
- Tune an RCF Model (p. 231)
- RCF Response Formats (p. 232)

Input/Output Interface for the RCF Algorithm

Amazon SageMaker Random Cut Forest supports the `train` and `test` data channels. The optional test channel is used to compute accuracy, precision, recall, and F1-score metrics on labeled data. Train and test data content types can be either `application/x-recordio-protobuf` or `text/csv` formats. For the test data, when using text/csv format, the content must be specified as text/csv;label_size=1 where the first column of each row represents the anomaly label: "1" for an anomalous data point and "0" for a normal data point. You can use either File mode or Pipe mode to train RCF models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`

Also note that the train channel only supports `S3DataDistributionType=ShardedByS3Key` and the test channel only supports `S3DataDistributionType=FullyReplicated`. The S3 distribution type can be specified using the Python SDK as follows:

```
import sagemaker

# specify Random Cut Forest training job information and hyperparameters
rcf = sagemaker.estimator.Estimator(...)

# explicitly specify "SharededByS3Key" distribution type
train_data = sagemaker.s3_input(
    s3_data=s3_training_data_location,
    content_type='text/csv;label_size=0',
    distribution='ShardedByS3Key')

# run the training job on input data stored in S3
rcf.fit({'train': train_data})
```

See the Amazon SageMaker Data Types documentation for more information on customizing the S3 data source attributes. Finally, in order to take advantage of multi-instance training the training data must be partitioned into at least as many files as instances.

For inference, RCF supports `application/x-recordio-protobuf`, `text/csv` and `application/json` input data content types. See the Common Data Formats for Built-in Algorithms (p. 64) documentation for more information. RCF inference returns `application/x-recordio-protobuf` or `application/json` formatted output. Each record in these output data contains the corresponding anomaly scores for each input data point. See Common Data Formats--Inference for more information.

For more information on input and output file formats, see RCF Response Formats (p. 232) for inference and the RCF Sample Notebooks (p. 228).

Instance Recommendations for the RCF Algorithm

For training, we recommend the `ml.m4`, `ml.c4`, and `ml.c5` instance families. For inference we recommend using a `ml.c5.xl` instance type in particular, for maximum performance as well as minimized cost per hour of usage. Although the algorithm could technically run on GPU instance types it does not take advantage of GPU hardware.

RCF Sample Notebooks

For an example of how to train an RCF model and perform inferences with it, see the Introduction to SageMaker Random Cut Forests notebook. For a sample notebook that uses the Amazon SageMaker Random Cut Forest algorithm for anomaly detection, see An Introduction to SageMaker Random Cut Forests. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its **Use** tab and select **Create copy**.

How RCF Works

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a dataset. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a dataset can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

The main idea behind the RCF algorithm is to create a forest of trees where each tree is obtained using a partition of a sample of the training data. For example, a random sample of the input data is first determined. The random sample is then partitioned according to the number of trees in the forest. Each tree is given such a partition and organizes that subset of points into a k-d tree. The anomaly score assigned to a data point by the tree is defined as the expected change in complexity of the tree as a result adding that point to the tree; which, in approximation, is inversely proportional to the resulting depth of the point in the tree. The random cut forest assigns an anomaly score by computing the average score from each constituent tree and scaling the result with respect to the sample size. The RCF algorithm is based on the one described in reference [1].

Sample Data Randomly

The first step in the RCF algorithm is to obtain a random sample of the training data. In particular, suppose we want a sample of size K from N total data points. If the training data is small enough, the entire dataset can be used, and we could randomly draw K elements from this set. However, frequently the training data is too large to fit all at once, and this approach isn't feasible. Instead, we use a technique called reservoir sampling.

Reservoir sampling is an algorithm for efficiently drawing random samples from a dataset $S = \{S_1, \ldots, S_N\}$ where the elements in the dataset can only be observed one at a time or in batches. In fact, reservoir sampling works even when N is not known *a priori*. If only one sample is requested, such as when $K = 1$, the algorithm is like this:

Algorithm: Reservoir Sampling

- Input: dataset or data stream $S = \{S_1, \ldots, S_N\}$
- Initialize the random sample $X = S_1$
- For each observed sample $S_n, n = 2, \ldots, N$:
  - Pick a uniform random number $\xi \in [0, 1]$
  - If $\xi < 1/n$
    - Set $X = S_n$
- Return X

This algorithm selects a random sample such that $P(X = S_n) = 1/N$ for all $n = 1, \ldots, N$. When $K > 1$ the algorithm is more complicated. Additionally, a distinction must be made between random sampling that is with and without replacement. RCF performs an augmented reservoir sampling without replacement on the training data based on the algorithms described in [2].

Train a RCF Model and Produce Inferences

The next step in RCF is to construct a random cut forest using the random sample of data. First, the sample is partitioned into a number of equal-sized partitions equal to the number of trees in the forest. Then, each partition is sent to an individual tree. The tree recursively organizes its partition into a binary tree by partitioning the data domain into bounding boxes.

This procedure is best illustrated with an example. Suppose a tree is given the following two-dimensional dataset. The corresponding tree is initialized to the root node:

A two-dimensional dataset where the majority of data lies in a cluster (blue) except for one anomalous data point (orange). The tree is initialized with a root node.

The RCF algorithm organizes these data in a tree by first computing a bounding box of the data, selecting a random dimension (giving more weight to dimensions with higher "variance"), and then randomly determining the position of a hyperplane "cut" through that dimension. The two resulting subspaces define their own sub tree. In this example, the cut happens to separate a lone point from the remainder of the sample. The first level of the resulting binary tree consists of two nodes, one which will consist of the subtree of points to the left of the initial cut and the other representing the single point on the right.
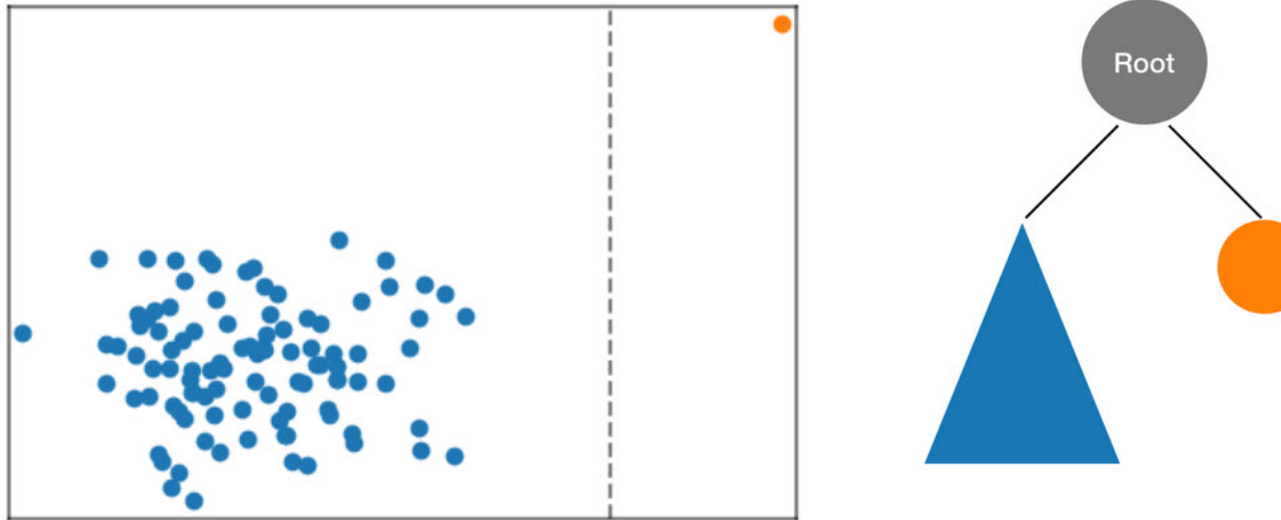
A random cut partitioning the two-dimensional dataset. An anomalous data point is more likely to lie isolated in a bounding box at a smaller tree depth than other points.

Bounding boxes are then computed for the left and right halves of the data and the process is repeated until every leaf of the tree represents a single data point from the sample. Note that if the lone point is sufficiently far away then it is more likely that a random cut would result in point isolation. This observation provides the intuition that tree depth is, loosely speaking, inversely proportional to the anomaly score.

When performing inference using a trained RCF model the final anomaly score is reported as the average across scores reported by each tree. Note that it is often the case that the new data point does not already reside in the tree. To determine the score associated with the new point the data point is inserted into the given tree and the tree is efficiently (and temporarily) reassembled in a manner equivalent to the training process described above. That is, the resulting tree is as if the input data point were a member of the sample used to construct the tree in the first place. The reported score is inversely proportional to the depth of the input point within the tree.

Choose Hyperparameters

The primary hyperparameters used to tune the RCF model are `num_trees` and `num_samples_per_tree`. Increasing `num_trees` has the effect of reducing the noise observed in anomaly scores since the final score is the average of the scores reported by each tree. While the optimal value is application-dependent we recommend using 100 trees to begin with as a balance between score noise and model complexity. Note that inference time is proportional to the number of trees. Although training time is also affected it is dominated by the reservoir sampling algorithm describe above.

The parameter `num_samples_per_tree` is related to the expected density of anomalies in the dataset. In particular, `num_samples_per_tree` should be chosen such that `1/num_samples_per_tree` approximates the ratio of anomalous data to normal data. For example, if 256 samples are used in each tree then we expect our data to contain anomalies 1/256 or approximately 0.4% of the time. Again, an optimal value for this hyperparameter is dependent on the application.

References

1. Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. "Robust random cut forest based anomaly detection on streams." In *International Conference on Machine Learning*, pp. 2712-2721. 2016.

2. Byung-Hoon Park, George Ostrouchov, Nagiza F. Samatova, and Al Geist. "Reservoir-based random sampling with replacement from data stream." In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 492-496. Society for Industrial and Applied Mathematics, 2004.

RCF Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker RCF algorithm. For more information, including recommendations on how to choose hyperparameters, see How RCF Works (p. 228).

Parameter Name	Description
`feature_dim`	The number of features in the data set. (If you are using the client libraries through a notebook, this value is calculated for you and need not be specified.) **Required** (When the job is run through the console.) Valid values: Positive integer (min: 1, max: 10000)
`eval_metrics`	A list of metrics used to score a labeled test data set. The following metrics can be selected for output: • `accuracy` - returns fraction of correct predictions. • `precision_recall_fscore` - returns the positive and negative precision, recall, and F1-scores. **Optional** Valid values: a list with possible values taken from `accuracy` or `precision_recall_fscore`. Default value: Both `accuracy`, `precision_recall_fscore` are calculated.
`num_samples_per_tree`	Number of random samples given to each tree from the training data set. **Optional** Valid values: Positive integer (min: 1, max: 2048) Default value: 256
`num_trees`	Number of trees in the forest. **Optional** Valid values: Positive integer (min: 50, max: 1000) Default value: 100

Tune an RCF Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable

hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker RCF algorithm is an unsupervised anomaly-detection algorithm that requires a labeled test dataset for hyperparameter optimization. It calculates anomaly scores for test datapoints and then labels the datapoints as anomalous if their scores are beyond three standard deviations from the mean score. This is known as the three-sigma limit heuristic. The F1 score is emitted based on the difference between calculated labels and actual labels. The hyperparameter tuning job finds the model that maximizes that score. The success of hyperparameter optimization depends on the applicability of the three-sigma limit heuristic to the test dataset.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the RCF Algorithm

The RCF algorithm computes the following metric during training. When tuning the model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
test:f1	F1 score on the test dataset, based on the difference between calculated labels and actual labels.	Maximize

Tunable RCF Hyperparameters

You can tune a RCF model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
num_samples_per_tree	IntegerParameterRanges	MinValue: 1, MaxValue:2048
num_trees	IntegerParameterRanges	MinValue: 50, MaxValue:1000

RCF Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. Note that Amazon SageMaker Random Cut Forest supports both dense and sparse JSON and RecordIO formats. This topic contains a list of the available output formats for the Amazon SageMaker RCF algorithm.

JSON Response Format

ACCEPT: application/json.

```
{


    "scores":    [
```

```
            {"score": 0.02},


            {"score": 0.25}



      ]



   }
```

JSONLINES Response Format

ACCEPT: application/jsonlines.

```
{"score": 0.02},
{"score": 0.25}
```

RECORDIO Response Format

ACCEPT: application/x-recordio-protobuf.

```
    [


        Record = {


            features = {},


            label = {


                'score': {


                    keys: [],


                    values: [0.25]  # float32


                }


            }
```

```
        },


    Record = {


        features = {},


        label = {


            'score': {


                keys: [],


                values: [0.23]  # float32


            }


        }


    }


]
```

Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm provides a fine-grained, pixel-level approach to developing computer vision applications. It tags every pixel in an image with a class label from a predefined set of classes. Tagging is fundamental for understanding scenes, which is critical to an increasing number of computer vision applications, such as self-driving vehicles, medical imaging diagnostics, and robot sensing.

For comparison, the Amazon SageMaker Image Classification Algorithm (p. 108) is a supervised learning algorithm that analyzes only whole images, classifying them into one of multiple output categories. The Object Detection Algorithm (p. 199) is a supervised learning algorithm that detects and classifies all instances of an object in an image. It indicates the location and scale of each object in the image with a rectangular bounding box.

Because the semantic segmentation algorithm classifies every pixel in an image, it also provides information about the shapes of the objects contained in the image. The segmentation output is represented as an RGB or grayscale image, called a *segmentation mask*. A segmentation mask is an RGB (or grayscale) image with the same shape as the input image.

Amazon SageMaker semantic segmentation algorithm is built using the MXNet Gluon framework and the Gluon CV toolkit provides you with a choice of three build-in algorithms to train a deep neural network. You can use the Fully-Convolutional Network (FCN) algorithm , Pyramid Scene Parsing (PSP) algorithm, or DeepLabV3.

Each of the three algorithms has two distinct components:

- The *backbone* (or *encoder*)—A network that produces reliable activation maps of features.
- The *decoder*—A network that constructs the segmentation mask from the encoded activation maps.

You also have a choice of backbones for the FCN, PSP, and DeepLabV3 algorithms: ResNet50 or ResNet101. These backbones include pretrained artifacts that were originally trained on the ImageNet classification task. You can fine-tune these backbones for segmentation using your own data. Or, you can initialize and train these networks from scratch using only your own data. The decoders are never pretrained.

To deploy the trained model for inference, use the Amazon SageMaker hosting service. During inference, you can request the segmentation mask either as a PNG image or as a set of probabilities for each class for each pixel. You can use these masks as part of a larger pipeline that includes additional downstream image processing or other applications.

Topics

- Semantic Segmentation Sample Notebooks (p. 235)
- Input/Output Interface for the Semantic Segmentation Algorithm (p. 235)
- EC2 Instance Recommendation for the Semantic Segmentation Algorithm (p. 238)
- Semantic Segmentation Hyperparameters (p. 238)

Semantic Segmentation Sample Notebooks

For a sample Jupyter notebook that uses the Amazon SageMaker semantic segmentation algorithm to train a model and deploy it to perform inferences, see the Semantic Segmentation Example. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36).

To see a list of all of the Amazon SageMaker samples, create and open a notebook instance, and choose the **SageMaker Examples** tab. The example semantic segmentation notebooks are located under **Introduction to Amazon algorithms**. To open a notebook, choose its **Use** tab, and choose **Create copy**.

Input/Output Interface for the Semantic Segmentation Algorithm

Amazon SageMaker semantic segmentation expects the customer's training dataset to be on Amazon Simple Storage Service (Amazon S3). Once trained, it produces the resulting model artifacts on Amazon S3. The input interface format for the Amazon SageMaker semantic segmentation is similar to that of most standardized semantic segmentation benchmarking datasets. The dataset in Amazon S3 is expected to be presented in two channels, one for `train` and one for `validation` using four directories, two for images and two for annotations. Annotations are expected to be uncompressed PNG images. The dataset might also have a label map that describes how the annotation mappings are established. If not, the algorithm uses a default. It also supports the augmented manifest image format (`application/x-image`) for training in Pipe input mode straight from Amazon S3. For inference, an endpoint accepts images with an `image/jpeg` content type.

How Training Works

The training data is split into four directories: `train`, `train_annotation`, `validation`, and `validation_annotation`. There is a channel for each of these directories. The dataset also expected to have one `label_map.json` file per channel for `train_annotation` and `validation_annotation` respectively. If you don't provide these JSON files, Amazon SageMaker provides the default set label map.

The dataset specifying these files should look similar to the following example:

```
s3://bucket_name
    |
    |- train
                |
                | - 0000.jpg
                | - coffee.jpg
    |- validation
                |
                | - 00a0.jpg
                | - bananna.jpg
    |- train_annotation
                |
                | - 0000.png
                | - coffee.png
    |- validation_annotation
                |
                | - 00a0.png
                | - bananna.png
    |- label_map
                | - train_label_map.json
                | - validation_label_map.json
```

Every JPG image in the train and validation directories has a corresponding PNG label image with the same name in the `train_annotation` and `validation_annotation` directories. This naming convention helps the algorithm to associate a label with its corresponding image during training. The `train`, `train_annotation`, `validation`, and `validation_annotation` channels are mandatory. The annotations are single-channel PNG images. The format works as long as the metadata (modes) in the image helps the algorithm read the annotation images into a single-channel 8-bit unsigned integer. For more information on our support for modes, see the Python Image Library documentation. We recommend using the 8-bit pixel, true color `P` mode.

The image that is encoded is a simple 8-bit integer when using modes. To get from this mapping to a map of a label, the algorithm uses one mapping file per channel, called the *label map*. The label map is used to map the values in the image with actual label indices. In the default label map, which is provided by default if you don't provide one, the pixel value in an annotation matrix (image) directly index the label. These images can be grayscale PNG files or 8-bit indexed PNG files. The label map file for the unscaled default case is the following:

```
{
  "scale": "1"
}
```

To provide some contrast for viewing, some annotation software scales the label images by a constant amount. To support this, the Amazon SageMaker semantic segmentation algorithm provides a rescaling option to scale down the values to actual label values. When scaling down doesn't convert the value to an appropriate integer, the algorithm defaults to the greatest integer less than or equal to the scale value. The following code shows how to set the scale value to rescale the label values:

```
{
  "scale": "3"
```

```
}
```

The following example shows how this `"scale"` value is used to rescale the `encoded_label` values of the input annotation image when they are mapped to the `mapped_label` values to be used in training. The label values in the input annotation image are 0, 3, 6, with scale 3, so they are mapped to 0, 1, 2 for training:

```
encoded_label = [0, 3, 6]
mapped_label = [0, 1, 2]
```

In some cases, you might need to specify a particular color mapping for each class. Use the map option in the label mapping as shown in the following example of a `label_map` file:

```
{
    "map": {
        "0": 5,
        "1": 0,
        "2": 2
    }
}
```

This label mapping for this example is:

```
encoded_label = [0, 5, 2]
mapped_label = [1, 0, 2]
```

With label mappings, you can use different annotation systems and annotation software to obtain data without a lot of preprocessing. You can provide one label map per channel. The files for a label map in the `label_map` channel must follow the naming conventions for the four directory structure. If you don't provide a label map, the algorithm assumes a scale of 1 (the default).

Training with the Augmented Manifest Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. The augmented manifest file contains data objects and should be in JSON Lines format, as described in the CreateTrainingJob (p. 667) request API. Each line in the manifest is an entry containing the Amazon S3 URI for the image and the URI for the annotation image.

Each JSON object in the manifest file must contain a `source-ref` key. The `source-ref` key should contain the value of the Amazon S3 URI to the image. The labels are provided under the `AttributeNames` parameter value as specified in the CreateTrainingJob (p. 667) request. It can also contain additional metadata under the metadata tag, but these are ignored by the algorithm. In the example below, the `AttributeNames` are contained in the list of image and annotation references `["source-ref", "city-streets-ref"]`. These names must have `-ref` appended to them. When using the Semantic Segmentation algorithm with Augmented Manifest, the value of the `RecordWrapperType` parameter must be `"RecordIO"` and value of the `ContentType` parameter must be `application/x-recordio`.

```
{"source-ref": "S3 bucket location", "city-streets-ref": "S3 bucket location", "city-
streets-metadata": {"job-name": "label-city-streets", }}
```

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 308).

Incremental Training

You can also seed the training of a new model with a model that you trained previously using Amazon SageMaker. This incremental training saves training time when you want to train a new model with the

same or similar data. Currently, incremental training is supported only for models trained with the built-in Amazon SageMaker Semantic Segmentation.

To use your own pre-trained model, specify the `ChannelName` as "model" in the `InputDataConfig` for the CreateTrainingJob (p. 667) request. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The `backbone`, `algorithm`, `crop_size`, and `num_classes` input parameters that define the network architecture must be consistently specified in the input hyperparameters of the new model and the pre-trained model that you upload to the model channel. For the pretrained model file, you can use the compressed (.tar.gz) artifacts from Amazon SageMaker outputs. You can use either RecordIO or Image formats for input data. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 282).

Produce Inferences

To query a trained model that is deployed to an endpoint, you need to provide an image and an `AcceptType` that denotes the type of output required. The endpoint takes JPEG images with an `image/jpeg` content type. If you request an `AcceptType` of `image/png`, the algorithm outputs a PNG file with a segmentation mask in the same format as the labels themselves. If you request an accept type of `application/x-recordio-protobuf`, the algorithm returns class probabilities encoded in recordio-protobuf format. The latter format outputs a 3D tensor where the third dimension is the same size as the number of classes. This component denotes the probability of each class label for each pixel.

EC2 Instance Recommendation for the Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm only supports GPU instances for training, and we recommend using GPU instances with more memory for training with large batch sizes. The algorithm can be trained using P2/P3 EC2 Amazon Elastic Compute Cloud (Amazon EC2) instances in single machine configurations. It supports the following GPU instances for training:

- `ml.p2.xlarge`
- `ml.p2.8xlarge`
- `ml.p2.16xlarge`
- `ml.p3.2xlarge`
- `ml.p3.8xlarge`
- `ml.p3.16xlarge`

For inference, you can use either CPU instances (such as c5 and m5) and GPU instances (such as p2 and p3) or both. For information about the instance types that provide varying combinations of CPU, GPU, memory, and networking capacity for inference, see Amazon SageMaker ML Instance Types.

Semantic Segmentation Hyperparameters

The following tables list the hyperparameters supported by the Amazon SageMaker semantic segmentation algorithm for network architecture, data inputs, and training. You specify Semantic Segmentation for training in the `AlgorithmName` of the CreateTrainingJob (p. 667) request.

Network Architecture Hyperparameters

Parameter Name	Description
backbone	The backbone to use for the algorithm's encoder component.

Parameter Name	Description
	Optional Valid values: `resnet-50`, `resnet-101` Default value: `resnet-50`
use_pretrained_model	Whether a pretrained model is to be used for the backbone. **Optional** Valid values: `True`, `False` Default value: `True`
algorithm	The algorithm to use for semantic segmentation. **Optional** Valid values: • `fcn`: Fully-Convolutional Network (FCN) algorithm • `psp`: Pyramid Scene Parsing (PSP) algorithm • `deeplab`: DeepLab V3 algorithm Default value: `fcn`

Data Hyperparameters

Parameter Name	Description
num_classes	The number of classes to segment. **Required** Valid values: 2 ≤ positive integer ≤ 254
num_training_samples	The number of samples in the training data. The algorithm uses this value to set up the learning rate scheduler. **Required** Valid values: positive integer
crop_size	The image size for input images. We rescale the input image to a square image to this `crop_size`. We do this by rescaling the shorter side to match this parameter while maintaining the aspect ratio, and then take a random crop along the longer side. **Optional** Valid values: positive integer > 16 Default value: 480

Training Hyperparameters

Parameter Name	Description
`early_stopping`	Whether to use early stopping logic during training. **Optional** Valid values: `True`, `False` Default value: `False`
`early_stopping_min_epochs`	The minimum number of epochs that must be run. **Optional** Valid values: integer Default value: 5
`early_stopping_patience`	The number of epochs that meet the tolerance for lower performance before the algorithm enforces an early stop. **Optional** Valid values: integer Default value: 4
`early_stopping_tolerance`	If the relative improvement of the score of the training job, the mIOU, is smaller than this value, early stopping considers the epoch as not improved. This is used only when `early_stopping = True`. **Optional** Valid values: 0 ≤ float ≤ 1 Default value: 0.0
`epochs`	The number of epochs with which to train. **Optional** Valid values: positive integer Default value: 30
`gamma1`	The decay factor for the moving average of the squared gradient for `rmsprop`. Used only for `rmsprop`. **Optional** Valid values: 0 ≤ float ≤ 1 Default value: 0.9
`gamma2`	The momentum factor for `rmsprop`. **Optional** Valid values: 0 ≤ float ≤ 1 Default value: 0.9

Parameter Name	Description
learning_rate	The initial learning rate. **Optional** Valid values: 0 < float ≤ 1 Default value: 0.001
lr_scheduler	The shape of the learning rate schedule that controls its decrease over time. **Optional** Valid values: • step: A stepwise decay, where the learning rate is reduced by a factor at certain intervals. • poly: A smooth decay using a polynomial function. • cosine: A smooth decay using a cosine function. Default value: poly
mini_batch_size	The batch size for training. Using a large mini_batch_size usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the mini_batch_size and image_shape parameters, and the backbone architecture. **Optional** Valid values: positive integer Default value: 4
momentum	The momentum for the sgd optimizer. When you use other optimizers, the semantic segmentation algorithm ignores this parameter. **Optional** Valid values: 0 < float ≤ 1 Default value: 0.9

Parameter Name	Description
`optimizer`	The type of optimizer. For more information about an optimizer, choose the appropriate link: • `adam`: Adaptive momentum estimation • `adagrad`: Adaptive gradient descent • `nag`: Nesterov accelerated gradient • `rmsprop`: Root mean square propagation • `sgd`: Stochastic gradient descent **Optional** Valid values: `adam`, `adagrad`, `nag`, `rmsprop`, `sgd` Default value: `sgd`

`validation_mini_batch_size`	The batch size for validation. A large `mini_batch_size` usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the `mini_batch_size` and `image_shape` parameters, and the backbone architecture.
	
	• To score the validation on the entire image without cropping the images, set this parameter to 1. Use this option if you want to measure performance on the entire image as a whole.
	
	    Note
	    Setting the `validation_mini_batch_size` parameter to 1 causes the algorithm to create a new network model for every image. This might slow validation and training.
	• To crop images to the size specified in the `crop_size` parameter, even during evaluation, set this parameter to a value greater than 1.
	
	Optional
	
	Valid values: positive integer
	
	Default value: 4

`weight_decay`	The weight decay coefficient for the `sgd` optimizer. When you use other optimizers, the algorithm ignores this parameter. **Optional** Valid values: 0 < float < 1 Default value: 0.0001

Sequence-to-Sequence Algorithm

Amazon SageMaker Sequence to Sequence is a supervised learning algorithm where the input is a sequence of tokens (for example, text, audio) and the output generated is another sequence of tokens. Example applications include: machine translation (input a sentence from one language and predict what that sentence would be in another language), text summarization (input a longer string of words and predict a shorter string of words that is a summary), speech-to-text (audio clips converted into output

sentences in tokens). Recently, problems in this domain have been successfully modeled with deep neural networks that show a significant performance boost over previous methodologies. Amazon SageMaker seq2seq uses Recurrent Neural Networks (RNNs) and Convolutional Neural Network (CNN) models with attention as encoder-decoder architectures.

Topics

Input/Output Interface for the Sequence-to-Sequence Algorithm

Training

Amazon SageMaker seq2seq expects data in RecordIO-Protobuf format. However, the tokens are expected as integers, not as floating points, as is usually the case.

A script to convert data from tokenized text files to the protobuf format is included in the seq2seq example notebook. In general, it packs the data into 32-bit integer tensors and generates the necessary vocabulary files, which are needed for metric calculation and inference.

After preprocessing is done, the algorithm can be invoked for training. The algorithm expects three channels:

- `train`: It should contain the training data (for example, the `train.rec` file generated by the preprocessing script).
- `validation`: It should contain the validation data (for example, the `val.rec` file generated by the preprocessing script).
- `vocab`: It should contain two vocabulary files (`vocab.src.json` and `vocab.trg.json`)

If the algorithm doesn't find data in any of these three channels, training results in an error.

Inference

For hosted endpoints, inference supports two data formats. To perform inference using space separated text tokens, use the `application/json` format. Otherwise, use the `recordio-protobuf` format to work with the integer encoded data. Both mode supports batching of input data. `application/json` format also allows you to visualize the attention matrix.

- `application/json`: Expects the input in JSON format and returns the output in JSON format. Both content and accept types should be `application/json`. Each sequence is expected to be a string with whitespace separated tokens. This format is recommended when the number of source sequences in the batch is small. It also supports the following additional configuration options:

  `configuration`: {`attention_matrix: true`}: Returns the attention matrix for the particular input sequence.
- `application/x-recordio-protobuf`: Expects the input in `recordio-protobuf` format and returns the output in `recordio-protobuf format`. Both content and accept types should be `applications/x-recordio-protobuf`. For this format, the source sequences must be converted

into a list of integers for subsequent protobuf encoding. This format is recommended for bulk inference.

For batch transform, inference supports JSON Lines format. Batch transform expects the input in JSON Lines format and returns the output in JSON Lines format. Both content and accept types should be `application/jsonlines`. The format for input is as follows:

```
content-type: application/jsonlines

{"source": "source_sequence_0"}
{"source": "source_sequence_1"}
```

The format for response is as follows:

```
accept: application/jsonlines

{"target": "predicted_sequence_0"}
{"target": "predicted_sequence_1"}
```

For additional details on how to serialize and deserialize the inputs and outputs to specific formats for inference, see the Sequence-to-Sequence Sample Notebooks (p. 244) .

EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm

Currently Amazon SageMaker seq2seq is only supported on GPU instance types and is only set up to train on a single machine. But it does also offer support for multiple GPUs.

Sequence-to-Sequence Sample Notebooks

For a sample notebook that shows how to use the Amazon SageMaker Sequence to Sequence algorithm to train a English-German translation model, see Machine Translation English-German Example Using SageMaker Seq2Seq. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Sequence-to-Sequence Works

Typically, a neural network for sequence-to-sequence modeling consists of a few layers, including:

- An **embedding layer**. In this layer, the input matrix, which is input tokens encoded in a sparse way (for example, one-hot encoded) are mapped to a dense feature layer. This is required because a high-dimensional feature vector is more capable of encoding information regarding a particular token (word for text corpora) than a simple one-hot-encoded vector. It is also a standard practice to initialize this embedding layer with a pre-trained word vector like FastText or Glove or to initialize it randomly and learn the parameters during training.
- An **encoder layer**. After the input tokens are mapped into a high-dimensional feature space, the sequence is passed through an encoder layer to compress all the information from the input embedding layer (of the entire sequence) into a fixed-length feature vector. Typically, an encoder is made of RNN-type networks like long short-term memory (LSTM) or gated recurrent units (GRU). (Colah's blog explains LSTM in a great detail.)
- A **decoder layer**. The decoder layer takes this encoded feature vector and produces the output sequence of tokens. This layer is also usually built with RNN architectures (LSTM and GRU).

The whole model is trained jointly to maximize the probability of the target sequence given the source sequence. This model was first introduced by Sutskever et al. in 2014.

Attention mechanism. The disadvantage of an encoder-decoder framework is that model performance decreases as and when the length of the source sequence increases because of the limit of how much information the fixed-length encoded feature vector can contain. To tackle this problem, in 2015, Bahdanau et al. proposed the attention mechanism. In an attention mechanism, the decoder tries to find the location in the encoder sequence where the most important information could be located and uses that information and previously decoded words to predict the next token in the sequence.

For more in details, see the whitepaper Effective Approaches to Attention-based Neural Machine Translation by Luong, et al. that explains and simplifies calculations for various attention mechanisms. Additionally, the whitepaper Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation by Wu, et al. describes Google's architecture for machine translation, which uses skip connections between encoder and decoder layers.

Sequence-to-Sequence Hyperparameters

Parameter Name	Description
`batch_size`	Mini batch size for gradient descent. **Optional** Valid values: positive integer Default value: 64
`beam_size`	Length of the beam for beam search. Used during training for computing `bleu` and used during inference. **Optional** Valid values: positive integer Default value: 5
`bleu_sample_size`	Number of instances to pick from validation dataset to decode and compute `bleu` score during training. Set to -1 to use full validation set (if `bleu` is chosen as `optimized_metric`). **Optional** Valid values: integer Default value: 0
`bucket_width`	Returns (source,target) buckets up to (`max_seq_len_source`, `max_seq_len_target`). The longer side of the data uses steps of `bucket_width` while the shorter side uses steps scaled down by the average target/source length ratio. If one sided reaches its maximum length before the other, width of extra buckets on that side is fixed to that side of `max_len`. **Optional** Valid values: positive integer

Parameter Name	Description
	Default value: 10
`bucketing_enabled`	Set to `false` to disable bucketing, unroll to maximum length. **Optional** Valid values: `true` or `false` Default value: `true`
`checkpoint_frequency_num_batches`	Checkpoint and evaluate every x batches. **Optional** Valid values: positive integer Default value: 1000
`checkpoint_threshold`	Maximum number of checkpoints model is allowed to not improve in `optimized_metric` on validation dataset before training is stopped. **Optional** Valid values: positive integer Default value: 3
`clip_gradient`	Clip absolute gradient values greater than this. Set to negative to disable. **Optional** Valid values: float Default value: 1
`cnn_activation_type`	The `cnn` activation type to be used. **Optional** Valid values: String. One of `glu`, `relu`, `softrelu`, `sigmoid`, or `tanh`. Default value: `glu`
`cnn_hidden_dropout`	Dropout probability for dropout between convolutional layers. **Optional** Valid values: Float. Range in [0,1]. Default value: 0

Parameter Name	Description
cnn_kernel_width_decoder	Kernel width for the cnn decoder. **Optional** Valid values: positive integer Default value: 5
cnn_kernel_width_encoder	Kernel width for the cnn encoder. **Optional** Valid values: positive integer Default value: 3
cnn_num_hidden	Number of cnn hidden units for encoder and decoder. **Optional** Valid values: positive integer Default value: 512
decoder_type	Decoder type. **Optional** Valid values: String. Either rnn or cnn. Default value: *rnn*
embed_dropout_source	Dropout probability for source side embeddings. **Optional** Valid values: Float. Range in [0,1]. Default value: 0
embed_dropout_target	Dropout probability for target side embeddings. **Optional** Valid values: Float. Range in [0,1]. Default value: 0
encoder_type	Encoder type. The rnn architecture is based on attention mechanism by Bahdanau et al. and *cnn* architecture is based on Gehring et al. **Optional** Valid values: String. Either rnn or cnn. Default value: rnn

Parameter Name	Description
`fixed_rate_lr_half_life`	Half life for learning rate in terms of number of checkpoints for `fixed_rate_*` schedulers. **Optional** Valid values: positive integer Default value: 10
`learning_rate`	Initial learning rate. **Optional** Valid values: float Default value: 0.0003
`loss_type`	Loss function for training. **Optional** Valid values: String. `cross-entropy` Default value: `cross-entropy`
`lr_scheduler_type`	Learning rate scheduler type. `plateau_reduce` means reduce the learning rate whenever `optimized_metric` on `validation_accuracy` plateaus. `inv_t` is inverse time decay. `learning_rate/(1+decay_rate*t)` **Optional** Valid values: String. One of `plateau_reduce`, `fixed_rate_inv_t`, or `fixed_rate_inv_sqrt_t`. Default value: `plateau_reduce`
`max_num_batches`	Maximum number of updates/batches to process. -1 for infinite. **Optional** Valid values: integer Default value: -1
`max_num_epochs`	Maximum number of epochs to pass through training data before fitting is stopped. Training continues until this number of epochs even if validation accuracy is not improving if this parameter is passed. Ignored if not passed. **Optional** Valid values: Positive integer and less than or equal to max_num_epochs. Default value: none

Parameter Name	Description
max_seq_len_source	Maximum length for the source sequence length. Sequences longer than this length are truncated to this length. **Optional** Valid values: positive integer Default value: 100
max_seq_len_target	Maximum length for the target sequence length. Sequences longer than this length are truncated to this length. **Optional** Valid values: positive integer Default value: 100
min_num_epochs	Minimum number of epochs the training must run before it is stopped via `early_stopping` conditions. **Optional** Valid values: positive integer Default value: 0
momentum	Momentum constant used for `sgd`. Don't pass this parameter if you are using `adam` or `rmsprop`. **Optional** Valid values: float Default value: none
num_embed_source	Embedding size for source tokens. **Optional** Valid values: positive integer Default value: 512
num_embed_target	Embedding size for target tokens. **Optional** Valid values: positive integer Default value: 512

Parameter Name	Description
`num_layers_decoder`	Number of layers for Decoder *rnn* or *cnn*. **Optional** Valid values: positive integer Default value: 1
`num_layers_encoder`	Number of layers for Encoder `rnn` or `cnn`. **Optional** Valid values: positive integer Default value: 1
`optimized_metric`	Metrics to optimize with early stopping. **Optional** Valid values: String. One of `perplexity`, `accuracy`, or `bleu`. Default value: `perplexity`
`optimizer_type`	Optimizer to choose from. **Optional** Valid values: String. One of `adam`, `sgd`, or `rmsprop`. Default value: `adam`
`plateau_reduce_lr_factor`	Factor to multiply learning rate with (for `plateau_reduce`). **Optional** Valid values: float Default value: 0.5
`plateau_reduce_lr_threshold`	For `plateau_reduce` scheduler, multiply learning rate with reduce factor if `optimized_metric` didn't improve for this many checkpoints. **Optional** Valid values: positive integer Default value: 3

Parameter Name	Description
rnn_attention_in_upper_layers	Pass the attention to upper layers of *rnn*, like Google NMT paper. Only applicable if more than one layer is used.
	Optional
	Valid values: boolean (`true` or `false`)
	Default value: `true`
rnn_attention_num_hidden	Number of hidden units for attention layers. defaults to `rnn_num_hidden`.
	Optional
	Valid values: positive integer
	Default value: `rnn_num_hidden`
rnn_attention_type	Attention model for encoders. `mlp` refers to concat and bilinear refers to general from the Luong et al. paper.
	Optional
	Valid values: String. One of `dot`, `fixed`, `mlp`, or `bilinear`.
	Default value: `mlp`
rnn_cell_type	Specific type of `rnn` architecture.
	Optional
	Valid values: String. Either `lstm` or `gru`.
	Default value: `lstm`
rnn_decoder_state_init	How to initialize `rnn` decoder states from encoders.
	Optional
	Valid values: String. One of `last`, `avg`, or `zero`.
	Default value: `last`
rnn_first_residual_layer	First *rnn* layer to have a residual connection, only applicable if number of layers in encoder or decoder is more than 1.
	Optional
	Valid values: positive integer
	Default value: 2

Parameter Name	Description
rnn_num_hidden	The number of *rnn* hidden units for encoder and decoder. This must be a multiple of 2 because the algorithm uses bi-directional Long Term Short Term Memory (LSTM) by default. **Optional** Valid values: positive even integer Default value: 1024
rnn_residual_connections	Add residual connection to stacked *rnn*. Number of layers should be more than 1. **Optional** Valid values: boolean (`true` or `false`) Default value: `false`
rnn_decoder_hidden_dropout	Dropout probability for hidden state that combines the context with the *rnn* hidden state in the decoder. **Optional** Valid values: Float. Range in [0,1]. Default value: 0
training_metric	Metrics to track on training on validation data. **Optional** Valid values: String. Either `perplexity` or `accuracy`. Default value: `perplexity`
weight_decay	Weight decay constant. **Optional** Valid values: float Default value: 0
weight_init_scale	Weight initialization scale (for `uniform` and `xavier` initialization). **Optional** Valid values: float Default value: 2.34

Parameter Name	Description
`weight_init_type`	Type of weight initialization. **Optional** Valid values: String. Either `uniform` or `xavier`. Default value: `xavier`
`xavier_factor_type`	Xavier factor type. **Optional** Valid values: String. One of `in`, `out`, or `avg`. Default value: `in`

Tune a Sequence-to-Sequence Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the Sequence-to-Sequence Algorithm

The sequence to sequence algorithm reports three metrics that are computed during training. Choose one of them as an objective to optimize when tuning the hyperparameter values.

Metric Name	Description	Optimization Direction
`validation:accuracy`	Accuracy computed on the validation dataset.	Maximize
`validation:bleu`	Bleu score computed on the validation dataset. Because BLEU computation is expensive, you can choose to compute BLEU on a random subsample of the validation dataset to speed up the overall training process. Use the `bleu_sample_size` parameter to specify the subsample.	Maximize
`validation:perplexity`	Perplexity, is a loss function computed on the validation dataset. Perplexity measures the cross-entropy between an empirical sample and the distribution predicted by a model and so provides a measure of how well a model predicts the sample values, Models that are good at predicting a sample have a low perplexity.	Minimize

Tunable Sequence-to-Sequence Hyperparameters

You can tune the following hyperparameters for the Amazon SageMaker Sequence to Sequence algorithm. The hyperparameters that have the greatest impact on sequence to sequence objective

metrics are: `batch_size`, `optimizer_type`, `learning_rate`, `num_layers_encoder`, and `num_layers_decoder`.

Parameter Name	Parameter Type	Recommended Ranges
`num_layers_encoder`	IntegerParameterRange	[1-10]
`num_layers_decoder`	IntegerParameterRange	[1-10]
`batch_size`	CategoricalParameterRange	[16,32,64,128,256,512,1024,2048]
`optimizer_type`	CategoricalParameterRange	['adam', 'sgd', 'rmsprop']
`weight_init_type`	CategoricalParameterRange	['xavier', 'uniform']
`weight_init_scale`	ContinuousParameterRange	For the xavier type: MinValue: 2.0, MaxValue: 3.0 For the uniform type: MinValue: -1.0, MaxValue: 1.0
`learning_rate`	ContinuousParameterRange	MinValue: 0.00005, MaxValue: 0.2
`weight_decay`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.1
`momentum`	ContinuousParameterRange	MinValue: 0.5, MaxValue: 0.9
`clip_gradient`	ContinuousParameterRange	MinValue: 1.0, MaxValue: 5.0
`rnn_num_hidden`	CategoricalParameterRange	Applicable only to recurrent neural networks (RNNs). [128,256,512,1024,2048]
`cnn_num_hidden`	CategoricalParameterRange	Applicable only to convolutional neural networks (CNNs). [128,256,512,1024,2048]
`num_embed_source`	IntegerParameterRange	[256-512]
`num_embed_target`	IntegerParameterRange	[256-512]
`embed_dropout_source`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
`embed_dropout_target`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
`rnn_decoder_hidden_dropout`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
`cnn_hidden_dropout`	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5

Parameter Name	Parameter Type	Recommended Ranges
`lr_scheduler_type`	CategoricalParameterRange	['plateau_reduce', 'fixed_rate_inv_t', 'fixed_rate_inv_sqrt_t']
`plateau_reduce_lr_factor`	ContinuousParameterRange	MinValue: 0.1, MaxValue: 0.5
`plateau_reduce_lr_threshold`	IntegerParameterRange	[1-5]
`fixed_rate_lr_half_life`	IntegerParameterRange	[10-30]

XGBoost Algorithm

The XGBoost (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining an ensemble of estimates from a set of simpler, weaker models. XGBoost has done remarkably well in machine learning competitions because it robustly handles a variety of data types, relationships, and distributions, and the large number of hyperparameters that can be tweaked and tuned for improved the fit. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

This current release of the XGBoost algorithm makes upgrades from the open source XGBoost code base easy to install and use in Amazon SageMaker. Customers can use this release of the XGBoost algorithm either as an Amazon SageMaker built-in algorithm, as with the previous 0.72-based version, or as a framework to run training scripts in their local environments as they would typically do, for example, with a TensorFlow deep learning framework. This implementation has a smaller memory footprint, better logging, improved hyperparameter validation, and an expanded set of metrics than the original 0.72-based version. It also provides an XGBoost `estimator` that executes a training script in a managed XGBoost environment. The current release of Amazon SageMaker XGBoost is based on version 0.90 and will be upgradeable to future releases. The previous implementation XGBoost Release 0.72 (p. 266) is still available to customers if they need to postpone migrating to the current version. But this previous implementation will remain tied to the 0.72 release of XGBoost.

Topics

How to Use Amazon SageMaker XGBoost

The XGBoost algorithm can be used as a built-in algorithm or as a framework such TensorFlow. Using XGBoost as a framework provides more flexible than using it as a built-in algorithm as it enables more advanced scenarios that allow pre-processing and post-processing scripts to be incorporated into your training script. Using XGBoost as a built-in Amazon SageMaker algorithm is how you had to use the original XGBoost Release 0.72 (p. 266) version and nothing changes here except the version of XGBoost that you use.

- **Use XGBoost as a framework**

  Use XGBoost as a framework to run scripts that can incorporate additional data processing into your training jobs. This way of using XGBoost should be to familiar to users who have worked with the open source XGBoost and other Amazon SageMaker frameworks such as Scikit-learn. You use the Amazon SageMaker Python SDK as you would for other frameworks such as TensorFlow. One change from other Amazon SageMaker frameworks is that the `framework_version` field of the `estimator` for `XGBoost` is mandatory and is not set by default. Note that the first part of the version refers to the upstream module version (aka, 0.90), while the second part refers to the Amazon SageMaker version for the container. An error is generated if the `framework_version` is not set.

  ```
  import sagemaker.XGBoost
  estimator = XGBoost(entry_point = 'myscript.py',
                      source_dir, model_dir, train_instance_type,
                      train_instance_count, hyperparameters, role, base_job_name,
                      framework_version = '0.90-1',
                      py_version)
  estimator.fit({'train':'s3://my-bucket/training',
                 'validation':'s3://my-bucket/validation'})
  ```

  The AWS SDK for Python (Boto 3) and the CLI also require this field.

- **Use XGBoost as a built-in algorithm**

  Use XGBoost to train and deploy a model as you would other built-in Amazon SageMaker algorithms. Using the current version of XGBoost as a built-in algorithm will be familiar to users who have used the original XGBoost Release 0.72 (p. 266) version with the Amazon SageMaker Python SDK and want to continue using the same procedures.

  ```
  import sagemaker
  import sagemaker.xgboost_utils
  # get the URI for new container
  container = get_image_uri(boto3.Session().region_name,
                            'xgboost',
                            repo_version='0.90-1');
  estimator = sagemaker.estimator.Estimator(container, role, instance_count, instance_type,
   train_volume_size, output_path, sagemaker.Session());
  estimator.fit({'train':'s3://my-bucket/training', 'validation':'s3://my-bucket/
  validation})
  ```

  If customers do not specify version in the `get_image_uri` function, they get the XGBoost Release 0.72 (p. 266) version by default. If you want to migrate to the current version, you have to specify `repo_version='0.90-1'` in the `get_image_uri` function. If you use the current version, you must update your code to use the new hyperparameters that are required by the 0.90 version of upstream algorithm. The AWS SDK for Python (Boto 3) and the CLI usage is similar. You also have to choose the version you want to run when using the Console to select the XGBoost algorithm.

Input/Output Interface for the XGBoost Algorithm

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The Amazon SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.
- For Inference ContentType, valid inputs are *text/libsvm* or (the default) *text/csv*.

Note
For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.
For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format: <label> <index0>:<value0> <index1>:<value1> ... Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other Amazon SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count * the memory available in the `InstanceType`) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

To use a model trained with SageMaker XGBoost in open source XGBoost

- Use the following Python code:

```
import pickle as pkl
model = pkl.load(open(model_file_path, 'rb'))
# prediction with test data
pred = model.predict(dtest)
```

To differentiate the importance of labelled data points use Instance Weight Supports

- Amazon SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For *text/libsvm* input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1...`. For *text/csv* input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`).

EC2 Instance Recommendation for the XGBoost Algorithm

Amazon SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M4) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

XGBoost Sample Notebooks

For a sample notebook that shows how to use Amazon SageMaker XGBoost as a built-in algorithm to train and host a regression model, see Regression with the Amazon SageMaker XGBoost algorithm. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How XGBoost Works

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using gradient boosting for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

For more detail on XGBoost, see:

- XGBoost: A Scalable Tree Boosting System
- Introduction to Boosted Trees

XGBoost Hyperparameters

The following table contains the subset of hyperparameters that are required or most commonly used for the Amazon SageMaker XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The Amazon SageMaker XGBoost algorithm is an implementation of the open-source DLMC XGBoost package. Currently Amazon SageMaker supports version 0.90. For details about full set of hyperparameter that can be configured for this version of XGBoost, see XGBoost Parameters.

Parameter Name	Description
num_round	The number of rounds to run the training. **Required** Valid values: integer
num_class	The number of classes. **Required** if `objective` is set to *multi:softmax* or *multi:softprob*. Valid values: integer Default value:
alpha	L1 regularization term on weights. Increasing this value makes models more conservative. **Optional** Valid values: float Default value: 0
base_score	The initial prediction score of all instances, global bias. **Optional**

Parameter Name	Description
	Valid values: float
	Default value: 0.5
`booster`	Which booster to use. The `gbtree` and `dart` values use a tree-based model, while `gblinear` uses a linear function.
	Optional
	Valid values: String. One of `gbtree`, `gblinear`, or `dart`.
	Default value: `gbtree`
`colsample_bylevel`	Subsample ratio of columns for each split, in each level.
	Optional
	Valid values: Float. Range: [0,1].
	Default value: 1
`colsample_bynode`	Subsample ratio of columns from each node.
	Optional
	Valid values: Float. Range: (0,1].
	Default value: 1
`colsample_bytree`	Subsample ratio of columns when constructing each tree.
	Optional
	Valid values: Float. Range: [0,1].
	Default value: 1
`csv_weights`	When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights.
	Optional
	Valid values: 0 or 1
	Default value: 0
`early_stopping_rounds`	The model trains until the validation score stops improving. Validation error needs to decrease at least every `early_stopping_rounds` to continue training. Amazon SageMaker hosting uses the best model for inference.
	Optional
	Valid values: integer
	Default value: -

Parameter Name	Description
`eta`	Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The `eta` parameter actually shrinks the feature weights to make the boosting process more conservative. **Optional** Valid values: Float. Range: [0,1]. Default value: 0.3
`eval_metric`	Evaluation metrics for validation data. A default metric is assigned according to the objective: • `rmse`: for regression • `error`: for classification • `map`: for ranking For a list of valid inputs, see XGBoost Parameters. **Optional** Valid values: string Default value: Default according to objective.
`gamma`	Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is. **Optional** Valid values: Float. Range: [0,∞). Default value: 0
`grow_policy`	Controls the way that new nodes are added to the tree. Currently supported only if `tree_method` is set to `hist`. **Optional** Valid values: String. Either `depthwise` or `lossguide`. Default value: `depthwise`
`lambda`	L2 regularization term on weights. Increasing this value makes models more conservative. **Optional** Valid values: float Default value: 1

Parameter Name	Description
lambda_bias	L2 regularization term on bias. **Optional** Valid values: Float. Range: [0.0, 1.0]. Default value: 0
max_bin	Maximum number of discrete bins to bucket continuous features. Used only if `tree_method` is set to `hist`. **Optional** Valid values: integer Default value: 256
max_delta_step	Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update. **Optional** Valid values: Integer. Range: [0,∞). Default value: 0
max_depth	Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when `grow_policy=depth-wise`. **Optional** Valid values: Integer. Range: [0,∞) Default value: 6
max_leaves	Maximum number of nodes to be added. Relevant only if `grow_policy` is set to `lossguide`. **Optional** Valid values: integer Default value: 0
min_child_weight	Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than `min_child_weight`, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is. **Optional** Valid values: Float. Range: [0,∞). Default value: 1

Parameter Name	Description
`normalize_type`	Type of normalization algorithm. **Optional** Valid values: Either *tree* or *forest*. Default value: *tree*
`nthread`	Number of parallel threads used to run *xgboost*. **Optional** Valid values: integer Default value: Maximum number of threads.
`objective`	Specifies the learning task and the corresponding learning objective. Examples: `reg:linear`, `reg:logistic`, `multi:softmax`. For a full list of valid inputs, refer to XGBoost Parameters. **Optional** Valid values: string Default value: `reg:linear`
`one_drop`	When this flag is enabled, at least one tree is always dropped during the dropout. **Optional** Valid values: 0 or 1 Default value: 0
`process_type`	The type of boosting process to run. **Optional** Valid values: String. Either `default` or `update`. Default value: `default`
`rate_drop`	The dropout rate that specifies the fraction of previous trees to drop during the dropout. **Optional** Valid values: Float. Range: [0.0, 1.0]. Default value: 0.0

Parameter Name	Description
`refresh_leaf`	This is a parameter of the 'refresh' updater plug-in. When set to `true` (1), tree leaves and tree node stats are updated. When set to `false`(0), only tree node stats are updated. **Optional** Valid values: 0/1 Default value: 1
`sample_type`	Type of sampling algorithm. **Optional** Valid values: Either `uniform` or `weighted`. Default value: `uniform`
`scale_pos_weight`	Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: `sum(negative cases) / sum(positive cases)`. **Optional** Valid values: float Default value: 1
`seed`	Random number seed. **Optional** Valid values: integer Default value: 0
`silent`	0 means print running messages, 1 means silent mode. Valid values: 0 or 1 **Optional** Default value: 0
`sketch_eps`	Used only for approximate greedy algorithm. This translates into O(1 / `sketch_eps`) number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy. **Optional** Valid values: Float, Range: [0, 1]. Default value: 0.03

Parameter Name	Description
`skip_drop`	Probability of skipping the dropout procedure during a boosting iteration. **Optional** Valid values: Float. Range: [0.0, 1.0]. Default value: 0.0
`subsample`	Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting. **Optional** Valid values: Float. Range: [0,1]. Default value: 1
`tree_method`	The tree construction algorithm used in XGBoost. **Optional** Valid values: One of `auto`, `exact`, `approx`, or `hist`. Default value: `auto`
`tweedie_variance_power`	Parameter that controls the variance of the Tweedie distribution. **Optional** Valid values: Float. Range: (1, 2). Default value: 1.5
`updater`	A comma-separated string that defines the sequence of tree updaters to run. This provides a modular way to construct and to modify the trees. For a full list of valid inputs, please refer to XGBoost Parameters. **Optional** Valid values: comma-separated string. Default value: `grow_colmaker`, prune

Tune an XGBoost Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

Note
Automatic model tuning for XGBoost 0.90 is only available from the SDKs, not from the Amazon SageMaker console.

For more information about model tuning, see .

Metrics Computed by the XGBoost Algorithm

The XGBoost algorithm computes the following nine metrics during training. When tuning the model, choose one of these metrics as the objective to evaluate the model.

Metric Name	Description	Optimization Direction
`validation:accuracy`	Classification rate, calculated as #(right)/#(all cases).	Maximize
`validation:auc`	Area under the curve.	Maximize
`validation:error`	Binary classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
`validation:f1`	Indicator of classification accuracy, calculated as the harmonic mean of precision and recall.	Maximize
`validation:logloss`	Negative log-likelihood.	Minimize
`validation:mae`	Mean absolute error.	Minimize
`validation:map`	Mean average precision.	Maximize
`validation:merror`	Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
`validation:mlogloss`	Negative log-likelihood for multiclass classification.	Minimize
`validation:mse`	Mean squared error.	Minimize
`validation:ndcg`	Normalized Discounted Cumulative Gain.	Maximize
`validation:rmse`	Root mean square error.	Minimize

Tunable XGBoost Hyperparameters

Tune the open-source XGBoost model with the following hyperparameters. The hyperparameters that have the greatest effect on XGBoost objective metrics are: `alpha`, `min_child_weight`, `subsample`, `eta`, and `num_round`.

Parameter Name	Parameter Type	Recommended Ranges
`alpha`	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
`colsample_bylevel`	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
`colsample_bynode`	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
`colsample_bytree`	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1

Parameter Name	Parameter Type	Recommended Ranges
eta	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 0.5
gamma	ContinuousParameterRanges	MinValue: 0, MaxValue: 5
lambda	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
max_delta_step	IntegerParameterRanges	[0, 10]
max_depth	IntegerParameterRanges	[0, 10]
min_child_weight	ContinuousParameterRanges	MinValue: 0, MaxValue: 120
num_round	IntegerParameterRanges	[1, 4000]
subsample	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1

XGBoost Previous Versions

This page contains links to the documentation for previous versions of Amazon SageMaker XGBoost.

Topics

- XGBoost Release 0.72 (p. 266)

XGBoost Release 0.72

This previous release of the Amazon SageMaker XGBoost algorithm is based on the 0.72 release. XGBoost (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models. XGBoost has done remarkably well in machine learning competitions because it robustly handles a variety of data types, relationships, and distributions, and the large number of hyperparameters that can be tweaked and tuned for improved fits. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

Customers should consider using the new release of XGBoost Algorithm (p. 255). They can use it as an Amazon SageMaker built-in algorithm or as a framework to run scripts in their local environments as they would typically, for example, do with a Tensorflow deep learning framework. The new implementation has a smaller memory footprint, better logging, improved hyperparameter validation, and an expanded set of metrics. The earlier implementation of XGBoost remains available to customers if they need to postpone migrating to the new version. But this previous implementation will remain tied to the 0.72 release of XGBoost.

Topics

- Input/Output Interface for the XGBoost Release 0.72 (p. 267)
- EC2 Instance Recommendation for the XGBoost Release 0.72 (p. 267)
- XGBoost Release 0.72 Sample Notebooks (p. 268)
- XGBoost Release 0.72 Hyperparameters (p. 268)
- Tune an XGBoost Release 0.72 Model (p. 274)

Input/Output Interface for the XGBoost Release 0.72

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The Amazon SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.
- For Inference ContentType, valid inputs are *text/libsvm* or (the default) *text/csv*.

> **Note**
> For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.
> For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format: <label> <index0>:<value0> <index1>:<value1> ... Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other Amazon SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count * the memory available in the `InstanceType`) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

To use a model trained with SageMaker XGBoost in open source XGBoost

- Use the following Python code:

```
import pickle as pkl
model = pkl.load(open(model_file_path, 'rb'))
# prediction with test data
pred = model.predict(dtest)
```

To differentiate the importance of labelled data points use Instance Weight Supports

- Amazon SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For *text/libsvm* input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1...`. For *text/csv* input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`).

EC2 Instance Recommendation for the XGBoost Release 0.72

Amazon SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M4) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

XGBoost Release 0.72 Sample Notebooks

For a sample notebook that shows how to use the latest version of Amazon SageMaker XGBoost as a built-in algorithm to train and host a regression model, see Regression with Amazon SageMaker XGBoost algorithm. To use the 0.72 version of XGBoost, you need to change the version in the sample code to 0.72. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

XGBoost Release 0.72 Hyperparameters

The following table contains the hyperparameters for the XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The Amazon SageMaker XGBoost algorithm is an implementation of the open-source XGBoost package. Currently Amazon SageMaker supports version 0.72. For more detail about hyperparameter configuration for this version of XGBoost, see XGBoost Parameters.

Parameter Name	Description
num_class	The number of classes. **Required** if `objective` is set to *multi:softmax* or *multi:softprob*. Valid values: integer
num_round	The number of rounds to run the training. **Required** Valid values: integer
alpha	L1 regularization term on weights. Increasing this value makes models more conservative. **Optional** Valid values: float Default value: 0
base_score	The initial prediction score of all instances, global bias. **Optional** Valid values: float Default value: 0.5
booster	Which booster to use. The `gbtree` and `dart` values use a tree-based model, while `gblinear` uses a linear function. **Optional** Valid values: String. One of `gbtree`, `gblinear`, or `dart`.

Parameter Name	Description
	Default value: `gbtree`
colsample_bylevel	Subsample ratio of columns for each split, in each level. **Optional** Valid values: Float. Range: [0,1]. Default value: 1
colsample_bytree	Subsample ratio of columns when constructing each tree. **Optional** Valid values: Float. Range: [0,1]. Default value: 1
csv_weights	When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights. **Optional** Valid values: 0 or 1 Default value: 0
early_stopping_rounds	The model trains until the validation score stops improving. Validation error needs to decrease at least every `early_stopping_rounds` to continue training. Amazon SageMaker hosting uses the best model for inference. **Optional** Valid values: integer Default value: -
eta	Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The `eta` parameter actually shrinks the feature weights to make the boosting process more conservative. **Optional** Valid values: Float. Range: [0,1]. Default value: 0.3

Parameter Name	Description
eval_metric	Evaluation metrics for validation data. A default metric is assigned according to the objective: • `rmse`: for regression • `error`: for classification • `map`: for ranking For a list of valid inputs, see XGBoost Parameters. **Optional** Valid values: string Default value: Default according to objective.
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is. **Optional** Valid values: Float. Range: [0,∞). Default value: 0
grow_policy	Controls the way that new nodes are added to the tree. Currently supported only if `tree_method` is set to `hist`. **Optional** Valid values: String. Either `depthwise` or `lossguide`. Default value: `depthwise`
lambda	L2 regularization term on weights. Increasing this value makes models more conservative. **Optional** Valid values: float Default value: 1
lambda_bias	L2 regularization term on bias. **Optional** Valid values: Float. Range: [0.0, 1.0]. Default value: 0

Parameter Name	Description
max_bin	Maximum number of discrete bins to bucket continuous features. Used only if `tree_method` is set to `hist`. **Optional** Valid values: integer Default value: 256
max_delta_step	Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update. **Optional** Valid values: Integer. Range: [0,∞). Default value: 0
max_depth	Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when `grow_policy=depth-wise`. **Optional** Valid values: Integer. Range: [0,∞) Default value: 6
max_leaves	Maximum number of nodes to be added. Relevant only if `grow_policy` is set to `lossguide`. **Optional** Valid values: integer Default value: 0
min_child_weight	Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than `min_child_weight`, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is. **Optional** Valid values: Float. Range: [0,∞). Default value: 1
normalize_type	Type of normalization algorithm. **Optional** Valid values: Either *tree* or *forest*. Default value: *tree*

Parameter Name	Description
nthread	Number of parallel threads used to run *xgboost*. **Optional** Valid values: integer Default value: Maximum number of threads.
objective	Specifies the learning task and the corresponding learning objective. Examples: `reg:linear`, `reg:logistic`, `multi:softmax`. For a full list of valid inputs, refer to XGBoost Parameters. **Optional** Valid values: string Default value: `reg:linear`
one_drop	When this flag is enabled, at least one tree is always dropped during the dropout. **Optional** Valid values: 0 or 1 Default value: 0
process_type	The type of boosting process to run. **Optional** Valid values: String. Either `default` or `update`. Default value: `default`
rate_drop	The dropout rate that specifies the fraction of previous trees to drop during the dropout. **Optional** Valid values: Float. Range: [0.0, 1.0]. Default value: 0.0
refresh_leaf	This is a parameter of the 'refresh' updater plug-in. When set to `true` (1), tree leaves and tree node stats are updated. When set to `false`(0), only tree node stats are updated. **Optional** Valid values: 0/1 Default value: 1

Parameter Name	Description
sample_type	Type of sampling algorithm. **Optional** Valid values: Either `uniform` or `weighted`. Default value: `uniform`
scale_pos_weight	Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: `sum(negative cases) / sum(positive cases)`. **Optional** Valid values: float Default value: 1
seed	Random number seed. **Optional** Valid values: integer Default value: 0
silent	0 means print running messages, 1 means silent mode. Valid values: 0 or 1 **Optional** Default value: 0
sketch_eps	Used only for approximate greedy algorithm. This translates into O(1 / `sketch_eps`) number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy. **Optional** Valid values: Float, Range: [0, 1]. Default value: 0.03
skip_drop	Probability of skipping the dropout procedure during a boosting iteration. **Optional** Valid values: Float. Range: [0.0, 1.0]. Default value: 0.0

Parameter Name	Description
subsample	Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting. **Optional** Valid values: Float. Range: [0,1]. Default value: 1
tree_method	The tree construction algorithm used in XGBoost. **Optional** Valid values: One of `auto`, `exact`, `approx`, or `hist`. Default value: `auto`
tweedie_variance_power	Parameter that controls the variance of the Tweedie distribution. **Optional** Valid values: Float. Range: (1, 2). Default value: 1.5
updater	A comma-separated string that defines the sequence of tree updaters to run. This provides a modular way to construct and to modify the trees. For a full list of valid inputs, please refer to XGBoost Parameters. **Optional** Valid values: comma-separated string. Default value: `grow_colmaker`, prune

Tune an XGBoost Release 0.72 Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 288).

Metrics Computed by the XGBoost Release 0.72 Algorithm

The XGBoost algorithm based on version 0.72 computes the following nine metrics during training. When tuning the model, choose one of these metrics as the objective to evaluate the model.

Metric Name	Description	Optimization Direction
validation:auc	Area under the curve.	Maximize

Metric Name	Description	Optimization Direction
`validation:error`	Binary classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
`validation:logloss`	Negative log-likelihood.	Minimize
`validation:mae`	Mean absolute error.	Minimize
`validation:map`	Mean average precision.	Maximize
`validation:merror`	Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
`validation:mlogloss`	Negative log-likelihood for multiclass classification.	Minimize
`validation:ndcg`	Normalized Discounted Cumulative Gain.	Maximize
`validation:rmse`	Root mean square error.	Minimize

Tunable XGBoost Release 0.72 Hyperparameters

Tune the XGBoost model with the following hyperparameters. The hyperparameters that have the greatest effect on XGBoost objective metrics are: `alpha`, `min_child_weight`, `subsample`, `eta`, and `num_round`.

Parameter Name	Parameter Type	Recommended Ranges
`alpha`	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
`colsample_bylevel`	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
`colsample_bytree`	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1
`eta`	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 0.5
`gamma`	ContinuousParameterRanges	MinValue: 0, MaxValue: 5
`lambda`	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
`max_delta_step`	IntegerParameterRanges	[0, 10]
`max_depth`	IntegerParameterRanges	[0, 10]
`min_child_weight`	ContinuousParameterRanges	MinValue: 0, MaxValue: 120
`num_round`	IntegerParameterRanges	[1, 4000]
`subsample`	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1

Train a Model

For an overview on training a model with Amazon SageMaker, see Train a Model with Amazon SageMaker (p. 4).

Amazon SageMaker provides features to monitor and manage the training and validation of machine learning models. For guidance on metrics available, incremental training, automatic model tuning, and the use of augmented manifest files to label training data, see the following topics.

- For guidance on metrics used to monitor and train models, see Monitor and Analyze Training Jobs Using Metrics (p. 276).
- For guidance on incremental training in Amazon SageMaker, see Incremental Training in Amazon SageMaker (p. 282).
- For guidance on using managed spot training in Amazon SageMaker, see Managed Spot Training in Amazon SageMaker (p. 287).
- For guidance on using training checkpoints in Amazon SageMaker, see Using Checkpoints in Amazon SageMaker (p. 288).
- For guidance on automatic model tuning, also known as hyperparameter tuning, see Automatic Model Tuning (p. 288).
- For guidance on using an augmented manifest file to label training data, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 308).

Monitor and Analyze Training Jobs Using Metrics

An Amazon SageMaker training job is an iterative process that teaches a model to make predictions by presenting examples from a training dataset. Typically, a training algorithm computes several metrics, such as training error and prediction accuracy. These metrics help diagnose whether the model is learning well and will generalize well for making predictions on unseen data. The training algorithm writes the values of these metrics to logs, which Amazon SageMaker monitors and sends to Amazon CloudWatch in real time. To analyze the performance of your training job, you can view graphs of these metrics in CloudWatch. When a training job has completed, you can also get a list of the metric values that it computes in its final iteration by calling the DescribeTrainingJob (p. 744) operation.

Topics
- Training Metrics Sample Notebooks (p. 276)
- Defining Training Metrics (p. 277)
- Monitoring Training Job Metrics (Console) (p. 279)
- Monitoring Training Job Metrics (Amazon SageMaker Console) (p. 279)
- Example: Viewing a Training and Validation Curve (p. 281)

Training Metrics Sample Notebooks

The following sample notebooks show how to view and plot training metrics:

- An Introduction to the Amazon SageMaker ObjectToVec Model for Sequence-to-sequence Embedding (object2vec_sentence_similarity.ipynb)

- Regression with the Amazon SageMaker XGBoost Algorithm (xgboost_abalone.ipynb)

For instructions how to create and access Jupyter notebook instances that you can use to run the examples in Amazon SageMaker, see Use Example Notebooks (p. 42). To see a list of all the Amazon SageMaker samples, after creating and opening a notebook instance, choose the **SageMaker Examples** tab. To access the example notebooks that show how to use training metrics, `object2vec_sentence_similarity.ipynb` and `xgboost_abalone.ipynb`., from the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Defining Training Metrics

Amazon SageMaker automatically parses the logs for metrics that built-in algorithms emit and sends those metrics to CloudWatch. If you want Amazon SageMaker to parse logs from a custom algorithm and send metrics that the algorithm emits to CloudWatch, you have to specify the metrics that you want Amazon SageMaker to send to CloudWatch when you configure the training job. You specify the name of the metrics that you want to send and the regular expressions that Amazon SageMaker uses to parse the logs that your algorithm emits to find those metrics.

You can specify the metrics that you want to track with the Amazon SageMaker console;, the Amazon SageMaker Python SDK (https://github.com/aws/sagemaker-python-sdk), or the low-level Amazon SageMaker API.

Topics

- Defining Regular Expressions for Metrics (p. 277)
- Defining Training Metrics (Low-level Amazon SageMaker API) (p. 278)
- Defining Training Metrics (Amazon SageMaker Python SDK) (p. 278)
- Define Training Metrics (Console) (p. 279)

Defining Regular Expressions for Metrics

To find a metric, Amazon SageMaker searches the logs that your algorithm emits and finds logs that match the regular expression that you specify for that metric. If you are using your own algorithm, do the following:

- Make sure that the algorithm writes the metrics that you want to capture to logs
- Define a regular expression that accurately searches the logs to capture the values of the metrics that you want to send to CloudWatch metrics.

For example, suppose your algorithm emits metrics for training error and validation error by writing logs similar to the following to `stdout` or `stderr`:

```
Train_error=0.138318;   Valid_error = 0.324557;
```

If you want to monitor both of those metrics in CloudWatch, your `AlgorithmSpecification` would look like the following:

```
"AlgorithmSpecification": {
        "TrainingImage": ContainerName,
        "TrainingInputMode": "File",
        "MetricDefinitions" : [
            {
```

```
            "Name": "train:error",
            "Regex": "Train_error=(.*?);"
        },
         {
            "Name": "validation:error",
            "Regex": "Valid_error=(.*?);"
        }

    ]}
```

In the regex for the `train:error` metric defined above, the first part of the regex finds the exact text "Train_error=", and the expression `(.*?);` captures zero or more of any character until the first semicolon character. In this expression, the parenthesis tell the regex to capture what is inside them, `.` means any character, `*` means zero or more, and `?` means capture only until the first instance of the `;` character.

Defining Training Metrics (Low-level Amazon SageMaker API)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions in the `MetricDefinitions` field of the AlgorithmSpecification (p. 863) input parameter that you pass to the CreateTrainingJob (p. 667) operation. For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your `AlgorithmSpecification` would look like the following:

```
"AlgorithmSpecification": {
        "TrainingImage": ContainerName,
        "TrainingInputMode": "File",
        "MetricDefinitions" : [
            {
            "Name": "train:error",
            "Regex": "Train_error=(.*?);"
        },
         {
            "Name": "validation:error",
            "Regex": "Valid_error=(.*?);"
        }

    ]}
```

For more information about defining and running a training job by using the low-level Amazon SageMaker API, see Create and Run a Training Job (AWS SDK for Python (Boto 3)) (p. 23).

Defining Training Metrics (Amazon SageMaker Python SDK)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions as the `metric_definitions` argument when you initialize an `Estimator` object. For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your `Estimator` initialization would look like the following:

```
estimator =
            Estimator(image_name=ImageName,
            role='SageMakerRole', train_instance_count=1,
            train_instance_type='ml.c4.xlarge',
            train_instance_type='ml.c4.xlarge',
            k=10,
            sagemaker_session=sagemaker_session,
            metric_definitions=[
                {'Name': 'train:error', 'Regex': 'Train_error=(.*?);'},
                {'Name': 'validation:error', 'Regex': 'Valid_error=(.*?);'
```

```
                ]
        )
```

For more information about training by using Amazon SageMaker Python SDK estimators, see https://
github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview.

Define Training Metrics (Console)

You can define metrics for a custom algorithm in the console when you create a training job by providing
the name and regular expression (regex) for **Metrics**.

For example, if you want to monitor both the `train:error` and `validation:error` metrics in
CloudWatch, your metric definitions would look like the following:

```
[
        {
        "Name": "train:error",
        "Regex": "Train_error=(.*?);"
    },
        {
        "Name": "validation:error",
        "Regex": "Valid_error=(.*?);"
    }

    ]}
```

Monitoring Training Job Metrics (Console)

You can monitor the metrics that a training job emits in real time in the CloudWatch console.

To monitor training job metrics (CloudWatch console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, then choose **/aws/sagemaker/TrainingJobs**.
3. Choose **TrainingJobName**.
4. On the **All metrics** tab, choose the names of the training metrics that you want to monitor.
5. On the **Graphed metrics** tab, configure the graph options. For more information about using
    CloudWatch graphs, see Graph Metrics in the *Amazon CloudWatch User Guide*.

Monitoring Training Job Metrics (Amazon SageMaker Console)

You can monitor the metrics that a training job emits in real time by using the Amazon SageMaker
console.

To monitor training job metrics (Amazon SageMaker console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Training jobs**, then choose the training job whose metrics you want to see.
3. Choose **TrainingJobName**.
4. In the **Monitor** section, you can review the graphs of instance utilization and algorithm metrics.

Example: Viewing a Training and Validation Curve

Typically, you split the data that you train your model on into training and validation datasets. You use the training set to train the model parameters that are used to make predictions on the training dataset. Then you test how well the model makes predictions by calculating predictions for the validation set. To analyze the performance of a training job, you commonly plot a training curve against a validation curve.

Viewing a graph that shows the accuracy for both the training and validation sets over time can help you to improve the performance of your model. For example, if training accuracy continues to increase over time, but, at some point, validation accuracy starts to decrease, you are likely overfitting your model. To address this, you can make adjustments to your model, such as increasing regularization.

For this example, you can use the **Image-classification-full-training** example that is in the **Example notebooks** section of your Amazon SageMaker notebook instance. If you don't have an Amazon SageMaker notebook instance, create one by following the instructions at Step 2: Create an Amazon SageMaker Notebook Instance (p. 17). If you prefer, you can follow along with the End-to-End Multiclass Image Classification Example in the example notebook on GitHub. You also need an Amazon S3 bucket to store the training data and for the model output. If you haven't created a bucket to use with Amazon SageMaker, create one by following the instructions at Step 1: Create an Amazon S3 Bucket (p. 17).

To view training and validation error curves

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

2. Choose **Notebooks**, and then choose **Notebook instances**.

3. Choose the notebook instance that you want to use, and then choose **Open**.

4. On the dashboard for your notebook instance, choose **SageMaker Examples**.

5. Expand the **Introduction to Amazon Algorithms** section, and then choose **Use** next to **Image-classification-full-training.ipynb**.

6. Choose **Create copy**. Amazon SageMaker creates an editable copy of the **Image-classification-full-training.ipynb** notebook in your notebook instance.

7. In the first code cell of the notebook, replace `<<bucket-name>>` with the name of your S3 bucket.

8. Run all of the cells in the notebook up to the **Deploy** section. You don't need to deploy an endpoint or get inference for this example.

9. After the training job starts, open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

10. Choose **Metrics**, then choose **/aws/sagemaker/TrainingJobs**.

11. Choose **TrainingJobName**.

12. On the **All metrics** tab, choose the **train:accuracy** and **validation:accuracy** metrics for the training job that you created in the notebook.

13. On the graph, choose an area that the metric's values to zoom in. You should see something like the following:
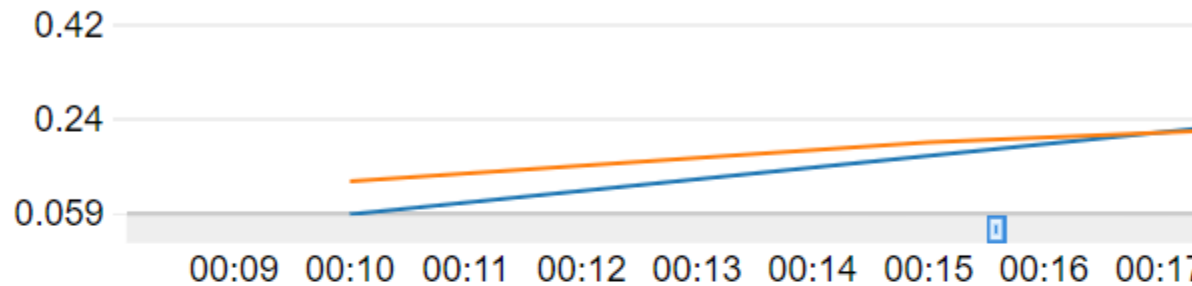
Incremental Training in Amazon SageMaker

Over time, you might find that a model generates inference that are not as good as they were in the past. With incremental training, you can use the artifacts from an existing model and use an expanded dataset to train a new model. Incremental training saves both time and resources.

Use incremental training to:

- Train a new model using an expanded dataset that contains an underlying pattern that was not accounted for in the previous training and which resulted in poor model performance.
- Use the model artifacts or a portion of the model artifacts from a popular publicly available model in a training job. You don't need to train a new model from scratch.

- Resume a training job that was stopped.
- Train several variants of a model, either with different hyperparameter settings or using different datasets.

For more information about training jobs, see Train a Model with Amazon SageMaker (p. 4).

You can train incrementally using the Amazon SageMaker console or the Amazon SageMaker Python SDK.

> **Important**
> Only two built-in algorithms currently support incremental training: Object Detection Algorithm (p. 199) and Image Classification Algorithm (p. 108).

Topics

Perform Incremental Training (Console)

To complete this procedure, you need:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see Common Parameters for Built-In Algorithms (p. 58).
- The URL of the S3 bucket where you've stored the model artifacts that you want to use in incremental training. To find the URL for the model artifacts, see the details page of the training job used to create the model. To find the details page, in the Amazon SageMaker console, choose **Inference**, choose **Models**, and then choose the model.

To restart a stopped training job, use the URL to the model artifacts that are stored in the details page as you would with a model or a completed training job.

To perform incremental training (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker.
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. The training job name must have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 56).
6. (Optional) For **Resource configuration**, either leave the default values or increase the resource consumption to reduce computation time.

   a. (Optional) For **Instance type**, choose the ML compute instance type that you want to use. In most cases, **ml.m4.xlarge** is sufficient.
   b. For **Instance count**, use the default, 1.
   c. (Optional) For **Additional volume per instance (GB)**, choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.

7. Provide information about the input data for the training dataset.

    a. For **Channel name**, either leave the default (`train`) or enter a more meaningful name for the training dataset, such as `expanded-training-dataset`.

    b. For **InputMode**, choose **File**. For incremental training, you need to use file input mode.

    c. For **S3 data distribution type**, choose **FullyReplicated**. This causes each ML compute instance to use a full replicate of the expanded dataset when training incrementally.

    d. If the expanded dataset is uncompressed, set the **Compression type** to **None**. If the expanded dataset is compressed using Gzip, set it to **Gzip**.

    e. (Optional) If you are using File input mode, leave **Content type** empty. For Pipe input mode, specify the appropriate MIME type. *Content type* is the multipurpose internet mail extension (MIME) type of the data.

    f. For **Record wrapper**, if the dataset is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO formatted file, choose **None**.

    g. For **S3 data type**, if the dataset us stored as a single file, choose **S3Prefix**. If the dataset is stored as several files in a folder, choose **Manifest**.

    h. For **S3 location**, provide the URL to the path where you stored the expanded dataset.

    i. Choose **Done**.

8. To use model artifacts in a training job, you need to add a new channel and provide the needed information about the model artifacts.

    a. For **Input data configuration**, choose **Add channel**.

    b. For **Channel name**, enter `model` to identify this channel as the source of the model artifacts.

    c. For **InputMode**, choose **File**. Model artifacts are stored as files.

    d. For **S3 data distribution type**, choose **FullyReplicated**. This indicates that each ML compute instance should use all of the model artifacts for training.

    e. For **Compression type**, choose **None** because we are using a model for the channel.

    f. Leave **Content type** empty. Content type is the multipurpose internet mail extension (MIME) type of the data. For model artifacts, we leave it empty.

    g. Set **Record wrapper** to **None** because model artifacts are not stored in RecordIO format.

    h. For **S3 data type**, if you are using a built-in algorithm or an algorithm that stores the model as a single file, choose **S3Prefix**. If you are using an algorithm that stores the model as several files, choose **Manifest**.

    i. For **S3 location**, provide the URL to the path where you stored the model artifacts. Typically, the model is stored with the name `model.tar.gz`. To find the URL for the model artifacts, in the navigation pane, choose **Inference**, then choose **Models**. From the list of models, choose a model to display its details page. The URL for the model artifacts is listed under **Primary container** .

    j. Choose **Done**.

9. For **Output data configuration**, provide the following information:

    a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.

    b. (Optional) For **Encryption key**, you can add your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. Provide the key ID or its Amazon Resource Number (ARN). For more information, see KMS-Managed Encryption Keys.

10. (Optional) For **Tags**, add one or more tags to the training job. A *tag* is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with `Project` as a key and a value referring to a project that is related to the training job, such as `Home value forecasts`.

11. Choose **Create training job**. Amazon SageMaker creates and runs training job.

After the training job has completed, the newly trained model artifacts are stored under the **S3 output path** that you provided in the **Output data configuration** field. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 26).

Perform Incremental Training (API)

This example shows how to use Amazon SageMaker APIs to train a model using the Amazon SageMaker image classification algorithm and the Caltech 256 Image Dataset, then train a new model using the first one. It uses Amazon S3 for input and output sources. Please see the incremental training sample notebook for more details on using incremental training.

> **Note**
> In this example we used the original datasets in the incremental training, however you can use different datasets, such as ones that contain newly added samples. Upload the new datasets to S3 and make adjustments to the `data_channels` variable used to train the new model.

Get an AWS Identity and Access Management (IAM) role that grants required permissions and initialize environment variables:

```
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)

sess = sagemaker.Session()

bucket=sess.default_bucket()
print(bucket)
prefix = 'ic-incr-training'
```

Get the training image for the image classification algorithm:

```
from sagemaker.amazon.amazon_estimator import get_image_uri

training_image = get_image_uri(sess.boto_region_name, 'image-classification',
 repo_version="latest")
#Display the training image
print (training_image)
```

Download the training and validation datasets, then upload them to Amazon Simple Storage Service (Amazon S3):

```
import os
import urllib.request
import boto3

# Define a download function
def download(url):
    filename = url.split("/")[-1]
    if not os.path.exists(filename):
        urllib.request.urlretrieve(url, filename)

# Download the caltech-256 training and validation datasets
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-train.rec')
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-val.rec')

# Create four channels: train, validation, train_lst, and validation_lst
s3train = 's3://{}/{}/train/'.format(bucket, prefix)
```

```
s3validation = 's3://{}/{}/validation/'.format(bucket, prefix)

# Upload the first files to the train and validation channels
!aws s3 cp caltech-256-60-train.rec $s3train --quiet
!aws s3 cp caltech-256-60-val.rec $s3validation --quiet
```

Define the training hyperparameters:

```
# Define hyperparameters for the estimator
hyperparams = { "num_layers": "18",
                "resize": "32",
                "num_training_samples": "50000",
                "num_classes": "10",
                "image_shape": "3,28,28",
                "mini_batch_size": "128",
                "epochs": "3",
                "learning_rate": "0.1",
                "lr_scheduler_step": "2,3",
                "lr_scheduler_factor": "0.1",
                "augmentation_type": "crop_color",
                "optimizer": "sgd",
                "momentum": "0.9",
                "weight_decay": "0.0001",
                "beta_1": "0.9",
                "beta_2": "0.999",
                "gamma": "0.9",
                "eps": "1e-8",
                "top_k": "5",
                "checkpoint_frequency": "1",
                "use_pretrained_model": "0",
                "model_prefix": "" }
```

Create an estimator object and train the first model using the training and validation datasets:

```
# Fit the base estimator
s3_output_location = 's3://{}/{}/output'.format(bucket, prefix)
ic = sagemaker.estimator.Estimator(training_image,
                                   role,
                                   train_instance_count=1,
                                   train_instance_type='ml.p2.xlarge',
                                   train_volume_size=50,
                                   train_max_run=360000,
                                   input_mode='File',
                                   output_path=s3_output_location,
                                   sagemaker_session=sess,
                                   hyperparameters=hyperparams)

train_data = sagemaker.session.s3_input(s3train, distribution='FullyReplicated',
                                            content_type='application/x-recordio',
 s3_data_type='S3Prefix')
validation_data = sagemaker.session.s3_input(s3validation, distribution='FullyReplicated',
                                            content_type='application/x-recordio',
 s3_data_type='S3Prefix')

data_channels = {'train': train_data, 'validation': validation_data}

ic.fit(inputs=data_channels, logs=True)
```

To use the model to incrementally train another model, create a new estimator object and use the model artifacts (ic.model_data, in this example) for the model_uri input argument:

```
# Given the base estimator, create a new one for incremental training
```

```
incr_ic = sagemaker.estimator.Estimator(training_image,
                                         role,
                                         train_instance_count=1,
                                         train_instance_type='ml.p2.xlarge',
                                         train_volume_size=50,
                                         train_max_run=360000,
                                         input_mode='File',
                                         output_path=s3_output_location,
                                         sagemaker_session=sess,
                                         hyperparameters=hyperparams,
                                         model_uri=ic.model_data) # This parameter will
 ingest the previous job's model as a new channel
incr_ic.fit(inputs=data_channels, logs=True)
```

After the training job has completed, the newly trained model artifacts are stored under the `S3 output path` that you provided in `Output_path`. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 26).

Managed Spot Training in Amazon SageMaker

Amazon SageMaker makes it easy to train machine learning models using managed Amazon EC2 Spot instances. Managed spot training can optimize the cost of training models up to 90% over on-demand instances. Amazon SageMaker manages the Spot interruptions on your behalf.

Managed Spot Training uses Amazon EC2 Spot instance to run training jobs instead of on-demand instances. You can specify which training jobs use spot instances and a stopping condition that specifies how long Amazon SageMaker waits for a job to run using Amazon EC2 Spot instances. Metrics and logs generated during training runs are available in CloudWatch.

Spot instances can be interrupted, causing jobs to take longer to start or finish. You can configure your managed spot training job to use checkpoints. Amazon SageMaker copies checkpoint data from a local path to Amazon S3. When the job is restarted, Amazon SageMaker copies the data from Amazon S3 back into the local path. The training can then resume from the last checkpoint instead of restarting. For more information about checkpointing, see Using Checkpoints in Amazon SageMaker (p. 288).

> **Note**
> Unless your training job will complete quickly, we recommend you use checkpointing with managed spot training. SageMaker built-in algorithms and marketplace algorithms that do not checkpoint are currently limited to a `MaxWaitTimeInSeconds` of 3600 seconds (60 minutes).

Topics
- Using Managed Spot Training (p. 287)
- Managed Spot Training Lifecycle (p. 288)

Using Managed Spot Training

To use managed spot training, create a training job. Set `EnableManagedSpotTraining` to `True` and specify the `MaxWaitTimeInSecods`. `MaxWaitTimeInSeconds` must be larger than `MaxRuntimeInSeconds`. For more information about creating a training job, see CreateTrainingJob (p. 667).

You can calculate the savings from using managed spot training using the formula `(1 - BillableTimeInSeconds / TrainingTimeInSeconds) * 100`. For example, if `BillableTimeInSeconds` is 100 and `TrainingTimeInSeconds` is 500, the savings is 80%.

Managed Spot Training Lifecycle

You can monitor a training job using `TrainingJobStatus` and `SecondaryStatus` returned by DescribeTrainingJob (p. 744). The list below shows how `TrainingJobStatus` and `SecondaryStatus` values change depending on the training scenario:

- **Spot instances acquired with no interruption during training**
  1. `InProgress: Starting→ Downloading → Training → Uploading`
- **Spot instances interrupted once. Later, enough spot instances were acquired to finish the training job.**
  1. `InProgress: Starting → Downloading → Training → Interrupted → Starting → Downloading → Training → Uploading`
- **Spot instances interrupted twice and `MaxWaitTimeInSeconds` exceeded.**
  1. `InProgress: Starting → Downloading → Training → Interrupted → Starting → Downloading → Training → Interrupted → Downloading → Training`
  2. `Stopping: Stopping`
  3. `Stopped: MaxWaitTimeExceeded`
- **Spot instances were never launched.**
  1. `InProgress: Starting`
  2. `Stopping: Stopping`
  3. `Stopped: MaxWaitTimeExceeded`

Using Checkpoints in Amazon SageMaker

A checkpoint is a snapshot of the state of the model. They can be used with Managed Spot Training. If a training job is interrupted, a snapshot can be used to resume from a previously saved point. This can save training time.

Snapshots are saved to an Amazon S3 location you specify. You can configure the local path to use for snapshots or use the default. When a training job is interrupted, Amazon SageMaker copies the training data to Amazon S3. When the training job is restarted, the checkpoint data is copied to the local path. It can be used to resume at the checkpoint.

To enable checkpoints, provide an Amazon S3 location. You can optionally provide a local path and choose to use a shared folder. The default local path is `/opt/ml/checkpoints/`. For more information, see CreateTrainingJob (p. 667)

Automatic Model Tuning

Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose.

For example, suppose that you want to solve a *binary classification* problem on a marketing dataset. Your goal is to maximize the *area under the curve (auc)* metric of the algorithm by training an XGBoost Algorithm (p. 255) model. You don't know which values of the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters to use to train the best model. To find the best values for these hyperparameters, you can specify ranges of values that Amazon SageMaker hyperparameter tuning

searches to find the combination of values that results in the training job that performs the best as measured by the objective metric that you chose. Hyperparameter tuning launches training jobs that use hyperparameter values in the ranges that you specified, and returns the training job with highest auc.

You can use Amazon SageMaker automatic model tuning with built-in algorithms, custom algorithms, and Amazon SageMaker pre-built containers for machine learning frameworks.

Before you start using hyperparameter tuning, you should have a well-defined machine learning problem, including the following:

- A dataset
- An understanding of the type of algorithm you need to train
- A clear understanding of how you measure success

You should also prepare your dataset and algorithm so that they work in Amazon SageMaker and successfully run a training job at least once. For information about setting up and running a training job, see Get Started (p. 16).

Topics

How Hyperparameter Tuning Works

Random Search

In a random search, hyperparameter tuning chooses a random combination of values from within the ranges that you specify for hyperparameters for each training job it launches. Because the choice of hyperparameter values doesn't depend on the results of previous training jobs, you can run the maximum number of concurrent training jobs without affecting the performance of the search.

For an example notebook that uses random search, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb.

Bayesian Search

Bayesian search treats hyperparameter tuning like a *[regression]* problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. To solve a regression problem, hyperparameter tuning makes guesses about which hyperparameter combinations are likely to get the best results, and runs training jobs to test these values. After testing the first set of hyperparameter values, hyperparameter tuning uses regression to choose the next set of hyperparameter values to test.

Hyperparameter tuning uses an Amazon SageMaker implementation of Bayesian optimization.

When choosing the best hyperparameters for the next training job, hyperparameter tuning considers everything that it knows about this problem so far. Sometimes it chooses a combination of hyperparameter values close to the combination that resulted in the best previous training job to incrementally improve performance. This allows hyperparameter tuning to exploit the best known results. Other times, it chooses a set of hyperparameter values far removed from those it has tried. This allows it to explore the range of hyperparameter values to try to find new areas that are not well understood. The explore/exploit trade-off is common in many machine learning problems.

For more information about Bayesian optimization, see the following:

Basic Topics on Bayesian Optimization

- A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning
- Practical Bayesian Optimization of Machine Learning Algorithms
- Taking the Human Out of the Loop: A Review of Bayesian Optimization

Speeding up Bayesian Optimization

- Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization
- Google Vizier: A Service for Black-Box Optimization
- Learning Curve Prediction with Bayesian Neural Networks
- Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves

Advanced Modeling and Transfer Learning

- Scalable Hyperparameter Transfer Learning
- Bayesian Optimization with Tree-structured Dependencies
- Bayesian Optimization with Robust Bayesian Neural Networks
- Scalable Bayesian Optimization Using Deep Neural Networks
- Input Warping for Bayesian Optimization of Non-stationary Functions

Note
Hyperparameter tuning might not improve your model. It is an advanced tool for building machine solutions, and, as such, should be considered part of the scientific development process.
When you build complex machine learning systems like deep learning neural networks, exploring all of the possible combinations is impractical. Hyperparameter tuning can accelerate your productivity by trying many variations of a model, focusing on the most promising combinations of hyperparameter values within the ranges that you specify. To get good results, you need to choose the right ranges to explore. Because the algorithm itself is stochastic, it's possible that the hyperparameter tuning model will fail to converge on the best answer, even if the best possible combination of values is within the ranges that you choose.

Define Metrics

Note
When you use one of the Amazon SageMaker built-in algorithms, you don't need to define metrics. Built-in algorithms automatically send metrics to hyperparameter tuning. You do need to choose one of the metrics that the built-in algorithm emits as the objective metric for the tuning job. For a list of metrics that a built-in algorithm emits, see the *Metrics* table for the algorithm in Use Amazon SageMaker Built-in Algorithms (p. 56).

To optimize hyperparameters for a machine learning model, a tuning job evaluates the training jobs it launches by using a metric that the training algorithm writes to logs. Amazon SageMaker hyperparameter tuning parses your algorithm's `stdout` and `stderr` streams to find algorithm metrics, such as loss or validation-accuracy, that show how well the model is performing on the dataset

> **Note**
> These are the same metrics that Amazon SageMaker sends to CloudWatch Logs. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 466).

If you use your own algorithm for hyperparameter tuning, make sure that your algorithm emits at least one metric by writing evaluation data to `stderr` or `stdout`.

> **Note**
> Hyperparameter tuning sends an additional hyperparameter, `_tuning_objective_metric` to the training algorithm. This hyperparameter specifies the objective metric being used for the hyperparameter tuning job, so that your algorithm can use that information during training.

You can define up to 20 metrics for your tuning job to monitor. You choose one of those metrics to be the objective metric, which hyperparameter tuning uses to evaluate the training jobs. The hyperparameter tuning job returns the training job that returned the best value for the objective metric as the best training job.

You define metrics for a tuning job by specifying a name and a regular expression for each metric that your tuning job monitors. Design the regular expressions to capture the values of metrics that your algorithm emits. You pass these metrics to the CreateHyperParameterTuningJob (p. 638) operation in the `TrainingJobDefinition` parameter as the `MetricDefinitions` field of the `AlgorithmSpecification` field.

The following example defines 4 metrics:

```
=[
    {
        "Name": "loss",
        "Regex": "Loss = (.*?);",
    },
    {
        "Name": "ganloss",
        "Regex": "GAN_loss=(.*?);",
    },
    {
        "Name": "discloss",
        "Regex": "disc_train_loss=(.*?);",
    },
    {
        "Name": "disc-combined",
        "Regex": "disc-combined=(.*?);",
    },
]
```

The following is an example of the log that the algorithm writes:

```
GAN_loss=0.138318;  Scaled_reg=2.654134; disc:[-0.017371,0.102429] real 93.3% gen 0.0%
 disc-combined=0.000000; disc_train_loss=1.374587;  Loss = 16.020744;  Iteration 0 took
 0.704s;  Elapsed=0s
```

Use the regular expression (regex) to match the algorithm's log output and capture the numeric values of metrics. For example, in the regex for the `loss` metric defined above, the first part of the regex finds the exact text "Loss = ", and the expression `(.*?);` captures zero or more of any character until the first semicolon character. In this expression, the parenthesis tell the regex to capture what is inside them, `.` means any character, `*` means zero or more, and `?` means capture only until the first instance of the `;` character.

Choose one of the metrics that you define as the objective metric for the tuning job. If you are using the API, specify the value of the `name` key in the `HyperParameterTuningJobObjective` field of the `HyperParameterTuningJobConfig` parameter that you send to the CreateHyperParameterTuningJob (p. 638) operation.

Define Hyperparameter Ranges

Hyperparameter tuning finds the best hyperparameter values for your model by searching over ranges of hyperparameters. You specify the hyperparameters and range of values over which to search by defining hyperparameter ranges for your tuning job. Choosing hyperparameters and ranges significantly affects the performance of your tuning job. For guidance on choosing hyperparameters and ranges, see Best Practices for Hyperparameter Tuning (p. 308).

To define hyperparameter ranges by using the low-level API, you specify the names of hyperparameters and ranges of values in the `ParameterRanges` field of the `HyperParameterTuningJobConfig` parameter that you pass to the CreateHyperParameterTuningJob (p. 638) operation. The `ParameterRanges` field has three subfields, one for each of the categorical, integer, and continuous hyperparameter ranges. You can define up to 20 hyperparameters to search over. Each value of a categorical hyperparameter range counts as a hyperparameter against the limit. Hyperparameter ranges have the following structure:

```
"ParameterRanges": {
    "CategoricalParameterRanges": [
      {
        "Name": "tree_method",
        "Values": ["auto", "exact", "approx", "hist"]
      }
    ],
    "ContinuousParameterRanges": [
      {
        "Name": "eta",
        "MaxValue" : "0.5",
        "MinValue": "0",
        "ScalingType": "Auto"
      }
    ],
    "IntegerParameterRanges": [
      {
        "Name": "max_depth",
        "MaxValue": "10",
        "MinValue": "1",
        "ScalingType": "Auto"
      }
    ]
  }
```

Hyperparameter Scaling

For integer and continuous hyperparameter ranges, you can choose the scale you want hyperparameter tuning to use to search the range of values by specifying a value for the `ScalingType` field of the hyperparameter range. You can choose from the following scaling types:

Auto

Amazon SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.

Linear

Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale. Typically, you choose this if the range of all values from the lowest to the highest is relatively small

(within one order of magnitude), because uniformly searching values from the range will give you a reasonable exploration of the entire range.

Logarithmic

Hyperparemeter tuning searches the values in the hyperparameter range by using a logarithmic scale.

Logarithmic scaling works only for ranges that have only values greater than 0.

Choose logarithmic scaling when you are searching a range that spans several orders of magnitude. For example, if you are tuning a Tune a Linear Learner Model (p. 162) model, and you specify a range of values between .0001 and 1.0 for the `learning_rate` hyperparameter, searching uniformly on a logarithmic scale gives you a better sample of the entire range than searching on a linear scale would, because searching on a linear scale would, on average, devote 90 percent of your training budget to only the values between .1 and 1.0, leaving only 10 percent of your training budget for the values between .0001 and .1.

ReverseLogarithmic

Hyperparemeter tuning searches the values in the hyperparameter range by using a reverse logarithmic scale. reverse logarithmic scaling is supported only for continuous hyperparameter ranges. It is not supported for integer hyperparameter ranges.

Reverse logarithmic scaling works only for ranges that are entirely within the range 0<=x<1.0.

Choose reverse logarithmic scaling when you are searching a range that is highly sensitive to small changes that are very close to 1.

For an example notebook that uses hyperparameter scaling, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb.

Example: Hyperparameter Tuning Job

This example shows how to create a new notebook for configuring and launching a hyperparameter tuning job. The tuning job uses the XGBoost Algorithm (p. 255) to train a model to predict whether a customer will enroll for a term deposit at a bank after being contacted by phone.

You use the low-level AWS SDK for Python (Boto) to configure and launch the hyperparameter tuning job, and the AWS Management Console to monitor the status of hyperparameter training jobs. You can also use the Amazon SageMaker high-level Amazon SageMaker Python SDK to configure, run, monitor, and analyze hyperparameter tuning jobs. For more information, see https://github.com/aws/sagemaker-python-sdk.

Prerequisites

To run the code in this example, you need

- An AWS account and an administrator user (p. 14)
- An Amazon S3 bucket for storing your training dataset and the model artifacts created during training (p. 17)
- A running Amazon SageMaker notebook instance (p. 17)

Topics
- Create a Notebook (p. 294)

Create a Notebook

Create a Jupyter notebook that contains a preinstalled environment with the default Anaconda installation and Python3.

To create a Jupyter notebook

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/
    .
2. Open a running notebook instance, by choosing **Open** next to its name. The Jupyter notebook server page appears:
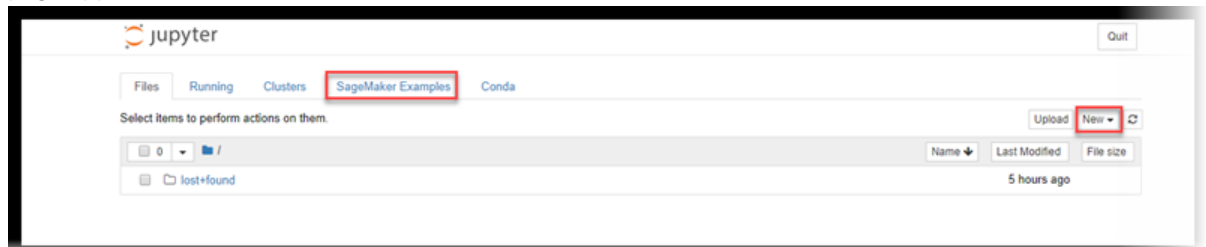
3. To create a notebook, choose **Files**, **New**, and **conda_python3**. .
4. Name the notebook.

Next Step

Get the Amazon Sagemaker Boto 3 Client

Import libraries and get a Boto3 client, which you use to call the hyperparameter tuning APIs.

In the new Jupyter notebook, type the following code:

```
import sagemaker
import boto3
from sagemaker.predictor import csv_serializer     # Converts strings for HTTP POST requests
 on inference

import numpy as np                                  # For performing matrix operations and
 numerical processing
import pandas as pd                                 # For manipulating tabular data
from time import gmtime, strftime
import os
```

```
region = boto3.Session().region_name
smclient = boto3.Session().client('sagemaker')
```

Next Step

Get the Amazon SageMaker Execution Role

Get the execution role for the notebook instance. This is the IAM role that you created when you created your notebook instance. You pass the role to the tuning job.

```
from sagemaker import get_execution_role

role = get_execution_role()
print(role)
```

Next Step

Specify a Bucket and Data Output Location

Specify the name of the Amazon S3 bucket where you want to store the output of the training jobs that the tuning job launches. The name of the bucket must contain **sagemaker**, and be globally unique. The bucket must be in the same AWS Region as the notebook instance that you use for this example. You can use the bucket that you created when you set up Amazon SageMaker, or you can create a new bucket. For information, see Step 1: Create an Amazon S3 Bucket (p. 17).

> **Note**
> The name of the bucket doesn't need to contain **sagemaker** if the role that you use to run the hyperparameter tuning job has a policy that gives the SageMaker service principle `S3FullAccess` permission.

`prefix` is the path within the bucket where Amazon SageMaker stores the output from training jobs.

```
bucket = 'sagemaker-MyBucket'                           # Replace with the name of your
 S3 bucket
prefix = 'sagemaker/DEMO-automatic-model-tuning-xgboost-dm'
```

Next Step

Download, Prepare, and Upload Training Data

For this example, you use a training dataset of information about bank customers that includes the customer's job, marital status, and how they were contacted during the bank's direct marketing campaign. To use a dataset for a hyperparameter tuning job, you download it, transform the data, and then upload it to an Amazon S3 bucket.

For more information about the dataset and the data transformation that the example performs, see the *hpo_xgboost_direct_marketing_sagemaker_APIs* notebook in the **Hyperparameter Tuning** section of the **SageMaker Examples** tab in your notebook instance.

Download and Explore the Training Dataset

To download and explore the dataset, run the following code in your notebook:

```
!wget -N https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-
additional.zip
!unzip -o bank-additional.zip
data = pd.read_csv('./bank-additional/bank-additional-full.csv', sep=';')
pd.set_option('display.max_columns', 500)    # Make sure we can see all of the columns
pd.set_option('display.max_rows', 5)         # Keep the output on one page
data
```

Prepare and Upload Data

Before creating the hyperparameter tuning job, prepare the data and upload it to an S3 bucket where the hyperparameter tuning job can access it.

Run the following code in your notebook:

```
data['no_previous_contact'] = np.where(data['pdays'] == 999, 1, 0)
        # Indicator variable to capture when pdays takes a value of 999
data['not_working'] = np.where(np.in1d(data['job'], ['student', 'retired', 'unemployed']),
 1, 0)   # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)
        # Convert categorical variables to sets of indicators
model_data
model_data = model_data.drop(['duration', 'emp.var.rate', 'cons.price.idx',
 'cons.conf.idx', 'euribor3m', 'nr.employed'], axis=1)

train_data, validation_data, test_data = np.split(model_data.sample(frac=1,
 random_state=1729), [int(0.7 * len(model_data)), int(0.9*len(model_data))])

pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)],
 axis=1).to_csv('train.csv', index=False, header=False)
pd.concat([validation_data['y_yes'], validation_data.drop(['y_no', 'y_yes'], axis=1)],
 axis=1).to_csv('validation.csv', index=False, header=False)
pd.concat([test_data['y_yes'], test_data.drop(['y_no', 'y_yes'], axis=1)],
 axis=1).to_csv('test.csv', index=False, header=False)

boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/
train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/
validation.csv')).upload_file('validation.csv')
```

Next Step

Configure and Launch a Hyperparameter Tuning Job

To configure and launch a hyperparameter tuning job, complete the following steps.

Topics

Specify the Hyperparameter Tuning Job Settings

To specify settings for the hyperparameter tuning job, you define a JSON object. You pass the object as the value of the `HyperParameterTuningJobConfig` parameter to CreateHyperParameterTuningJob (p. 638) when you create the tuning job.

In this JSON object, you specify:

- The ranges of hyperparameters that you want to tune. For more information, see Define Hyperparameter Ranges (p. 292)
- The limits of the resource that the hyperparameter tuning job can consume.
- The objective metric for the hyperparameter tuning job. An *objective metric* is the metric that the hyperparameter tuning job uses to evaluate the training job that it launches.

    Note
    To use your own algorithm for hyperparameter tuning, you need to define metrics for your algorithm. For information,see Define Metrics (p. 290).

The hyperparameter tuning job defines ranges for the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters of the XGBoost Algorithm (p. 255) built-in algorithm. The objective metric for the hyperparameter tuning job maximizes the `validation:auc` metric that the algorithm sends to CloudWatch Logs.

```
tuning_job_config = {
    "ParameterRanges": {
      "CategoricalParameterRanges": [],
      "ContinuousParameterRanges": [
        {
          "MaxValue": "1",
          "MinValue": "0",
          "Name": "eta"
        },
        {
          "MaxValue": "2",
          "MinValue": "0",
          "Name": "alpha"
        },
        {
          "MaxValue": "10",
          "MinValue": "1",
          "Name": "min_child_weight"
        }
      ],
      "IntegerParameterRanges": [
        {
          "MaxValue": "10",
          "MinValue": "1",
          "Name": "max_depth"
        }
      ]
    },
    "ResourceLimits": {
      "MaxNumberOfTrainingJobs": 20,
      "MaxParallelTrainingJobs": 3
    },
    "Strategy": "Bayesian",
    "HyperParameterTuningJobObjective": {
      "MetricName": "validation:auc",
      "Type": "Maximize"
    }
  }
```

Configure the Training Jobs

To configure the training jobs that the tuning job launches, define a JSON object that you pass as the value of the `TrainingJobDefinition` parameter of the CreateHyperParameterTuningJob (p. 638) call.

In this JSON object, you specify:

- Optional—Metrics that the training jobs emit.

    Note
    Define metrics only when you use a custom training algorithm. Because this example uses a built-in algorithm, you don't specify metrics. For information about defining metrics, see Define Metrics (p. 290).

- The container image that specifies the training algorithm.
- The input configuration for your training and test data.
- The storage location for the algorithm's output. Specify the S3 bucket where you want to store the output of the training jobs.
- The values of algorithm hyperparameters that are not tuned in the tuning job.
- The type of instance to use for the training jobs.
- The stopping condition for the training jobs. This is the maximum duration for each training job.

In this example, we set static values for the `eval_metric`, `num_round`, `objective`, `rate_drop`, and `tweedie_variance_power` parameters of the XGBoost Algorithm (p. 255) built-in algorithm.

```
from sagemaker.amazon.amazon_estimator import get_image_uri
training_image = get_image_uri(boto3.Session().region_name, 'xgboost')

s3_input_train = 's3://{}/{}/train'.format(bucket, prefix)
s3_input_validation ='s3://{}/{}/validation/'.format(bucket, prefix)

training_job_definition = {
    "AlgorithmSpecification": {
      "TrainingImage": training_image,
      "TrainingInputMode": "File"
    },
    "InputDataConfig": [
      {
        "ChannelName": "train",
        "CompressionType": "None",
        "ContentType": "csv",
        "DataSource": {
          "S3DataSource": {
            "S3DataDistributionType": "FullyReplicated",
            "S3DataType": "S3Prefix",
            "S3Uri": s3_input_train
          }
        }
      },
      {
        "ChannelName": "validation",
        "CompressionType": "None",
        "ContentType": "csv",
        "DataSource": {
          "S3DataSource": {
            "S3DataDistributionType": "FullyReplicated",
            "S3DataType": "S3Prefix",
            "S3Uri": s3_input_validation
          }
```

```
      }
    }
  ],
  "OutputDataConfig": {
    "S3OutputPath": "s3://{}/{}/output".format(bucket,prefix)
  },
  "ResourceConfig": {
    "InstanceCount": 2,
    "InstanceType": "ml.c4.2xlarge",
    "VolumeSizeInGB": 10
  },
  "RoleArn": role,
  "StaticHyperParameters": {
    "eval_metric": "auc",
    "num_round": "100",
    "objective": "binary:logistic",
    "rate_drop": "0.3",
    "tweedie_variance_power": "1.4"
  },
  "StoppingCondition": {
    "MaxRuntimeInSeconds": 43200
  }
}
```

Name and Launch the Hyperparameter Tuning Job

Now you can provide a name for the hyperparameter tuning job and then launch it by calling the CreateHyperParameterTuningJob (p. 638) API. Pass `tuning_job_config`, and `training_job_definition` that you created in previous steps as the values of the parameters.

```
tuning_job_name = "MyTuningJob"
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName = tuning_job_name,
                                           HyperParameterTuningJobConfig =
 tuning_job_config,
                                           TrainingJobDefinition = training_job_definition)
```

Next Step

Monitor the Progress of a Hyperparameter Tuning Job

To monitor the progress of a hyperparameter tuning job and the training jobs that it launches, use the Amazon SageMaker console.

Topics

- View the Status of the Hyperparameter Tuning Job (p. 299)
- View the Status of the Training Jobs (p. 300)
- View the Best Training Job (p. 301)

View the Status of the Hyperparameter Tuning Job

To view the status of the hyperparameter tuning job

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/
   .

2. Choose **Hyperparameter tuning jobs**.
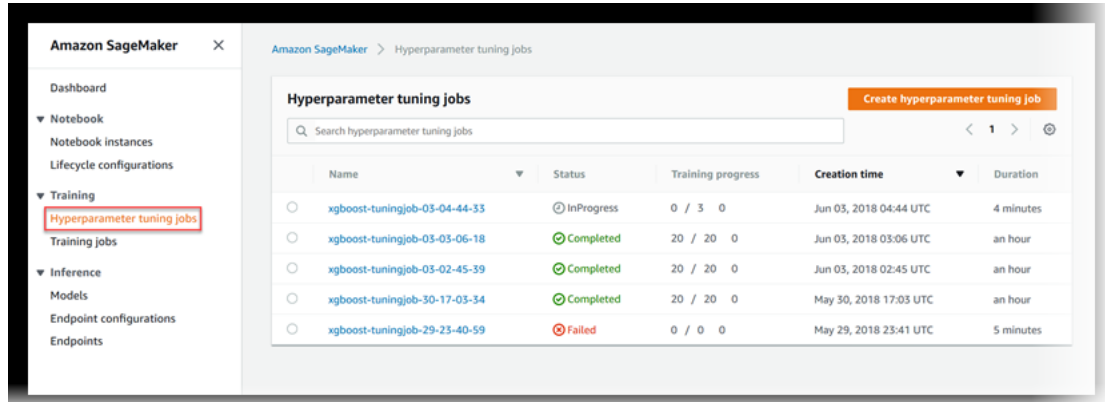
3. In the list of hyperparameter tuning jobs, check the status of the hyperparameter tuning job you launched. A tuning job can be:

- `Completed`—The hyperparameter tuning job successfully completed.
- `InProgress`—The hyperparameter tuning job is in progress. One or more training jobs are still running.
- `Failed`—The hyperparameter tuning job failed.
- `Stopped`—The hyperparameter tuning job was manually stopped before it completed. All training jobs that the hyperparameter tuning job launched are stopped.
- `Stopping`—The hyperparameter tuning job is in the process of stopping.

View the Status of the Training Jobs

To view the status of the training jobs that the hyperparameter tuning job launched

1. In the list of hyperparameter tuning jobs, choose the job that you launched.
2. Choose **Training jobs**.
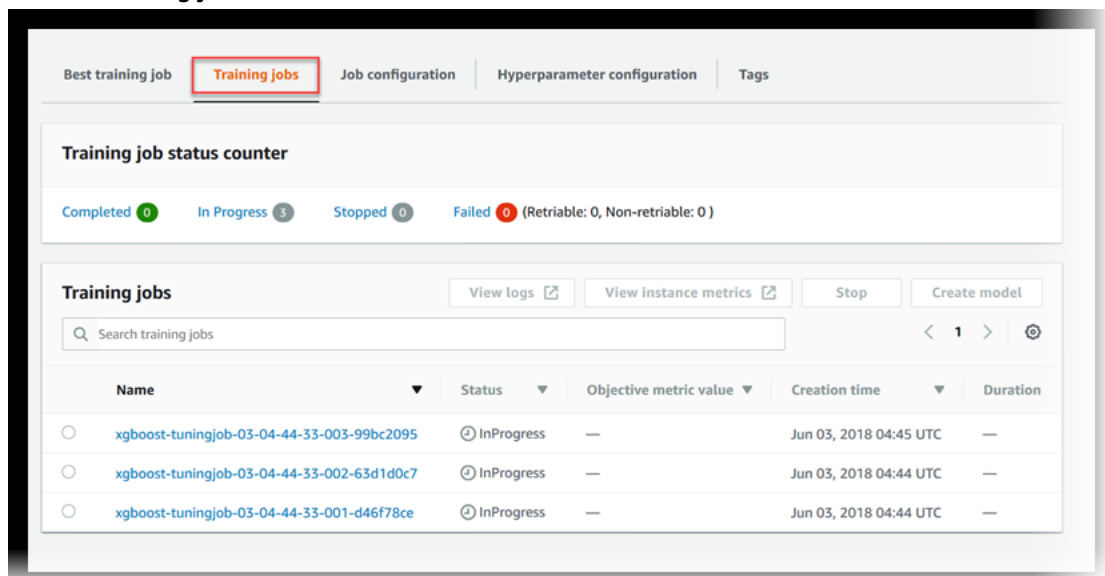
3. View the status of each training job. To see more details about a job, choose it in the list of training jobs. To view a summary of the status of all of the training jobs that the hyperparameter tuning job launched, see **Training job status counter**.

A training job can be:

- `Completed`—The training job successfully completed.
- `InProgress`—The training job is in progress.
- `Stopped`—The training job was manually stopped before it completed.
- `Failed (Retriable)`—The training job failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.
- `Failed (Non-retriable)`—The training job failed and can't be retried. A failed training job can't be retried when a client error occurs.

View the Best Training Job

A hyperparameter tuning job uses the objective metric that each training job returns to evaluate training jobs. While the hyperparameter tuning job is in progress, the best training job is the one that has returned the best objective metric so far. After the hyperparameter tuning job is complete, the best training job is the one that returned the best objective metric.

To view the best training job, choose **Best training job**.
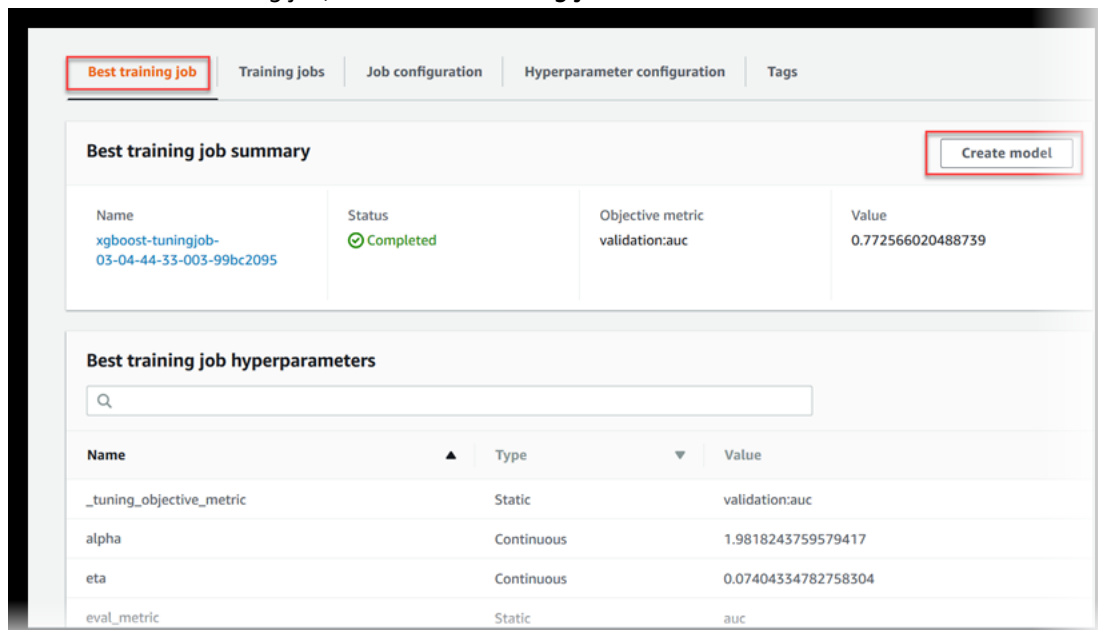
To deploy the best training job as a model that you can host at an Amazon SageMaker endpoint, choose **Create model**.

Next Step

Clean up

To avoid incurring unnecessary charges, when you are done with the example, use the AWS Management Console to delete the resources that you created for it.

> **Note**
> If you plan to explore other examples, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/ and delete the notebook instance. Stop the instance before deleting it.

2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/ and delete the bucket that you created to store model artifacts and the training dataset.

3. Open the IAM console at https://console.aws.amazon.com/iam/ and delete the IAM role. If you created permission policies, you can delete them, too.

4. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/ and delete all of the log groups that have names starting with `/aws/sagemaker/`.

Stop Training Jobs Early

Stop the training jobs that a hyperparameter tuning job launches early when they are not improving significantly as measured by the objective metric. Stopping training jobs early can help reduce compute time and helps you avoid overfitting your model. To configure a hyperparameter tuning job to stop training jobs early, do one of the following:

- If you are using the AWS SDK for Python (Boto 3), set the `TrainingJobEarlyStoppingType` field of the HyperParameterTuningJobConfig (p. 922) object that you use to configure the tuning job to `AUTO`.

- If you are using the Amazon SageMaker Python SDK, set the `early_stopping_type` parameter of the HyperParameterTuner object to `Auto`.

- In the Amazon SageMaker console, in the **Create hyperparameter tuning job** workflow, under **Early stopping**, choose **Auto**.

For a sample notebook that demonstrates how to use early stopping, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_early_stopping/hpo_image_classification_early_stopping.ipynb or open the `hpo_image_classification_early_stopping.ipynb` notebook in the **Hyperparameter Tuning** section of the **SageMaker Examples** in a notebook instance. For information about using sample notebooks in a notebook instance, see Use Example Notebooks (p. 42).

How Early Stopping Works

When you enable early stopping for a hyperparameter tuning job, Amazon SageMaker evaluates each training job the hyperparameter tuning job launches as follows:

- After each epoch of training, get the value of the objective metric.

- Compute the running average of the objective metric for all previous training jobs up to the same epoch, and then compute the median of all of the running averages.

- If the value of the objective metric for the current training job is worse (higher when minimizing or lower when maximizing the objective metric) than the median value of running averages of the objective metric for previous training jobs up to the same epoch, Amazon SageMaker stops the current training job.

Algorithms That Support Early Stopping

To support early stopping, an algorithm must emit objective metrics for each epoch. The following built-in Amazon SageMaker algorithms support early stopping:

- Linear Learner Algorithm (p. 162)—Supported only if you use `objective_loss` as the objective metric.

- XGBoost Algorithm (p. 255)

- Image Classification Algorithm (p. 108)
- Object Detection Algorithm (p. 199)
- Sequence-to-Sequence Algorithm (p. 242)
- IP Insights Algorithm (p. 131)

> **Note**
> This list of built-in algorithms that support early stopping is current as of December 13, 2018.
> Other built-in algorithms might support early stopping in the future. If an algorithm emits a
> metric that can be used as an objective metric for a hyperparameter tuning job (preferably a
> validation metric), then it supports early stopping.

To use early stopping with your own algorithm, you must write your algorithms such that it emits the
value of the objective metric after each epoch. The following list shows how you can do that in different
frameworks:

TensorFlow

Use the `tf.contrib.learn.monitors.ValidationMonitor` class. For information, see https://
www.tensorflow.org/api_docs/python/tf/contrib/learn/monitors.

MXNet

Use the `mxnet.callback.LogValidationMetricsCallback`. For information, see https://
mxnet.apache.org/api/python/callback/callback.html.

Chainer

Extend chainer by using the `extensions.Evaluator` class. For information, see https://
docs.chainer.org/en/v1.24.0/reference/extensions.html#evaluator.

PyTorch and Spark

There is no high-level support. You must explicitly write your training code so that it computes
objective metrics and writes them to logs after each epoch.

Run a Warm Start Hyperparameter Tuning Job

Use warm start to start a hyperparameter tuning job using one or more previous tuning jobs as a starting
point. The results of previous tuning jobs are used to inform which combinations of hyperparameters
to search over in the new tuning job. Hyperparameter tuning uses either Bayesian or random search to
choose combinations of hyperparameter values from ranges that you specify. For more information, see
How Hyperparameter Tuning Works (p. 289). Using information from previous hyperparameter tuning
jobs can help increase the performance of the new hyperparameter tuning job by making the search for
the best combination of hyperparameters more efficient.

> **Note**
> Warm start tuning jobs typically take longer to start than standard hyperparameter tuning
> jobs, because the results from the parent jobs have to be loaded before the job can start. The
> increased time depends on the total number of training jobs launched by the parent jobs.

Reasons you might want to consider warm start include:

- You want to gradually increase the number of training jobs over several tuning jobs based on the
results you see after each iteration.
- You get new data, and want to tune a model using the new data.
- You want to change the ranges of hyperparameters that you used in a previous tuning job, change
static hyperparameters to tunable, or change tunable hyperparameters to static values.
- You stopped a previous hyperparameter job early or it stopped unexpectedly.

Topics

Types of Warm Start Tuning Jobs

There are two different types of warm start tuning jobs:

IDENTICAL_DATA_AND_ALGORITHM

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed.

Use identical data and algorithm when you use the same training data as you used in a previous hyperparameter tuning job, but you want to increase the total number of training jobs or change ranges or values of hyperparameters.

When you run an warm start tuning job of type IDENTICAL_DATA_AND_ALGORITHM, there is an additional field in the response to DescribeHyperParameterTuningJob (p. 715) named OverallBestTrainingJob. The value of this field is the TrainingJobSummary (p. 1019) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.

TRANSFER_LEARNING

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The training algorithm image can also be a different version from the version used in the parent hyperparameter tuning job. When you use transfer learning, changes in the dataset or the algorithm that significantly affect the value of the objective metric might reduce the usefulness of using warm start tuning.

Warm Start Tuning Restrictions

The following restrictions apply to all warm start tuning jobs:

- A tuning job can have a maximum of 5 parent jobs, and all parent jobs must be in a terminal state (Completed, Stopped, or Failed) before you start the new tuning job.
- The objective metric used in the new tuning job must be the same as the objective metric used in the parent jobs.
- The total number of static plus tunable hyperparameters must remain the same between parent jobs and the new tuning job. Because of this, if you think you might want to use a hyperparameter as tunable in a future warm start tuning job, you should add it as a static hyperparameter when you create a tuning job.
- The type of each hyperparameter (continuous, integer, categorical) must not change between parent jobs and the new tuning job.

- The number of total changes from tunable hyperparameters in the parent jobs to static hyperparameters in the new tuning job, plus the number of changes in the values of static hyperparameters cannot be more than 10. Each value in a categorical hyperparameter counts against this limit. For example, if the parent job has a tunable categorical hyperparameter with the possible values `red` and `blue`, you change that hyperparameter to static in the new tuning job, that counts as 2 changes against the allowed total of 10. If the same hyperparameter had a static value of `red` in the parent job, and you change the static value to `blue` in the new tuning job, it also counts as 2 changes.
- Warm start tuning is not recursive. For example, if you create `MyTuningJob3` as a warm start tuning job with `MyTuningJob2` as a parent job, and `MyTuningJob2` is itself an warm start tuning job with a parent job `MyTuningJob1`, the information that was learned when running `MyTuningJob1` is not used for `MyTuningJob3`. If you want to use the information from `MyTuningJob1`, you must explicitly add it as a parent for `MyTuningJob3`.
- The training jobs launched by every parent job in a warm start tuning job count against the 500 maximum training jobs for a tuning job.
- Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

Warm Start Tuning Sample Notebook

For a sample notebook that shows how to use warm start tuning, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_warmstart/hpo_image_classification_warmstart.ipynb. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Example Notebooks (p. 42). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The warm start tuning example notebook is located in the **Hyperparameter tuning** section, and is named `hpo_image_classification_warmstart.ipynb`. To open a notebook, click on its **Use** tab and select **Create copy**.

Create a Warm Start Tuning Job

You can use either the low-level AWS SDK for Python (Boto 3) or the high-level Amazon SageMaker Python SDK to create a warm start tuning job.

Topics
- Create a Warm Start Tuning Job (Low-level Amazon SageMaker API for Python (Boto 3)) (p. 305)
- Create a Warm Start Tuning Job (Amazon SageMaker Python SDK) (p. 306)

Create a Warm Start Tuning Job (Low-level Amazon SageMaker API for Python (Boto 3))

To use warm start tuning, you specify the values of a HyperParameterTuningJobWarmStartConfig (p. 927) object, and pass that as the `WarmStartConfig` field in a call to CreateHyperParameterTuningJob (p. 638).

The following code shows how to create a HyperParameterTuningJobWarmStartConfig (p. 927) object and pass it to CreateHyperParameterTuningJob (p. 638) job by using the low-level Amazon SageMaker API for Python (Boto 3).

Create the `HyperParameterTuningJobWarmStartConfig` object:

```
warm_start_config = {
        "ParentHyperParameterTuningJobs" : [
        {"HyperParameterTuningJobName" : 'MyParentTuningJob'}
```

```
        ],
        "WarmStartType" : "IdenticalDataAndAlgorithm"
}
```

Create the warm start tuning job:

```
smclient = boto3.Session().client('sagemaker')
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName =
 'MyWarmStartTuningJob',
    HyperParameterTuningJobConfig = tuning_job_config, # See notebook for tuning
 configuration
    TrainingJobDefinition = training_job_definition, # See notebook for job definition
    WarmStartConfig = warm_start_config)
```

Create a Warm Start Tuning Job (Amazon SageMaker Python SDK)

To use the Amazon SageMaker Python SDK to run a warm start tuning job, you:

- Specify the parent jobs and the warm start type by using a `WarmStartConfig` object.
- Pass the `WarmStartConfig` object as the value of the `warm_start_config` argument of a HyperparameterTuner object.
- Call the `fit` method of the `HyperparameterTuner` object.

For more information about using the Amazon SageMaker Python SDK for hyperparameter tuning, see https://github.com/aws/sagemaker-python-sdk#sagemaker-automatic-model-tuning.

This example uses an estimator that uses the Image Classification Algorithm (p. 108) algorithm for training. The following code sets the hyperparameter ranges that the warm start tuning job searches within to find the best combination of values. For information about setting hyperparameter ranges, see Define Hyperparameter Ranges (p. 292).

```
hyperparameter_ranges = {'learning_rate': ContinuousParameter(0.0, 0.1),
                          'momentum': ContinuousParameter(0.0, 0.99)}
```

The following code configures the warm start tuning job by creating a `WarmStartConfig` object.

```
from sagemaker.tuner import WarmStartConfig,
        WarmStartTypes

parent_tuning_job_name = "MyParentTuningJob"
warm_start_config = WarmStartConfig(type=WarmStartTypes.IDENTICAL_DATA_AND_ALGORITHM,
 parents={parent_tuning_job_name})
```

Now set the values for static hyperparameters, which are hyperparameters that keep the same value for every training job that the warm start tuning job launches. In the following code, `imageclassification` is an estimator that was created previously.

```
imageclassification.set_hyperparameters(num_layers=18,
                                        image_shape='3,224,224',
                                        num_classes=257,
                                        num_training_samples=15420,
                                        mini_batch_size=128,
                                        epochs=30,
                                        optimizer='sgd',
                                        top_k='2',
                                        precision_dtype='float32',
                                        augmentation_type='crop')
```

Now create the `HyperparameterTuner` object and pass the `WarmStartConfig` object that you previously created as the `warm_start_config` argument.

```
tuner_warm_start = HyperparameterTuner(imageclassification,
                          'validation:accuracy',
                          hyperparameter_ranges,
                          objective_type='Maximize',
                          max_jobs=10,
                          max_parallel_jobs=2,
                          base_tuning_job_name='warmstart',
                          warm_start_config=warm_start_config)
```

Finally, call the `fit` method of the `HyperparameterTuner` object to launch the warm start tuning job.

```
tuner_warm_start.fit(
        {'train': s3_input_train, 'validation': s3_input_validation},
        include_cls_metadata=False)
```

Automatic Model Tuning Resource Limits

Amazon SageMaker sets default limits for the following resources:

- Number of concurrent hyperparameter tuning jobs - 100
- Number of hyperparameters that can be searched - 20

    Note
    Every possible value in a categorical hyperparameter counts against this limit.

- Number of metrics defined per hyperparameter tuning job - 20
- Number of concurrent training jobs per hyperparameter tuning job - 10
- Number of training jobs per hyperparameter tuning job - 500
- Maximum run time for a hyperparameter tuning job - 30 days

When you plan hyperparameter tuning jobs, you also have to take the limits on training resources into account. For information about the default resource limits for Amazon SageMaker training jobs, see Amazon SageMaker Limits. Every concurrent training instance that all of your hyperparameter tuning jobs run on count against the total number of training instances allowed. For example, suppose you run 10 concurrent hyperparameter tuning jobs. Each of those hyperparameter tuning jobs runs 100 total training jobs, and runs 20 concurrent training jobs. Each of those traning jobs runs on one **ml.m4.xlarge** instance. The following limits apply:

- Number of concurrent hyperparameter tuning jobs - You don't need to increase the limit, because 10 tuning jobs is below the limit of 100.
- Number of training jobs per hyperparameter tuning job - You don't need to increase the limit, because 100 training jobs is below the limit of 500.
- Number of concurrent training jobs per hyperparameter tuning job - You need to request a limit increase to 20, because the default limit is 10.
- Amazon SageMaker training **ml.m4.xlarge** instances - You need to request limit increase to 200, because you have 10 hyperparameter tuning jobs, with each of them running 20 concurrent training jobs. The default limit is 20 instances.
- Amazon SageMaker training total instance count - You need to request a limit increase to 200, because you have 10 hyperparameter tuning jobs, with each of them running 20 concurrent training jobs. The default limit is 20 instances.

For information about requesting limit increases for AWS resources, see AWS Service Limits.

Best Practices for Hyperparameter Tuning

Hyperparameter optimization is not a fully-automated process. To improve optimization, use the following guidelines when you create hyperparameters.

Topics

Choosing the Number of Hyperparameters

The difficulty of a hyperparameter tuning job depends primarily on the number of hyperparameters that Amazon SageMaker has to search. Although you can simultaneously use up to 20 variables in a hyperparameter tuning job, limiting your search to a much smaller number is likely to give better results.

Choosing Hyperparameter Ranges

The range of values for hyperparameters that you choose to search can significantly affect the success of hyperparameter optimization. Although you might want to specify a very large range that covers every possible value for a hyperparameter, you will get better results by limiting your search to a small range of values. If you get the best metric values within a part of a range, consider limiting the range to that part.

Using Logarithmic Scales for Hyperparameters

During hyperparameter tuning, Amazon SageMaker attempts to figure out if your hyperparameters are log-scaled or linear-scaled. Initially, it assumes that hyperparameters are linear-scaled. If they should be log-scaled, it might take some time for Amazon SageMaker to discover that. If you know that a hyperparameter should be log-scaled and can convert it yourself, doing so could improve hyperparameter optimization.

Choosing the Best Number of Concurrent Training Jobs

Running more hyperparameter tuning jobs concurrently gets more work done quickly, but a tuning job improves only through successive rounds of experiments. Typically, running one training job at a time achieves the best results with the least amount of compute time.

Running Training Jobs on Multiple Instances

When a training job runs on multiple instances, hyperparameter tuning uses the last-reported objective metric from all instances of that training job as the value of the objective metric for that training job. Design distributed training jobs so that you get they report the objective metric that you want.

Provide Dataset Metadata to Training Jobs with an Augmented Manifest File

To classify data into different groupings, you train a model by using a dataset and metadata that act as labels. To include metadata with your dataset in a training job, use an augmented manifest file.

When using an augmented manifest file, your dataset must be stored in Amazon Simple Storage Service (Amazon S3) and you must configure your training job to use dataset stored there. You specify the location and format of this dataset for one or more Channel (p. 876).

When specifying a channel's parameters, you specify a path to the file, called a `S3Uri`. Amazon SageMaker interprets this URI based on the specified `S3DataType` in S3DataSource (p. 994). The `AugmentedManifestFile` option defines a manifest format that includes metadata with the input data. Using an augmented manifest file is an alternative to preprocessing when you have labeled data. For training jobs using labeled data, you typically need to preprocess the dataset to combine input data with metadata before training. If your training dataset is large, preprocessing can be time consuming and expensive.

Augmented Manifest File Format

An augmented manifest file must be formatted in JSON Lines format. In JSON Lines format, each line in the file is a complete JSON object followed by a newline separator.

During training, Amazon SageMaker parses each JSON line and sends some or all of its attributes on to the training algorithm. You specify which attribute contents to pass and the order in which to pass them with the `AttributeNames` parameter of the CreateTrainingJob (p. 667) API. The `AttributeNames` parameter is an ordered list of attribute names that Amazon SageMaker looks for in the JSON object to use as training input.

For example, if you list `["line", "book"]` for `AttributeNames`, the input data must include the attribute names of `line` and `book` in the specified order. For this example, the following augmented manifest file content is valid:

```
{"author": "Herman Melville", "line": "Call me Ishmael", "book": "Moby Dick"}
{"line": "It was love at first sight.", "author": "Joseph Heller", "book": "Catch-22"}
```

Amazon SageMaker ignores unlisted attribute names even if they precede, follow, or are in between listed attributes.

When using augmented manifest files, observe the following guidelines:

- The order of the attributes listed in the `AttributeNames` parameter determines the order of the attributes passed to the algorithm in the training job.
- The listed `AttributeNames` can be a subset of all of the attributes in the JSON line. Amazon SageMaker ignores unlisted attributes in the file.
- You can specify any type of data allowed by the JSON format in `AttributeNames`, including text, numerical, data arrays, or objects.
- To include an S3 URI as an attribute name, add the suffix `-ref` to it.

If an attribute name contains the suffix `-ref`, the attribute's value must be an S3 URI to a data file that is accessible to the training job. For example, if `AttributeNames` contains `["image-ref", "is-a-cat"]`, a valid augmented manifest file might contain these lines:

```
{"image-ref": "s3://mybucket/sample01/image1.jpg", "is-a-cat": 1}
{"image-ref": "s3://mybucket/sample02/image2.jpg", "is-a-cat": 0}
```

For the first line of this manifest, Amazon SageMaker retrieves the contents of the S3 object `s3://mybucket/foo/image1.jpg` and streams it to the algorithm for training. The second line is the string representation of the `is-a-cat` attribute "1", which is followed by the contents of the second line.

To create an augmented manifest file, use Amazon SageMaker Ground Truth to create a labeling job. For more information, see Output Data (p. 545).

Stream Augmented Manifest File Data

Augmented manifest files are supported only for channels using Pipe input mode. For each channel, the data is extracted from its augmented manifest file and streamed (in order) to the algorithm through the channel's named pipe. Pipe mode uses the first in first out (FIFO) method, so records are processed in the order in which they are queued. For information about Pipe input mode, see InputMode.

Attribute names with a `"-ref"` suffix point to preformatted binary data. In some cases, the algorithm knows how to parse the data. In other cases, you might need to wrap the data so that records are delimited for the algorithm. If the algorithm is compatible with RecordIO-formatted data, specifying `RecordIO` for `RecordWrapperType` solves this issue. If the algorithm is not compatible with `RecordIO` format, specify `None` for `RecordWrapperType` and make sure that your data is parsed correctly for your algorithm. Using the `["image-ref", "is-a-cat"]` example, if you use RecordIO wrapping, the following stream of data is sent to the queue:

```
recordio_formatted(s3://mybucket/foo/
image1.jpg)recordio_formatted("1")recordio_formatted(s3://mybucket/bar/
image2.jpg)recordio_formatted("0")
```

Images that aren't wrapped with RecordIO format, are streamed with the corresponding `is-a-cat` attribute value as one record. This can cause a problem because the algorithm might not delimit the images and attributes correctly.

With augmented manifest files and Pipe mode in general, size limits of the EBS volume do not apply. This includes settings that otherwise must be within the EBS volume size limit such as S3DataDistributionType. For more information about Pipe mode and how to use it, see Using Your Own Training Algorithms - Input Data Configuration.

Use an Augmented Manifest File (Console)

To complete this procedure, you need:

- The URL of the S3 bucket where you've stored the augmented manifest file.
- To store the data that is listed in the augmented manifest file in an S3 bucket.
- The URL of the S3 bucket where you want to store the output of the job.

To use an augmented manifest file in a training job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker.
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. It can have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about supported built-in algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 56). If you want to use a custom algorithm, make sure that it is compatible with Pipe mode.
6. (Optional) For **Resource configuration**, either accept the default values or, to reduce computation time, increase the resource consumption.

    a. (Optional) For **Instance type**, choose the ML compute instance type that you want to use. In most cases, **ml.m4.xlarge** is sufficient.

    b. For **Instance count**, use the default, 1.

    c. (Optional) For **Additional volume per instance (GB)**, choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.

7. Provide information about the input data for the training dataset.

   a. For **Channel name**, either accept the default (`train`) or enter a more meaningful name, such as `training-augmented-manifest-file`.

   b. For **InputMode**, choose **Pipe**.

   c. For **S3 data distribution type**, choose **FullyReplicated**. When training incrementally, fully replicating causes each ML compute instance to use a complete copy of the expanded dataset. For neural-based algorithms, such as Neural Topic Model (NTM) Algorithm (p. 177), choose `ShardedByS3Key`.

   d. If the data specified in the augmented manifest file is uncompressed, set the **Compression type** to **None**. If the data is compressed using gzip, set it to **Gzip**.

   e. (Optional) For **Content type**, specify the appropriate MIME type. Content type is the multipurpose internet mail extension (MIME) type of the data.

   f. For **Record wrapper**, if the dataset specified in the augmented manifest file is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO-formatted file, choose **None**.

   g. For **S3 data type**, choose **AugmentedManifestFile**.

   h. For **S3 location**, provide the path to the bucket where you stored the augmented manifest file.

   i. For **AugmentedManifestFile attribute names**, specify the name of an attribute that you want to use. The attribute name must be present within the augmented manifest file, and is case-sensitive.

   j. (Optional) To add more attribute names, choose **Add row** and specify another attribute name for each attribute.

   k. (Optional) To adjust the order of attribute names, choose the up or down buttons next to the names. When using an augmented manifest file, the order of the specified attribute names is important.

   l. Choose **Done**.

8. For **Output data configuration**, provide the following information:

   a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.

   b. (Optional) You can use your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. For **Encryption key**, provide the key ID or its Amazon Resource Number (ARN). For more information, see KMS-Managed Encryption Keys.

9. (Optional) For **Tags**, add one or more tags to the training job. A *tag* is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with `Project` as a key and a value that refers to a project that is related to the training job, such as `Home value forecasts`.

10. Choose **Create training job**. Amazon SageMaker creates and runs the training job.

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 26).

Use an Augmented Manifest File (API)

The following shows how to train a model with an augmented manifest file using the Amazon SageMaker high-level Python library:

```
# Create a model object set to using "Pipe" mode.
model = sagemaker.estimator.Estimator(training_image,
                                      role,
                                      train_instance_count=1,
```

```
                                    train_instance_type='ml.p3.2xlarge',
                                    train_volume_size = 50,
                                    train_max_run = 360000,
                                    input_mode = 'Pipe',
                                    output_path=s3_output_location,
                                    sagemaker_session=session)

# Create a train data channel with S3_data_type as 'AugmentedManifestFile' and attribute
 names.
train_data = sagemaker.session.s3_input(your_augmented_manifest_file,
                                    distribution='FullyReplicated',
                                    content_type='image/jpeg',
                                    s3_data_type='AugmentedManifestFile',
                                    attribute_names=['source-ref', 'annotations'])
data_channels = {'train': train_data}

# Train a model.
model.fit(inputs=data_channels, logs=True)
```

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 26).

Deploy a Model

After you build and train your model, you can deploy it to get predictions in one of two ways:

- To set up a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services. For an overview on deploying a model with Amazon SageMaker hosting services, see Deploy a Model on Amazon SageMaker Hosting Services (p. 7).
- To get predictions for an entire dataset, use Amazon SageMaker batch transform. For an overview on deploying a model with Amazon SageMaker batch transform, see Get Inferences for an Entire Dataset with Batch Transform (p. 10).

Prerequisites

These topics assume that you have built and trained a machine learning model and are ready to deploy it. If you are new to Amazon SageMaker and have not completed these prerequisite tasks, work through the steps in the Get Started (p. 16) tutorial to familiarize yourself with an example of how Amazon SageMaker manages the data science process and how it handles model deployment. For more information about building a model, see Build a Model (p. 56). For information about training a model, see Train a Model (p. 276).

What do you want to do?

Amazon SageMaker provides features to manage resources and optimize inference performance when deploying machine learning models. For guidance on using inference pipelines, compiling and deploying models with Neo, Elastic Inference, and automatic model scaling, see the following topics.

- To manage data processing and real-time predictions or to process batch transforms in a pipeline, see Deploy an Inference Pipeline (p. 314).
- To train TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models once and optimize them to deploy on ARM, Intel, and Nvidia processors, see Amazon SageMaker Neo (p. 328).
- To preprocess entire datasets quickly or to get inferences from a trained model for large datasets when you don't need a persistent endpoint, see Batch Transform (p. 348).
- To speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as Amazon SageMaker hosted models using a GPU instance for your endpoint, see Amazon SageMaker Elastic Inference (EI) (p. 355).
- To dynamically adjust the number of instances provisioned in response to changes in your workload, see Automatically Scale Amazon SageMaker Models (p. 365).

Manage Model Deployments

For guidance on managing model deployments, including monitoring, troubleshooting, and best practices, and for information on storage associated with inference hosting instances:

- For tools that can be used to monitor model deployments, see Monitor Amazon SageMaker (p. 461).
- For troubleshooting model deployments, see Troubleshoot Amazon SageMaker Model Deployments (p. 380).
- For model deployment best practices, see Best Practices for Deploying Amazon SageMaker Models (p. 381).

- For information about the size of storage volumes provided for different sizes of hosting instances, see Hosting Instance Storage Volumes (p. 381).

Deploy Your Own Inference Code

For developers that need more advanced guidance on how to run your own inference code:

- To run your own inference code hosting services, see Use Your Own Inference Code with Hosting Services (p. 408).
- To run your own inference code for batch transforms, see Use Your Own Inference Code with Batch Transform (p. 411).

Guide to Amazon SageMaker

What Is Amazon SageMaker? (p. 1)

Topics

Deploy an Inference Pipeline

An *inference pipeline* is an Amazon SageMaker model that is composed of a linear sequence of two to five containers that process requests for inferences on data. You use an inference pipeline to define and deploy any combination of pretrained Amazon SageMaker built-in algorithms and your own custom algorithms packaged in Docker containers. You can use an inference pipeline to combine preprocessing, predictions, and post-processing data science tasks. Inference pipelines are fully managed.

You can add Amazon SageMaker Spark ML Serving and scikit-learn containers that reuse the data transformers developed for training models. The entire assembled inference pipeline can be considered as an Amazon SageMaker model that you can use to make either real-time predictions or to process batch transforms directly without any external preprocessing.

Within an inference pipeline model, Amazon SageMaker handles invocations as a sequence of HTTP requests. The first container in the pipeline handles the initial request, then the intermediate response is sent as a request to the second container, and so on, for each container in the pipeline. Amazon SageMaker returns the final response to the client.

When you deploy the pipeline model, Amazon SageMaker installs and runs all of the containers on each Amazon Elastic Compute Cloud (Amazon EC2) instance in the endpoint or transform job. Feature processing and inferences run with low latency because the containers are co-located on the same EC2 instances. You define the containers for a pipeline model using the CreateModel (p. 648) operation or from the console. Instead of setting one `PrimaryContainer`, you use the `Containers` parameter. to set the containers that make up the pipeline You also specify the order in which the containers are executed.

A pipeline model is immutable, but you can update an inference pipeline by deploying a new one using the UpdateEndpoint (p. 840) operation. This modularity supports greater flexibility during experimentation.

There are no additional costs for using this feature. You pay only for the instances running on an endpoint.

Topics

- Sample Notebooks for Inference Pipelines (p. 315)
- Feature Processing with Spark ML and Scikit-learn (p. 315)
- Create a Pipeline Model (p. 316)
- Run Real-time Predictions with an Inference Pipeline (p. 318)
- Run Batch Transforms with Inference Pipelines (p. 320)
- Inference Pipeline Logs and Metrics (p. 321)
- Troubleshoot Inference Pipelines (p. 326)

Sample Notebooks for Inference Pipelines

For a sample notebook that uploads and processes a dataset, trains a model, and builds a pipeline model, see the Inference Pipelines with Spark ML and XGBoost on Abalone notebook. This notebook shows how you can build your machine learning pipeline by using Spark feature Transformers and the Amazon SageMaker XGBoost algorithm. After training the model, the sample shows how to deploy the pipeline (feature Transformer and XGBoost) for real-time predictions and also performs a batch transform job using the same pipeline.

For more examples that show how to create and deploy inference pipelines, see the Inference Pipelines with SparkML and BlazingText on DBPedia and Training using SparkML on EMR and hosting on SageMaker sample notebooks. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36).

To see a list of all the Amazon SageMaker samples, after creating and opening a notebook instance, choose the **SageMaker Examples** tab. There are three inference pipeline notebooks. The first two inference pipeline notebooks just described are located in the `advanced_functionality` folder and the third notebook is in the `sagemaker-python-sdk` folder. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Feature Processing with Spark ML and Scikit-learn

Before training a model with either Amazon SageMaker built-in algorithms or custom algorithms, you can use Spark and scikit-learn preprocessors to transform your data and engineer features.

Feature Processing with Spark ML

You can run Spark ML jobs with AWS Glue, a serverless ETL (extract, transform, load) service, from your Amazon SageMaker notebook. You can also connect to existing EMR clusters to run Spark ML jobs with Amazon EMR. To do this, you need an AWS Identity and Access Management (IAM) role that grants permission for making calls from your Amazon SageMaker notebook to AWS Glue.

> **Note**
> Currently, AWS Glue supports only Python 2.7.

After engineering features, you package and serialize Spark ML jobs with MLeap into MLeap containers that you can add to an inference pipeline. You don't need to use externally managed Spark clusters.

With this approach, you can seamlessly scale from a sample of rows to terabytes of data. The same transformers work for both training and inference, so you don't need to duplicate preprocessing and feature engineering logic or develop a one-time solution to make the models persist. With inference pipelines, you don't need to maintain outside infrastructure, and you can make predictions directly from data inputs.

When you run a Spark ML job on AWS Glue, a Spark ML pipeline is serialized into MLeap format. Then, you can use the job with the SparkML Model Serving Container in an Amazon SageMaker Inference Pipeline.*MLeap* is a serialization format and execution engine for machine learning pipelines. It supports Spark, Scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

Feature Processing with Sci-kit Learn

You can run and package scikit-learn jobs into containers directly in Amazon SageMaker. For an example of Python code for building a scikit-learn featurizer model that trains on Fisher's Iris flower data set and predicts the species of Iris based on morphological measurements, see IRIS Training and Prediction with Sagemaker Scikit-learn.

Create a Pipeline Model

To create a pipeline model that can be deployed to an endpoint or used for a batch transform job, use the Amazon SageMaker console or the `CreateModel` operation.

To create an inference pipeline (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/
   .
2. Choose **Models**, and then choose **Create models** from the **Inference** group.
3. On the **Create model** page, provide a model name, choose an IAM role, and, if you want to use a private VPC, specify VPC values.
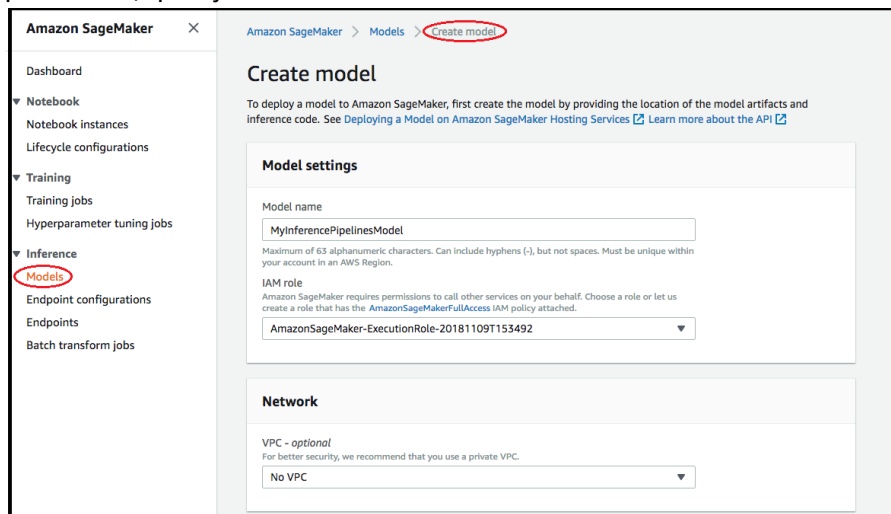
4. To add information about the containers in the inference pipeline, choose **Add container**, then choose **Next**.
5. Complete the fields for each container in the order that you want to execute them, up to the maximum of five. Complete the **Container input options**, , **Location of inference code image**, and, optionally, **Location of model artifacts**, **Container host name**, and **Environmental variables** fields. .

Container definition 1

▼ **Container input options**

○ Provide model artifacts and inference image.

▼ **Provide model artifacts and inference image**

Location of inference code image
The registry path where the inference code image is stored in Amazon ECR.

```
123456789012.dkr.ecr.us-east-2.amazonaws.com/myimage:v1
```

Location of model artifacts - *optional*
The URL for the S3 location where model artifacts are stored.

```
s3://bucket/path-to-your-data/
```

The path must point to a single gzip compressed tar archive (.tar.gz suffix).

Container host name - *optional*
The DNS host name for the container.

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

▼ **Environment variables** - *optional*

Key	Value	
key1	value1	Remove
key2	value2	Remove

Add environment variable

Container definition 2 - *optional*   Remove container

▼ **Container input options**

○ Provide model artifacts and inference image.

▼ **Provide model artifacts and inference image**

Location of inference code image
The registry path where the inference code image is stored in Amazon ECR.

```
123456789012.dkr.ecr.us-east-2.amazonaws.com/myimage:v1
```

Location of model artifacts - *optional*
The URL for the S3 location where model artifacts are stored.

```
s3://bucket/path-to-your-data/
```

The path must point to a single gzip compressed tar archive (.tar.gz suffix).

Container host name - *optional*
The DNS host name for the container.

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

▼ **Environment variables** - *optional*

Key	Value	
		Remove

Add environment variable

Container definition 3 - *optional*   Remove container

▼ **Container input options**

○ Provide model artifacts and inference image.

▼ **Provide model artifacts and inference image**

Location of inference code image
The registry path where the inference code image is stored in Amazon ECR.

```
123456789012.dkr.ecr.us-east-2.amazonaws.com/myimage:v1
```

The **MyInferencePipelineModel** page summarizes the settings for the containers that provide input for the model. If you provided the environment variables in a corresponding container definition, Amazon SageMaker shows them in the **Environment variables** field.

Run Real-time Predictions with an Inference Pipeline

You can use trained models in an inference pipeline to make real-time predictions directly without performing external preprocessing. When you configure the pipeline, you can choose to use the built-in feature transformers already available in Amazon SageMaker. Or, you can implement your own transformation logic using just a few lines of scikit-learn or Spark code.

MLeap, a serialization format and execution engine for machine learning pipelines, supports Spark, scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

The containers in a pipeline listen on the port specified in the `SAGEMAKER_BIND_TO_PORT` environment variable (instead of 8080). When running in an inference pipeline, Amazon SageMaker automatically provides this environment variable to containers. If this environment variable isn't present, containers

default to using port 8080. To indicate that your container complies with this requirement, use the following command to add a label to your Dockerfile:

```
LABEL com.amazonaws.sagemaker.capabilities.accept-bind-to-port=true
```

If your container needs to listen on a second port, choose a port in the range specified by the `SAGEMAKER_SAFE_PORT_RANGE` environment variable. Specify the value as an inclusive range in the format **"XXXX-YYYY"**, where `XXXX` and `YYYY` are multi-digit integers. Amazon SageMaker provides this value automatically when you run the container in a multicontainer pipeline.

> **Note**
> To use custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you need an Amazon Elastic Container Registry (Amazon ECR) policy. Your Amazon ECR repository must grant Amazon SageMaker permission to pull the image. For more information, see Troubleshoot Amazon ECR Permissions for Inference Pipelines (p. 326).

Create and Deploy an Inference Pipeline Endpoint

The following code creates and deploys a real-time inference pipeline model with SparkML and XGBoost models in series using the Amazon SageMaker SDK.

```python
from sagemaker.model import Model
from sagemaker.pipeline_model import PipelineModel
from sagemaker.sparkml.model import SparkMLModel

sparkml_data = 's3://{}/{}/{}'.format(s3_model_bucket, s3_model_key_prefix, 'model.tar.gz')
sparkml_model = SparkMLModel(model_data=sparkml_data)
xgb_model = Model(model_data=xgb_model.model_data, image=training_image)

model_name = 'serial-inference-' + timestamp_prefix
endpoint_name = 'serial-inference-ep-' + timestamp_prefix
sm_model = PipelineModel(name=model_name, role=role, models=[sparkml_model, xgb_model])
sm_model.deploy(initial_instance_count=1, instance_type='ml.c4.xlarge',
 endpoint_name=endpoint_name)
```

Request Real-Time Inference from an Inference Pipeline Endpoint

The following example shows how to make real-time predictions by calling an inference endpoint and passing a request payload in JSON format:

```python
from sagemaker.predictor import json_serializer, json_deserializer, RealTimePredictor
from sagemaker.content_types import CONTENT_TYPE_CSV, CONTENT_TYPE_JSON

payload = {
        "input": [
            {
                "name": "Pclass",
                "type": "float",
                "val": "1.0"
            },
            {
                "name": "Embarked",
                "type": "string",
                "val": "Q"
            },
            {
                "name": "Age",
                "type": "double",
```

```
                "val": "48.0"
            },
            {
                "name": "Fare",
                "type": "double",
                "val": "100.67"
            },
            {
                "name": "SibSp",
                "type": "double",
                "val": "1.0"
            },
            {

                "name": "Sex",
                "type": "string",
                "val": "male"
            }
        ],
        "output": {
            "name": "features",
            "type": "double",
            "struct": "vector"
        }
    }

predictor = RealTimePredictor(endpoint=endpoint_name, sagemaker_session=sess,
 serializer=json_serializer,
                              content_type=CONTENT_TYPE_JSON, accept=CONTENT_TYPE_CSV)

print(predictor.predict(payload))
```

Run Batch Transforms with Inference Pipelines

To get inferences on an entire dataset you run a batch transform on a trained model , To run inferences on a full dataset, you can use the same inference pipeline model created and deployed to an endpoint for real-time processing in a batch transform job. To run a batch transform job in a pipeline, you download the input data from Amazon S3 and send it in one or more HTTP requests to the inference pipeline model. For an example that shows how to prepare data for a batch transform, see the "Preparing Data for Batch Transform" section of the ML Pipeline with SparkML and XGBoost - Training and Inference sample notebook. For information about Amazon SageMaker batch transforms, see Get Inferences for an Entire Dataset with Batch Transform (p. 10).

> **Note**
> To use custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you need an Amazon Elastic Container Registry (Amazon ECR) policy. Your Amazon ECR repository must grant Amazon SageMaker permission to pull the image. For more information, see Troubleshoot Amazon ECR Permissions for Inference Pipelines (p. 326).

The following example shows how to run a transform job using the Amazon SageMaker Python SDK. In this example, `model_name` is the inference pipeline that combines SparkML and XGBoost models (created in previous examples). The Amazon S3 location specified by `input_data_path` contains the input data, in CSV format, to be downloaded and sent to the Spark ML model. After the transform job has finished, the Amazon S3 location specified by `output_data_path` contains the output data returned by the XGBoost model in CVS format.

```
input_data_path = 's3://{}/{}/{}'.format(default_bucket, 'key', 'file_name')
output_data_path = 's3://{}/{}'.format(default_bucket, 'key')
transform_job = sagemaker.transformer.Transformer(
    model_name = model_name,
    instance_count = 1,
    instance_type = 'ml.m4.xlarge',
```

```
    strategy = 'SingleRecord',
    assemble_with = 'Line',
    output_path = output_data_path,
    base_transform_job_name='inference-pipelines-batch',
    sagemaker_session=sess,
    accept = CONTENT_TYPE_CSV)
transform_job.transform(data = input_data_path,
                        content_type = CONTENT_TYPE_CSV,
                        split_type = 'Line')
```

Inference Pipeline Logs and Metrics

Monitoring is important for maintaining the reliability, availability, and performance of Amazon
SageMaker resources. To monitor and troubleshoot inference pipeline performance, use Amazon
CloudWatch logs and error messages. For information about the monitoring tools that Amazon
SageMaker provides, see Monitor Amazon SageMaker (p. 461).

Use Metrics to Monitor Multi-container Models

To monitor the multi-container models in Inference Pipelines, use Amazon CloudWatch. CloudWatch
collects raw data and processes it into readable, near real-time metrics. Amazon SageMaker training jobs
and endpoints write CloudWatch metrics and logs in the `AWS/SageMaker` namespace.

The following tables list the metrics and dimensions for the following:

- Endpoint invocations
- Training jobs, batch transform jobs, and endpoint instances

A *dimension* is a name/value pair that uniquely identifies a metric. You can assign up to 10 dimensions
to a metric. For more information on monitoring with CloudWatch, see Monitor Amazon SageMaker with
Amazon CloudWatch (p. 461).

Endpoint Invocation Metrics

The `AWS/SageMaker` namespace includes the following request metrics from calls to
InvokeEndpoint (p. 853) .

Metrics are reported at a 1-minute intervals.

Metric	Description
`Invocation4XXErrors`	The number of `InvokeEndpoint` requests that the model returned a `4xx` HTTP response code for. For each `4xx` response, Amazon SageMaker sends a 1. Units: None Valid statistics: `Average`, `Sum`
`Invocation5XXErrors`	The number of `InvokeEndpoint` requests that the model returned a `5xx` HTTP response code for. For each `5xx` response, Amazon SageMaker sends a 1. Units: None Valid statistics: `Average`, `Sum`

Metric	Description
Invocations	The number of InvokeEndpoint requests sent to a model endpoint.
	To get the total number of requests sent to a model endpoint, use the Sum statistic.
	Units: None
	Valid statistics: Sum, Sample Count
InvocationsPerInstance	The number of endpoint invocations sent to a model, normalized by InstanceCount in each ProductionVariant. Amazon SageMaker sends 1/numberOfInstances as the value for each request, where numberOfInstances is the number of active instances for the ProductionVariant at the endpoint at the time of the request.
	Units: None
	Valid statistics: Sum
ModelLatency	The time the model or models took to respond. This includes the time it took to send the request, to fetch the response from the model container, and to complete the inference in the container. ModelLatency is the total time taken by all containers in an inerence pipeline..
	Units: Microseconds
	Valid statistics: Average, Sum, Min, Max, Sample Count
OverheadLatency	The time added to the time taken to respond to a client request by Amazon SageMaker for overhead. OverheadLatency is measured from the time that Amazon SageMaker receives the request until it returns a response to the client, minus the ModelLatency. Overhead latency can vary depending on request and response payload sizes, request frequency, and authentication or authorization of the request, among other factors.
	Units: Microseconds
	Valid statistics: Average, Sum, Min, Max, Sample Count
ContainerLatency	The time it took for an Inference Pipelines container to respond as viewed from Amazon SageMaker. ContainerLatency includes the time it took to send the request, to fetch the response from the model's container, and to complete inference in the container.
	Units: Microseconds
	Valid statistics: Average, Sum, Min, Max, Sample Count

Dimensions for Endpoint Invocation Metrics

Dimension	Description
EndpointName, VariantName, ContainerName	Filters endpoint invocation metrics for a ProductionVariant at the specified endpoint and for the specified variant.

For an inference pipeline endpoint, CloudWatch lists per-container latency metrics in your account as **Endpoint Container Metrics** and **Endpoint Variant Metrics** in the **SageMaker** namespace, as follows. The `ContainerLatency` metric appears only for inferences pipelines.
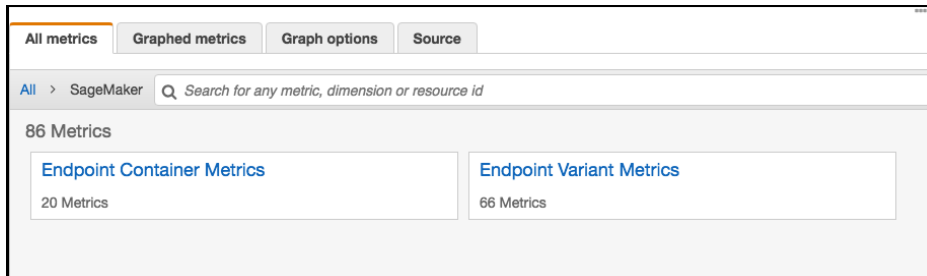
For each endpoint and each container, latency metrics display names for the container, endpoint, varian, and metric.

Training Job, Batch Transform Job, and Endpoint Instance Metrics

The namespaces `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs`, and `/aws/sagemaker/Endpoints` include the following metrics for training jobs and endpoint instances.

Metrics are reported at a 1-minute intervals.

Metric	Description
`CPUUtilization`	The percentage of CPU units that are used by the containers running on an instance. The value ranges from 0% to 100%, and is multiplied by the number of CPUs. For example, if there are four CPUs, `CPUUtilization` can range from 0% to 400%.
	For training jobs, `CPUUtilization` is the CPU utilization of the algorithm container running on the instance.
	For batch transform jobs, `CPUUtilization` is the CPU utilization of the transform container running on the instance.
	For multi-container models, `CPUUtilization` is the sum of CPU utilization by all containers running on the instance.
	For endpoint variants, `CPUUtilization` is the sum of CPU utilization by all of the containers running on the instance.
	Units: Percent
`MemoryUtilizaton`	The percentage of memory that is used by the containers running on an instance. This value ranges from 0% to 100%.
	For training jobs, `MemoryUtilizaton` is the memory used by the algorithm container running on the instance.
	For batch transform jobs, `MemoryUtilizaton` is the memory used by the transform container running on the instance.

Metric	Description
	For multi-container models, `MemoryUtilizaton` is the sum of memory used by all containers running on the instance. For endpoint variants, `MemoryUtilizaton` is the sum of memory used by all of the containers running on the instance. Units: Percent
`GPUUtilization`	The percentage of GPU units that are used by the containers running on an instance. `GPUUtilization` ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, `GPUUtilization` can range from 0% to 400%. For training jobs, `GPUUtilization` is the GPU used by the algorithm container running on the instance. For batch transform jobs, `GPUUtilization` is the GPU used by the transform container running on the instance. For multi-container models, `GPUUtilization` is the sum of GPU used by all containers running on the instance. For endpoint variants, `GPUUtilization` is the sum of GPU used by all of the containers running on the instance. Units: Percent
`GPUMemoryUtilization`	The percentage of GPU memory used by the containers running on an instance. GPUMemoryUtilization ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, `GPUMemoryUtilization` can range from 0% to 400%. For training jobs, `GPUMemoryUtilization` is the GPU memory used by the algorithm container running on the instance. For batch transform jobs, `GPUMemoryUtilization` is the GPU memory used by the transform container running on the instance. For multi-container models, `GPUMemoryUtilization` is sum of GPU used by all containers running on the instance. For endpoint variants, `GPUMemoryUtilization` is the sum of the GPU memory used by all of the containers running on the instance. Units: Percent
`DiskUtilization`	The percentage of disk space used by the containers running on an instance. DiskUtilization ranges from 0% to 100%. This metric is not supported for batch transform jobs. For training jobs, `DiskUtilization` is the disk space used by the algorithm container running on the instance. For endpoint variants, `DiskUtilization` is the sum of the disk space used by all of the provided containers running on the instance. Units: Percent

Dimensions for Training Job, Batch Transform Job, and Endpoint Instance Metrics

Dimension	Description
`Host`	For training jobs, `Host` has the format `[training-job-name]/algo-[instance-number-in-cluster]`. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the `/aws/sagemaker/TrainingJobs` namespace. For batch transform jobs, `Host` has the format `[transform-job-name]/[instance-id]`. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the `/aws/sagemaker/TransformJobs` namespace. For endpoints, `Host` has the format `[endpoint-name]/[ production-variant-name ]/[instance-id]`. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the `/aws/sagemaker/Endpoints` namespace.

To help you debug your training jobs, endpoints, and notebook instance lifecycle configurations, Amazon SageMaker also sends anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` to Amazon CloudWatch Logs. You can use this information for debugging and to analyze progress.

Use Logs to Monitor an Inference Pipeline

The following table lists the log groups and log streams Amazon SageMaker. sends to Amazon CloudWatch

A *log stream* is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

Logs

Log Group Name	Log Stream Name
`/aws/sagemaker/ TrainingJobs`	`[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]`
`/aws/sagemaker/ Endpoints/ [EndpointName]`	`[production-variant-name]/[instance-id]`
	`[production-variant-name]/[instance-id]`
	`[production-variant-name]/[instance-id]/[container-name provided in the Amazon SageMaker model] (For Inference Pipelines)` For Inference Pipelines logs, if you don't provide container names, CloudWatch uses **container-1, container-2**, and so on, in the order that containers are provided in the model.
`/aws/sagemaker/ NotebookInstances`	`[notebook-instance-name]/[LifecycleConfigHook]`
`/aws/sagemaker/ TransformJobs`	`[transform-job-name]/[instance-id]-[epoch_timestamp]`
	`[transform-job-name]/[instance-id]-[epoch_timestamp]/data-log`

Log Group Name	Log Stream Name
	`[transform-job-name]/[instance-id]-[epoch_timestamp]/` `[container-name provided in the Amazon SageMaker model]` `(For Inference Pipelines)` For Inference Pipelines logs, if you don't provide container names, CloudWatch uses **container-1, container-2**, and so on, in the order that containers are provided in the model.

Note
Amazon SageMaker creates the `/aws/sagemaker/NotebookInstances` log group when you create a notebook instance with a lifecycle configuration. For more information, see Customize a Notebook Instance (p. 40).

For more information about Amazon SageMaker logging, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 466).

Troubleshoot Inference Pipelines

To troubleshoot inference pipeline issues, use CloudWatch logs and error messages. If you are using custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you might also encounter permissions problems. To grant the required permissions, create an Amazon Elastic Container Registry (Amazon ECR) policy.

Topics

Troubleshoot Amazon ECR Permissions for Inference Pipelines

When you use custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you need an Amazon ECR policy. The policy allows your Amazon ECR repository to grant permission for Amazon SageMaker to pull the image. The policy must add the following permissions:

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "allowSageMakerToPull",
            "Effect": "Allow",
            "Principal": {
                "Service": "sagemaker.amazonaws.com"
            },
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ]
        }
    ]
}
```

Use CloudWatch Logs to Troubleshoot Amazon SageMaker Inference Pipelines

Amazon SageMaker publishes the container logs for endpoints that deploy an inference pipeline to Amazon CloudWatch at the following path for each container.

```
/aws/sagemaker/Endpoints/{EndpointName}/{Variant}/{InstanceId}/{ContainerHostname}
```

For example, logs for this endpoint are published to the following log groups and streams:

```
EndpointName: MyInferencePipelinesEndpoint
Variant: MyInferencePipelinesVariant
InstanceId: i-0179208609ff7e488
ContainerHostname: MyContainerName1 and MyContainerName2
```

```
logGroup: /aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint
logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName1
logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName2
```

A *log stream* is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

To see the log groups and streams

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation page, choose **Logs**.
3. In **Log Groups**. filter on `MyInferencePipelinesEndpoint`:
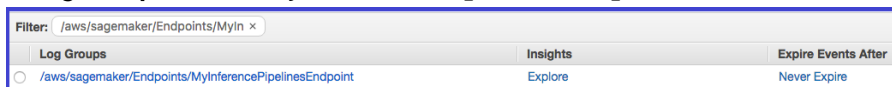
4. To see the log streams, on the CloudWatch **Log Groups** page, choose `MyInferencePipelinesEndpoint`, and then **Search Log Group**.
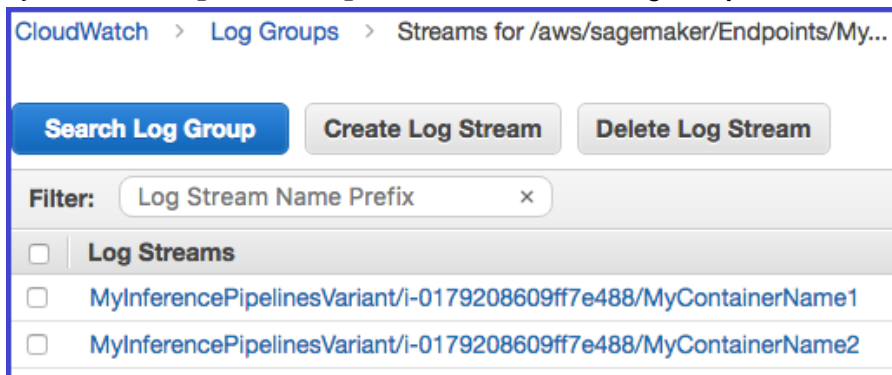
For a list of the logs that Amazon SageMaker publishes, see Inference Pipeline Logs and Metrics (p. 321).

Use Error Messages to Troubleshoot Inference Pipelines

The inference pipeline error messages indicate which containers failed.

If an error occurs while Amazon SageMaker is invoking an endpoint, the service returns a `ModelError` (error code 424), which indicates which container failed. If the request payload (the response from the previous container) exceeds the limit of 5 MB, Amazon SageMaker provides a detailed error message, such as:

Received response from MyContainerName1 with status code 200. However, the request payload from MyContainerName1 to MyContainerName2 is 6000000 bytes, which has exceeded the maximum limit of 5 MB. See https://us-west-2.console.aws.amazon.com/cloudwatch/home?region=us-west-2#logEventViewer:group=/aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint in account 123456789012 for more information.

If a container fails the ping health check while Amazon SageMaker is creating an endpoint, it returns a `ClientError` and indicates all of the containers that failed the ping check in the last health check.

Amazon SageMaker Neo

Neo is a new capability of Amazon SageMaker that enables machine learning models to train once and run anywhere in the cloud and at the edge.

Ordinarily, optimizing machine learning models for inference on multiple platforms is extremely difficult because you need to hand-tune models for the specific hardware and software configuration of each platform. If you want to get optimal performance for a given workload, you need to know the hardware architecture, instruction set, memory access patterns, and input data shapes among other factors. For traditional software development, tools such as compilers and profilers simplify the process. For machine learning, most tools are specific to the framework or to the hardware. This forces you into a manual trial-and-error process that is unreliable and unproductive.

Neo eliminates the time and effort required to do this by automatically optimizing TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models for deployment on ARM, Intel, and Nvidia processors. Neo consists of a compiler and a runtime. First, the Neo compilation API reads models exported from various frameworks. It converts the framework-specific functions and operations into a framework-agnostic intermediate representation. Next, it performs a series of optimizations. Then it generates binary code for the optimized operations, writes them to a shared object library, and saves the model definition and parameters into separate files. Neo also provides a runtime for each target platform that loads and executes the compiled model.

You can create a Neo compilation job from either the Amazon SageMaker console, AWS Command Line Interface (AWS CLI), Python notebook, or the Amazon SageMaker SDK. With a few CLI commands, an API invocation, or a few clicks, you can convert a model for your chosen platform. You can deploy the model to an Amazon SageMaker endpoint or on an AWS IoT Greengrass device quickly. Amazon SageMaker provides Neo container images for Amazon SageMaker XGBoost and Image Classification models, and supports Amazon SageMaker-compatible containers for your own compiled models.

Note
Neo currently supports image classification models exported as frozen graphs from TensorFlow, MXNet, or PyTorch, and XGBoost models. Neo is available in the following AWS Regions:

- US East (N. Virginia), us-east-1
- US West (Oregon), us-west-2
- EU (Ireland), eu-west-1

Topics

Amazon SageMaker Neo Sample Notebooks

For sample notebooks that uses Amazon SageMaker Neo to train, compile, optimize, and deploy machine learning models to make inferences, see:

- MNIST Training, Compilation and Deployment with MXNet Module
- MNIST Training, Compilation and Deployment with Tensorflow Module
- Deploying pre-trained PyTorch vision models with Amazon SageMaker Neo
- Model Optimization with an Image Classification Example
- Model Optimization with XGBoost Example

For instructions on how to run these example notebooks in Amazon SageMaker, see Use Example Notebooks (p. 42). If you need intructions on how to create a notebook instance to run these examples, see Amazon SageMaker, see Use Notebook Instances (p. 36). To navigate to the relevant example in your notebook instance, choose the **Amazon SageMaker Examples** tab to see a list of all of the Amazon SageMaker samples. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Use Neo to Compile a Model

This section show how to create, describe, stop, and list compilation jobs. There are three options available in Neo for managing the compilation jobs for machine learning models: Using the Neo CLI, the Amazon SageMaker console, or the Amazon SageMaker SDK.

Topics

Compile a Model (API)

This section shows how to manage compilation jobs for machine learning models. You can create, describe, stop, and list compilation jobs.

Create a Compilation Job

As shown in the following JSON file, you specify the data input format, the S3 bucket where you stored your model, the S3 bucket where you want to write the compiled model, and the target hardware:

```
job.json
{
    "CompilationJobName": "job002",
    "RoleArn": "arn:aws:iam::<your-account>:role/service-role/AmazonSageMaker-
ExecutionRole-20180829T140091",
    "InputConfig": {
        "S3Uri": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/train",
        "DataInputConfig":  "{\"data\": [1,3,1024,1024]}",
        "Framework": "MXNET"
    },
    "OutputConfig": {
        "S3OutputLocation": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/
compile",
```

```
        "TargetDevice": "ml_c5"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 300
    }
}


aws sagemaker create-compilation-job \
--cli-input-json file://job.json \
--region us-west-2

# You should get CompilationJobArn
```

Describe a Compilation Job

```
aws sagemaker describe-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2
```

Stop a Compilation Job

```
aws sagemaker stop-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2

# There is no output for compilation-job operation
```

List a Compilation Job

```
aws sagemaker list-compilation-jobs \
--region us-west-2
```

Compile a Model (Console)

You can create a Neo compilation job in the Amazon SageMaker console. In the **Amazon SageMaker** console, choose **Compilation jobs**, and then choose **Create compilation job**.
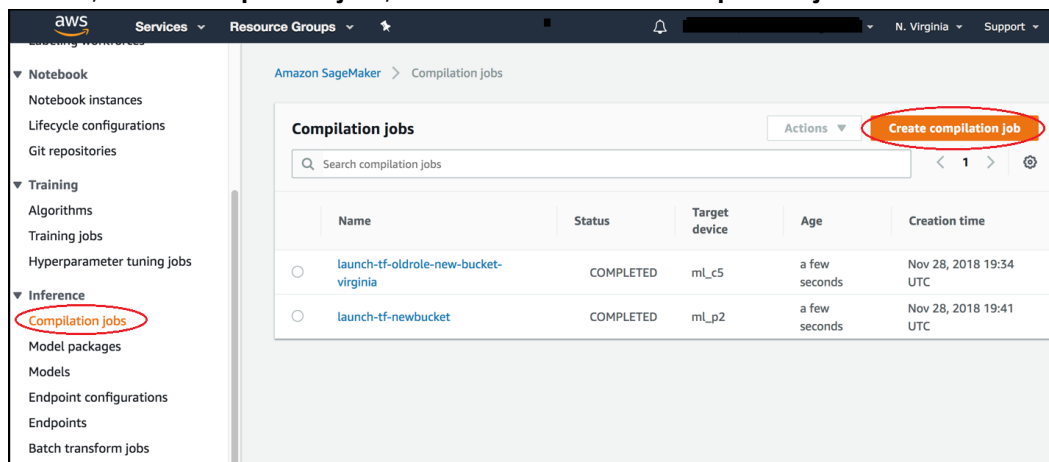
On the **Create compilation job** page, for **Job name**, enter a name. Then select an **IAM role**.
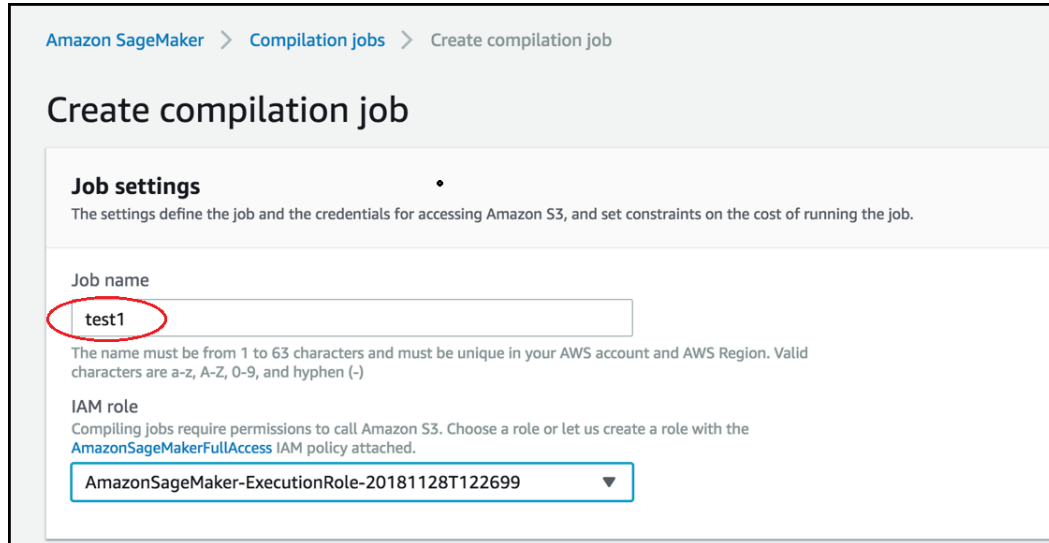
If you don't have an IAM role, choose **Create a new role**.
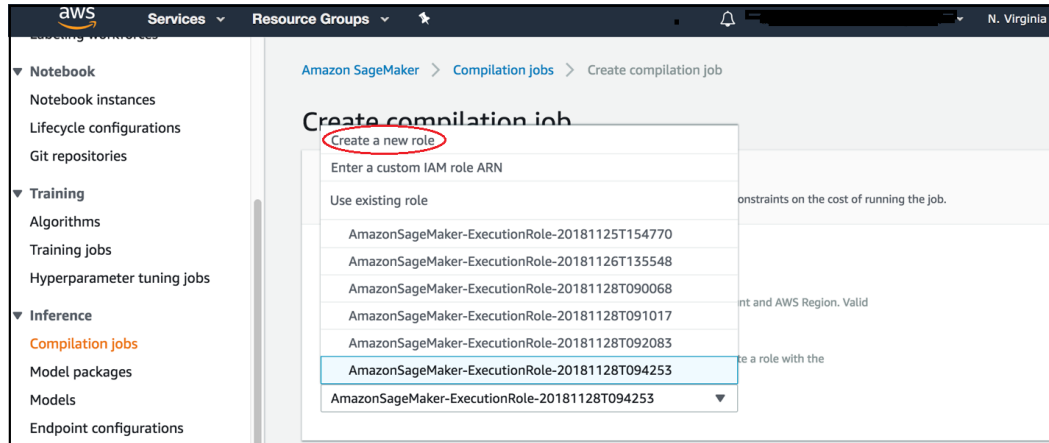
On the **Create an IAM role** page, choose **Any S3 bucket**, and choose **Create role**.

In the **Input configuration** section, for **Location of model artifacts**, enter the path of the S3 bucket that contains your model artifacts. For **Data input configuration**, enter the JSON string that specifies how many data matrix inputs you and the shape of each data matrices. For **Machine learning framework**, choose the framework.

In the **Output configuration** section, for **S3 Output location**, enter the path to the S3 bucket or folder where you want to store the model. For **Target device**, choose which device you want to deploy your model to, and choose **Create job**.

Check the status of the compilation job when started.

Check the status of the compilation job when completed.

Compile a Model (Amazon SageMaker SDK)

Follow the steps described in the **Running the Training Job** section of the MNIST Training, Compilation and Deployment with MXNet Module sample to produce a machine learning model train using Amazon SageMaker. Then you can use Neo to further optimize the model with the following code:

```
output_path = '/'.join( mnist_estimator.output_path.split('/')[:-1])
compiled_model = mnist_estimator.compile_model(target_instance_family='ml_c5',
```

```
                                        input_shape={'data':[1, 784]},
                                        role=role,
                                        output_path=output_path)
```

This code compiles the model and saves the optimized model in `output_path`. Sample notebooks of using SDK are provided in the Amazon SageMaker Neo Sample Notebooks (p. 329) section.

Deploy a Model

You can deploy the compact module to performance-critical cloud services with Amazon SageMaker Hosting Services or to resource-constrained edge devices with AWS IoT Greengrass.

Topics

- Deploy a Model Compiled with Neo with Hosting Services (p. 334)
- Deploy a Model Compiled with Neo (AWS IoT Greengrass) (p. 341)

Deploy a Model Compiled with Neo with Hosting Services

To deploy a Neo-compiled model to an HTTPS endpoint, you must configure and create the endpoint for the model using Amazon SageMaker hosting services. Currently developers can use Amazon SageMaker APIs to deploy modules on to ml.c5, ml.c4, ml.m5, ml.m4, ml.p3, and ml.p2 instances.

When you deploy a compiled model, you need to use the same instance for the target that you used for compilation. This creates an Amazon SageMaker endpoint that you can use to perform inferences. There are three options available for deploying Neo-compiled models:

Topics

- Deploy a Model Compiled with Neo (AWS CLI) (p. 334)
- Deploy a Model Compiled with Neo (Console) (p. 336)
- Deploy a Model Compiled with Neo (Amazon SageMaker SDK) (p. 341)

Deploy a Model Compiled with Neo (AWS CLI)

The deployment of a Neo-compiled model with the CLI has three steps.

Topics

- Create a Model That Was Compiled with Neo (AWS CLI) (p. 334)
- Create the Endpoint Configuration (AWS CLI) (p. 336)
- Create an Endpoint (AWS CLI) (p. 336)

Create a Model That Was Compiled with Neo (AWS CLI)

For the full syntax of the `CreateModel` API, see CreateModel (p. 648).

For Neo-compiled models, use one of the following values for `PrimaryContainer/ContainerHostname`, depending on your region and applications:

- **Amazon SageMaker Image Classification**
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest`
  - `785573368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest`

- **Amazon SageMaker XGBoost**
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest`
  - `785573368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest`
- **TensorFlow** : The TensorFlow version used must be in TensorFlow SageMaker Estimators list.
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`
  - `785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`
- **MXNet** The MXNet version used must be in MXNet SageMaker Estimators list.
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`
  - `785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`
- **Pytorch** The Pytorch version used must be in Pytorch SageMaker Estimators list.
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`
  - `785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`

Also, if you are using **TensorFlow**, **Pytorch**, or **MXNet**, add the following key-value pair to `PrimaryContainer/Environment`:

```
"Environment": {
"SAGEMAKER_SUBMIT_DIRECTORY" : "[Full S3 path for *.tar.gz file containing the training
 script]"
}
```

The script must be packaged as a `*.tar.gz` file. The `*.tar.gz` file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

- `neo_preprocess(payload, content_type)`: Function that takes in the payload and `Content-Type` of each incoming request and returns a NumPy array.
- `neo_postprocess(result)`: Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the Amazon SageMaker Neo Sample Notebooks (p. 329) for examples using these functions.

Create the Endpoint Configuration (AWS CLI)

For the full syntax of the `CreateEndpointConfig` API, see CreateEndpointConfig (p. 635). You must specify the correct instance type in `ProductionVariants/InstanceType`. It is imperative that this value matches the instance type specified in your compilation job.

Create an Endpoint (AWS CLI)

For the full syntax of the `CreateEndpoint` API, see CreateEndpoint (p. 632).

Deploy a Model Compiled with Neo (Console)

You can create a Neo endpoint in the Amazon SageMaker console. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

Choose **Models**, and then choose **Create models** from the **Inference** group. On the **Create model** page, complete the **Model name**, **IAM role**, and, if needed, **VPC** fields.

To add information about the container used to deploy your model, choose **Add** container, then choose **Next**. Complete the **Container input options**, **Location of inference code image**, and **Location of model artifacts**, and optionally, **Container host name**, and **Environmental variables** fields.

To deploy Neo-compiled models, choose the following:

- **Container input options**: Provide model artifacts and inference image
- **Location of inference code image**: Choose one of the following images, depending the region and kind of application:
  - **Amazon SageMaker Image Classification**
    - `301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest`
    - `785573368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest`
    - `007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest`
    - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest`
  - **Amazon SageMaker XGBoost**
    - `301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest`
    - `785573368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest`
    - `007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest`
    - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest`
  - **TensorFlow** : The TensorFlow version used must be in TensorFlow SageMaker Estimators list.

- 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3

- 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3

- 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3

- 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3

- **MXNet** The MXNet version used must be in MXNet SageMaker Estimators list.

  - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-
    version]-[cpu/gpu]-py3

  - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-
    version]-[cpu/gpu]-py3

  - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-
    version]-[cpu/gpu]-py3

  - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-
    version]-[cpu/gpu]-py3

- **Pytorch** The Pytorch version used must be in Pytorch SageMaker Estimators list.

  - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch:
    [pytorch-version]-[cpu/gpu]-py3

  - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch:
    [pytorch-version]-[cpu/gpu]-py3

  - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch:
    [pytorch-version]-[cpu/gpu]-py3

  - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:
    [pytorch-version]-[cpu/gpu]-py3

- **Location of model artifact**: the full S3 path of the compiled model artifact generated by the Neo compilation API.

- **Environmental variables**:

  - Omit this field for **SageMaker Image Classification** and **SageMaker XGBoost**.

  - For **TensorFlow**, **Pytorch**, and **MXNet**, specify the environment variable **SAGEMAKER_SUBMIT_DIRECTORY** as the full S3 path that contains the training script.

The script must be packaged as a `*.tar.gz` file. The `*.tar.gz` file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

- `neo_preprocess(payload, content_type)`: Function that takes in the payload and `Content-Type` of each incoming request and returns a NumPy array.

- `neo_postprocess(result)`: Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the Amazon SageMaker Neo Sample Notebooks (p. 329) for examples using these functions.

Confirm that the information for the containers is accurate, and then choose **Create model**.This takes you to the create model landing page. Select the **Create endpoint** button there.
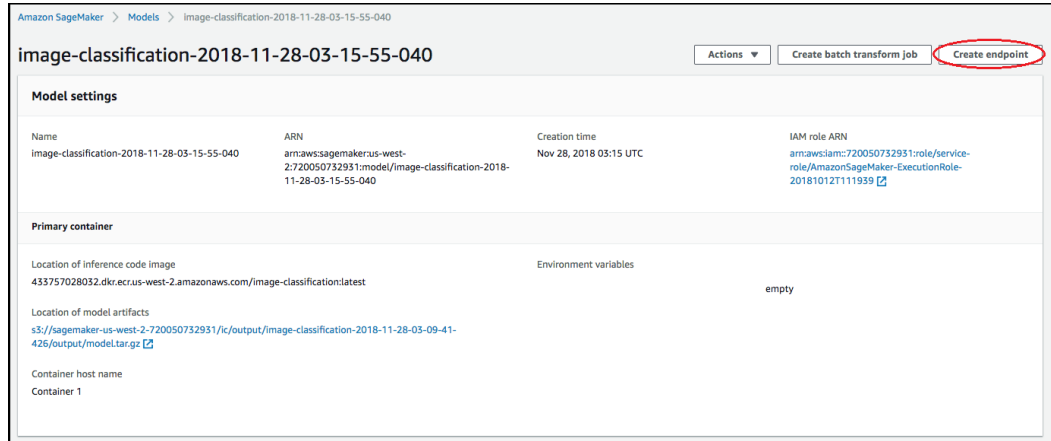
In **Create and configure endpoint** diagram, specify the **Endpoint name**. Choose **Create a new endpoint configuration** in **Attach endpoint configuration**.
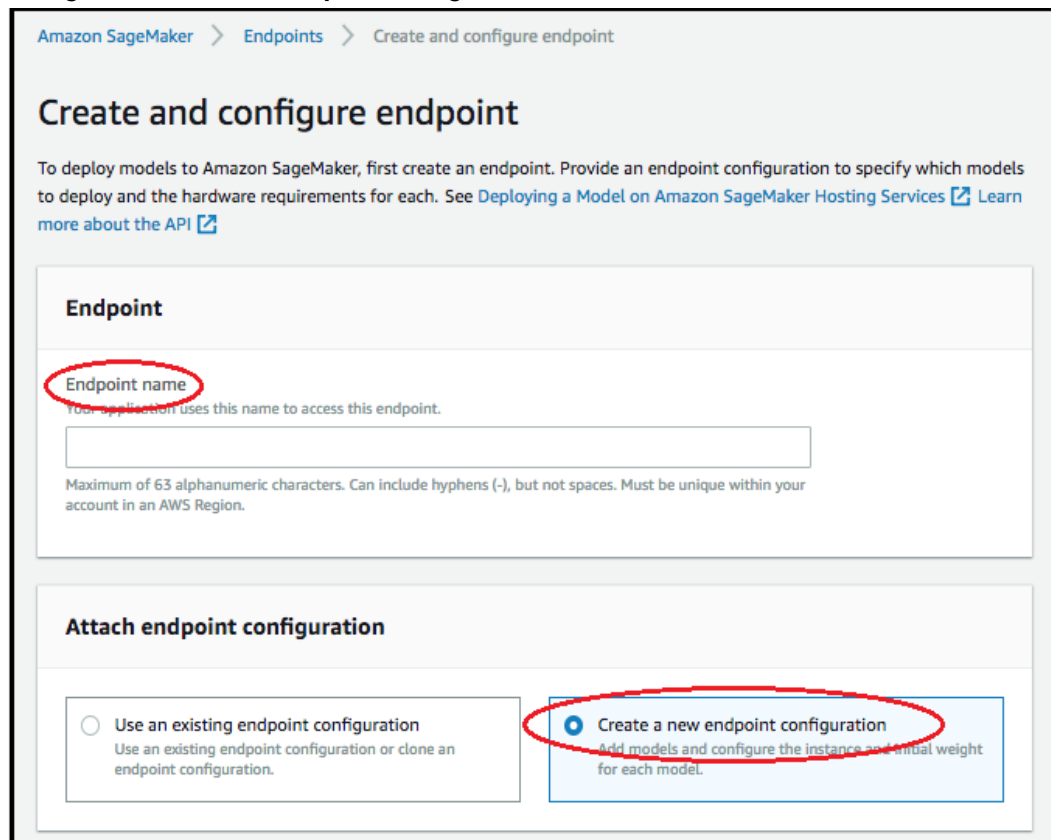
In **New endpoint configuration** page, specify the **Endpoint configuration name**.

Then press **Edit** next to the name of the model and specify the correct **Instance type** on the **Edit Production Variant** page. It is imperative that the **Instance type** value match the one specified in your compilation job.

When you're done click **Save**, then click **Create endpoint configuration** on the **New endpoint configuration** page, and then click **Create endpoint**.

Deploy a Model Compiled with Neo (Amazon SageMaker SDK)

The object handle for the compiled model supplies the `deploy` function, which allows you to create an endpoint to serve inference requests. The function lets you set the number and type of instances that are used for the endpoint. You must choose an instance for which you have compiled your model. For example, in the job compiled in Compile a Model (Amazon SageMaker SDK) (p. 333) section, this is `ml_c5`. The Neo API uses a special runtime, the *Neo runtime*, to run Neo-optimized models.

```
predictor = compiled_model.deploy(initial_instance_count = 1, instance_type =
 'ml.c5.4xlarge')
```

After the command is done, the name of the newly created endpoint is printed in the jupyter notebook.

Deploy a Model Compiled with Neo (AWS IoT Greengrass)

AWS IoT Greengrass extends cloud capabilities to local devices. It enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. With AWS IoT Greengrass, you can perform machine learning inference at the edge on locally generated data using cloud-trained models. Currently, you can deploy models on to all AWS IoT Greengrass devices based on ARM® Cortex-A™, Intel® Atom™, and Nvidia® Jetson™ series processors. For more information on deploying a Lambda inference application to perform machine learning inferences with AWS IoT Greengrass, see Perform Machine Learning Inference.

Request Inferences from a Deployed Service

If you have followed instructions in Deploy a Model Compiled with Neo with Hosting Services (p. 334), you should have an Amazon SageMaker endpoint set up and running. You can now submit inference requests using Boto3 client. Here is an example of sending an image for inference:

```
import boto3
import json

endpoint = '<insert name of your endpoint here>'

runtime = boto3.Session().client('sagemaker-runtime')

# Read image into memory
with open(image, 'rb') as f:
    payload = f.read()
# Send image via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='application/x-
image', Body=payload)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

For XGBoost application, you should submit a CSV text instead:

```
import boto3
import json

endpoint = '<insert your endpoint here>'

runtime = boto3.Session().client('sagemaker-runtime')

csv_text = '1,-1.0,1.0,1.5,2.6'
# Send CSV text via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='text/csv',
 Body=csv_text)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

Note that BYOM allows for a custom content type. For more information, see InvokeEndpoint (p. 853).

Troubleshooting Neo Compilation Errors

This section contains information about how to understand and prevent common errors, the error messages they generate, and guidance on how to resolve these errors. It also contains lists of the frameworks and the operations in each of those frameworks that Neo supports.

Topics

- Prevent Neo Input Errors (p. 342)
- Neo Error Messages (p. 346)
- Resolve Neo Errors (p. 348)

Prevent Neo Input Errors

Some of the most common errors are due to invalid inputs. This section contains information arranged in question and answer form to help you avoid these errors.

Which frameworks does Neo support?

- TensorFlow
- PyTorch
- Apache MXNET
- XGBoost
- ONNX

Which operators does Amazon SageMaker Neo support for these frameworks?

The following table lists the supported operations for each framework.

MXNet	TensorFlow	PyTorch/ONNX
`'_add_scalar'`	`'Add'`	`'Abs'`
`'_add_symbol'`	`'ArgMax'`	`'Add'`
`'_contrib_MultiBoxDetection'`	`'ArgMin'`	`'ArgMax'`
`'_contrib_MultiBoxPrior'`	`'AvgPool'`	`'ArgMin'`
`'_copy'`	`'BatchNormWithGlobalNormalization'`	`'AveragePool'`
`'_div_scalar'`	`'BiasAdd'`	`'BatchNormalization'`
`'_div_symbol'`	`'Cast'`	`'Cast'`
`'_minus_scalar'`	`'Ceil'`	`'Ceil'`
`'_minus_scalar'`	`'CheckNumerics'`	`'Clip'`
`'_mul_symbol'`	`'Concat'`	`'Concat'`
`'_Plus'`	`'ConcatV2'`	`'Constant'`
`'_plus_scalar'`	`'Conv2D'`	`'ConstantFill'`
`'_pow_scalar'`	`'DecodeJpeg'`	`'Conv'`
`'_rdiv_scalar'`	`'DepthwiseConv2dNative'`	`'ConvTranspose'`
`'_rminus_scalar'`	`'Elu'`	`'Div'`
`'_rpow_scalar'`	`'Equal'`	`'Dropout'`
`'_rsub_scalar'`	`'ExpandDims'`	`'Elu'`
`'_sub_scalar'`	`'Fill'`	`'Exp'`
`'_sub_symbol'`	`'Floor'`	`'FC'`
`'Activation'`	`'FusedBatchNorm'`	`'Flatten'`
`'add_n'`	`'FusedBatchNormV2'`	`'Floor'`
`'argmax'`	`'GatherV2'`	`'Gather'`
`'BatchNorm'`	`'Greater'`	`'Gemm'`
`'BatchNorm_v1'`	`'GreaterEqual'`	`'GlobalAveragePool'`

MXNet	TensorFlow	PyTorch/ONNX
'broadcast_add'	'Identity'	'GlobalMaxPool'
'broadcast_div'	'LeakyRelu'	'HardSigmoid'
'broadcast_mul'	'Less'	'Identity'
'broadcast_sub'	'LessEqual'	'ImageScaler'
'broadcast_to'	'LRN'	'LeakyRelu'
'cast'	'MatMul'	'Log'
'Cast'	'Maximum'	'LogSoftmax'
'clip'	'MaxPool'	'LRN'
'Concat'	'Mean'	'MatMul'
'concat'	'Minimum'	'Max'
'Convolution'	'Mul'	'MaxPool'
'Convolution_v1'	'NotEqual'	'Mean'
'Crop'	'Pack'	'Min'
'Deconvolution'	'Pad'	'Mul'
'Dropout'	'PadV2'	'Neg'
'elemwise_add'	'Range'	'Pad'
'elemwise_div'	'Rank'	'ParametricSoftplus'
'elemwise_mul'	'Relu'	'Pow'
'elemwise_sub'	'Relu6'	'PRelu'
'exp'	'Reshape'	'Reciprocal'
'expand_dims'	'ResizeBilinear'	'ReduceMax'
'flatten'	'Rsqrt'	'ReduceMean'
'Flatten'	'Selu'	'ReduceMin'
'FullyConnected'	'Shape'	'ReduceProd'
'LeakyReLU'	'Sigmoid'	'ReduceSum'
'LinearRegressionOutput'	'Softmax'	'Relu'
'log'	'Square'	'Reshape'
'log_softmax'	'Squeeze'	'Scale'
'LRN'	'StridedSlice'	'ScaledTanh'
'max'	'Sub'	'Selu'
'max_axis'	'Sum'	'Shape'

MXNet	TensorFlow	PyTorch/ONNX
`'min'`	`'Tanh'`	`'Sigmoid'`
`'min_axis'`	`'Transpose'`	`'Slice'`
`'negative'`		`'Softmax'`
`'Pooling'`		`'SoftPlus'`
`'Pooling_v1'`		`'Softsign'`
`'relu'`		`'SpatialBN'`
`'Reshape'`		`'Split'`
`'reshape'`		`'Sqrt'`
`'reshape_like'`		`'Squeeze'`
`'sigmoid'`		`'Sub'`
`'slice_like'`		`'Sum'`
`'SliceChannel'`		`'Tanh'`
`'softmax'`		`'ThresholdedRelu'`
`'Softmax'`		`'Transpose'`
`'SoftmaxActivation'`		`'Unsqueeze'`
`'SoftmaxOutput'`		`'Upsample'`
`'split'`		
`'sum'`		
`'sum_axis'`		
`'tanh'`		
`'transpose'`		
`'UpSampling'`		

Which model architectures does Neo support?

Neo supports image classification models.

Which model format files does Neo read in?

The file needs to be formatted as a tar.gz file that includes additional files that depend on the type of framework.

- **TensorFlow**: Neo supports saved models and frozen models.

  For *saved models*, Neo expects one .pb or one .pbtxt file and a variables directory that contains variables.

  For *frozen models*, Neo expect only one .pb or .pbtxt file.
- **PyTorch**: Neo expects one .pth file containing the model definition.

- **MXNET**: Neo expects one symbol file (.json) and one parameter file (.params).
- **XGBoost**: Neo expects one XGBoost model file (.model) where the number of nodes in a tree can't exceed 2^31.
- **ONNX**: Neo expects one .onnx file.

What input data shapes does Neo expect?

Neo expects the name and shape of the expected data inputs for your trained model with a JSON dictionary form or list form. The data inputs are framework specific.

- `TensorFlow`: You must specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, `{"input":[1,1024,1024,3]}`
    - If using the CLI, `{\"input\":[1,1024,1024,3]}`
  - Examples for two inputs:
    - If using the console, `{"data1": [1,28,28,1], "data2":[1,28,28,1]}`
    - If using the CLI, `{\"data1\": [1,28,28,1], \"data2\":[1,28,28,1]}`
- `MXNET/ONNX`: You must specify the name and shape (NCHW format) of the expected data inputs in order using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, `{"data":[1,3,1024,1024]}`
    - If using the CLI, `{\"data\":[1,3,1024,1024]}`
  - Examples for two inputs:
    - If using the console, `{"var1": [1,1,28,28], "var2":[1,1,28,28]}`
    - If using the CLI, `{\"var1\": [1,1,28,28], \"var2\":[1,1,28,28]}`
- `PyTorch`: You can either specify the name and shape (NCHW format) of expected data inputs in order using a dictionary format for your trained model or you can specify the shape only using a list format. The dictionary formats required for the console and CLI are different. The list formats for the console and CLI are the same.
  - Examples for one input in dictionary format:
    - If using the console, `{"input0":[1,3,224,224]}`
    - If using the CLI, `{\"input0\":[1,3,224,224]}`
  - Example for one input in list format: `[[1,3,224,224]]`
  - Examples for two inputs in dictionary format:
    - If using the console, `{"input0":[1,3,224,224], "input1":[1,3,224,224]}`
    - If using the CLI, `{\"input0\":[1,3,224,224], \"input1\":[1,3,224,224]}`
  - Example for two inputs in list format: `[[1,3,224,224], [1,3,224,224]]`
- `XGBOOST`: input data name and shape are not needed.

Neo Error Messages

This section lists and classifies Neo errors and error messages.

Neo Error Messages

This list catalogs the user and system error messages you can receive from Neo deployments.

- **User error messages**
  - **Client permission error**: Neo passes the errors for these straight through from the dependent service.

    Access Denied when calling sts:AssumeRole

    Any *400 error* when calling S3 to download or upload a client model.

    PassRole error
  - **Load error**: Keywords in error messages, 'InputConfiguration','ModelSizeTooBig'.

    Load Error: InputConfiguration: Exactly one {.xxx} file is allowed for {yyy} model.

    Load Error: ModelSizeTooBig: number of nodes in a tree can't exceed 2^31
  - **Compilation error**: Keywords in error messages, 'OperatorNotImplemented',' OperatorAttributeNotImplemented', 'OperatorAttributeRequired', 'OperatorAttributeValueNotValid'.

    OperatorNotImplemented: {xxx} is not supported.

    OperatorAttributeNotImplemented: {xxx} is not supported in {yyy}.

    OperatorAttributeRequired: Required attribute {xxx} not found in {yyy}.

    OperatorAttributeValueNotValid: The value of attribute {xxx} in operator {yyy} cannot be negative.
  - **Any Malformed Input Errors**
- **System error messages**
  - For system errors, Neo shows only one error message similar to the following: There was an unexpected error during compilation, check your inputs and try again in a few minutes.
  - This covers all unexpected errors and errors that are not user errors.

Neo Error Classifications

This list classifies the *user errors* you can receive from Neo. These include access and permission errors and load errors for each of the supported frameworks. All other errors are *system errors*.

- **Client permission error**: Neo passes the errors for these straight through from the dependent service.

  Access Denied when calling sts:AssumeRole

  Any *400 error* when calling Amazon S3 to download or upload a client model.

  PassRole error
- **Load error**: Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, check whether the tarball contains the necessary files for compilation. The checking criteria is framework-specific:
  - **TensorFlow**: Expects only protobuf file (*.pb or *.pbtxt). For *saved models*, expects one `variables` folder.
  - **Pytorch**: Expect only one pytorch file (*.pth).
  - **MXNET**: Expect only one symbol file (*.json) and one parameter file (*.params).
  - **XGBoost**: Expect only one XGBoost model file (*.model). The input model has size limitation.
- **Compilation error**: Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, and that the tarball contains necessary files for compilation. The checking criteria is:
  - **OperatorNotImplemented**: An operator has not been implemented.
  - **OperatorAttributeNotImplemented**: The attribute in the specified operator has not been implemented.

- **OperatorAttributeRequired**: An attribute is required for an internal symbol graph, but it is not listed in the user input model graph.

- **OperatorAttributeValueNotValid**: The value of the attribute in the specific operator is not valid.

Resolve Neo Errors

This section provides guidance on troubleshooting common issues with Neo. These include permission, load, compilation, and system errors and errors involving invalid inputs and unsupported operations.

- **Catalog of Known Issues**:
  - If you see **Client Permission Error**, review the set up documentation and make sure that you have correctly granted the permissions that are failing.
  - If you see **Load Error**, check the model format files that Neo expects for different frameworks.
  - If you see **Compilation Error**, check and address the details error message in your input model graph.
  - If you see **System Error**, try again in a few minutes. If that fails, file a ticket.
- **Lack of Roles and Permissions**: Review the set up documentation and make sure that you have correctly granted the permissions that are failing.
- **Invalid API and Console Inputs**: Fix your input as described in the validation error.
- **Unsupported Operators**:
  - Check the failure reason where Neo has listed all unsupported operators with the keyword 'OperatorNotImplemented'.
  - For example: Compilation Error: OperatorNotImplemented: The following operators are not implemented: {'_sample_multinomial', 'RNN' }
  - Remove the unsupported operators from your input model graph and test it again.

Batch Transform

To preprocess or get inferences for an entire dataset, use batch transform. Use batch transform when you need to work with large datasets, process datasets quickly or sub-second latency. Use preprocessing to remove noise or bias from your dataset that interferes with training or inference. Use batch transform for inference when you don't need a persistent endpoint. You can use batch transform for example, to compare production variants that deploy different models.

To filter input data before performing inferences or to associate input records with inferences about those records, use Associate Prediction Results with their Corresponding Input Records (p. 351). This is useful for example, to provide context for creating and interpreting reports about the output data.

For more information about batch transform, see Get Inferences for an Entire Dataset with Batch Transform (p. 10).

Topics

Use Batch Transform with Large Datasets

Batch transform automatically manages the processing of large datasets within the limits of specified parameters. For example, suppose that you have a dataset file, `input1.csv`, stored in an S3 bucket. The content of the input file might look like this:

```
Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-AttributeM
Record2-Attribute1, Record2-Attribute2, Record2-Attribute3, ..., Record2-AttributeM
Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-AttributeM
...
RecordN-Attribute1, RecordN-Attribute2, RecordN-Attribute3, ..., RecordN-AttributeM
```

When a batch transform job starts, Amazon SageMaker initializes compute instances and distributes the inference or preprocessing workload between them. When you have multiples files, one instance might process `input1.csv`, and the other instance might process another file named `input2.csv`. To keep large payloads within the MaxPayloadInMB limit, you might split an input file into several mini-batches. For example, you might create a mini-batch created from `input1.csv`, as follows.

```
Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-AttributeM
Record4-Attribute1, Record4-Attribute2, Record4-Attribute3, ..., Record4-AttributeM
```

> **Note**
> Amazon SageMaker processes each input file separately. It doesn't combine mini-batches from different input files to comply with the MaxPayloadInMB limit.

To split input files into mini-batches, when you create a batch transform job, set the SplitType parameter value to `Line`. If SplitType is set to `None` or if an input file can't be split into mini-batches, Amazon SageMaker uses the entire input file in a single request.

If the batch transform job successfully processes all of the records in an input file, it creates an output file with the same name and an `.out` file extension. For multiple input files, such as `input1.csv` and `input2.csv`, the output files are named `input1.csv.out`, and `input2.csv.out`. The batch transform job stores the output files in the specified location in Amazon S3, such as `s3://awsexamplebucket/output/`. The predictions in an output file are listed in the same order as the corresponding records in the input file. The following would be the contents of the output file `input1.csv.out`, based on the input file shown earlier.

```
Inference1-Attribute1, Inference1-Attribute2, Inference1-Attribute3, ..., Inference1-
AttributeM
Inference2-Attribute1, Inference2-Attribute2, Inference2-Attribute3, ..., Inference2-
AttributeM
Inference3-Attribute1, Inference3-Attribute2, Inference3-Attribute3, ..., Inference3-
AttributeM
...
InferenceN-Attribute1, Inference3-Attribute2, Inference3-Attribute3, ..., InferenceN-
AttributeM
```

To combine the results of multiple output files into a single output file, set the AssembleWith parameter to `Line`.

When the input data is very large and is transmitted using HTTP chunked encoding, to stream the data to the algorithm, set MaxPayloadInMB to `0`. Currently, Amazon SageMaker built-in algorithms don't support this feature.

For information about using the API to create a batch transform job, see the
CreateTransformJob (p. 673) API. For more information about the correlation between batch transform
input and output objects, see OutputDataConfig. For an example of how to use batch transform, see
Step 6.2: Deploy the Model with Batch Transform (p. 28).

Speed Up a Batch Transform Job

If you are using the CreateTransformJob API, you can reduce the time it takes to complete batch
transform jobs by using different parameter values, such as MaxPayloadInMB, MaxConcurrentTransforms,
and BatchStrategy. Amazon SageMaker automatically finds the optimal parameter settings for built-in
algorithms. For custom algorithms, provide these values through an execution-parameters endpoint.

If you are using the Amazon SageMaker console, you can reduce the time it takes to complete batch
transform jobs by using different parameter values, such as **Max payload size (MB)**, **Max concurrent
transforms**, and **Batch strategy**, in the **Additional configuration** section of the **Batch transform job
configuration** page. Amazon SageMaker automatically finds the optimal parameter settings for built-in
algorithms. For custom algorithms, provide these values through an execution-parameters endpoint.

Use Batch Transform to Test Production Variants

To test different models or various hyperparameter settings, create a separate transform job for each
new model variant and use a validation dataset. For each transform job, specify a unique model name
and location in Amazon S3 for the output file. To analyze the results, use Inference Pipeline Logs and
Metrics (p. 321).

Batch Transform Errors

Amazon SageMaker uses the Amazon S3 Multipart Upload API to upload results from a batch transform
job to Amazon S3. If an error occurs, the uploaded results are removed from Amazon S3. In some cases,
such as when a network outage occurs, an incomplete multipart upload might remain in Amazon S3.
To avoid incurring storage charges, we recommend that you add the S3 bucket policy to the S3 bucket
lifecycle rules. This policy deletes incomplete multipart uploads that might be stored in the S3 bucket.
For more information, see Object Lifecycle Management.

If a batch transform job fails to process an input file because of a problem with the dataset, Amazon
SageMaker marks the job as "failed" to alert you. If an input file contains a bad record, the transform
job doesn't create an output file for that input file because it can't maintain the same order in the
transformed data. When your dataset has multiple input files, a transform job continues to process input
files even if it fails to process one. The processed files still generate useable results.

Exceeding the MaxPayloadInMB limit causes an error. This might happen with a large dataset if it can't be
split, the SplitType parameter is set to `none`, or individual records within the dataset exceed the limit.

If you are using your own algorithms, you can use placeholder text, such as `ERROR`, when the algorithm
finds a bad record in an input file. For example, if the last record in a dataset is bad, the algorithm should
place the error placeholder for that record in the output file.

Batch Transform Sample Notebooks

For a sample notebook that uses batch transform to with a PCA model as a data reduction step on user-
item review matrix followed by DBSCAN to cluster movies, see https://github.com/awslabs/amazon-
sagemaker-examples/blob/master/sagemaker_batch_transform/introduction_to_batch_transform/
batch_transform_pca_dbscan_movie_clusters.ipynb. For instructions on creating and accessing Jupyter
notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook
Instances (p. 36). After creating and opening a notebook instance, choose the **SageMaker Examples** tab

to see a list of all the Amazon SageMaker examples. The topic modeling example notebooks that use the NTM algorithms are located in the **Advanced functionality** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Associate Prediction Results with their Corresponding Input Records

When making predictions on a large dataset, attributes that are not needed for prediction can be excluded. After the predictions have been made, you often want to associate some of the excluded attributes with those predictions or with other input data in your report. Amazon SageMaker Batch Tranform enables these data processing steps, often eliminating the need for any additional pre-processing or post-processing. The feature supports JSON and CSV formatted input files.

Topics

- Data Processing Workflow for a Batch Transform Job (p. 351)
- Use Data Processing in Batch Transform Jobs (p. 352)
- Supported JSONPath Operators (p. 352)
- Examples (p. 353)

Data Processing Workflow for a Batch Transform Job

The following diagram shows the data processing workflow for a batch transform job.
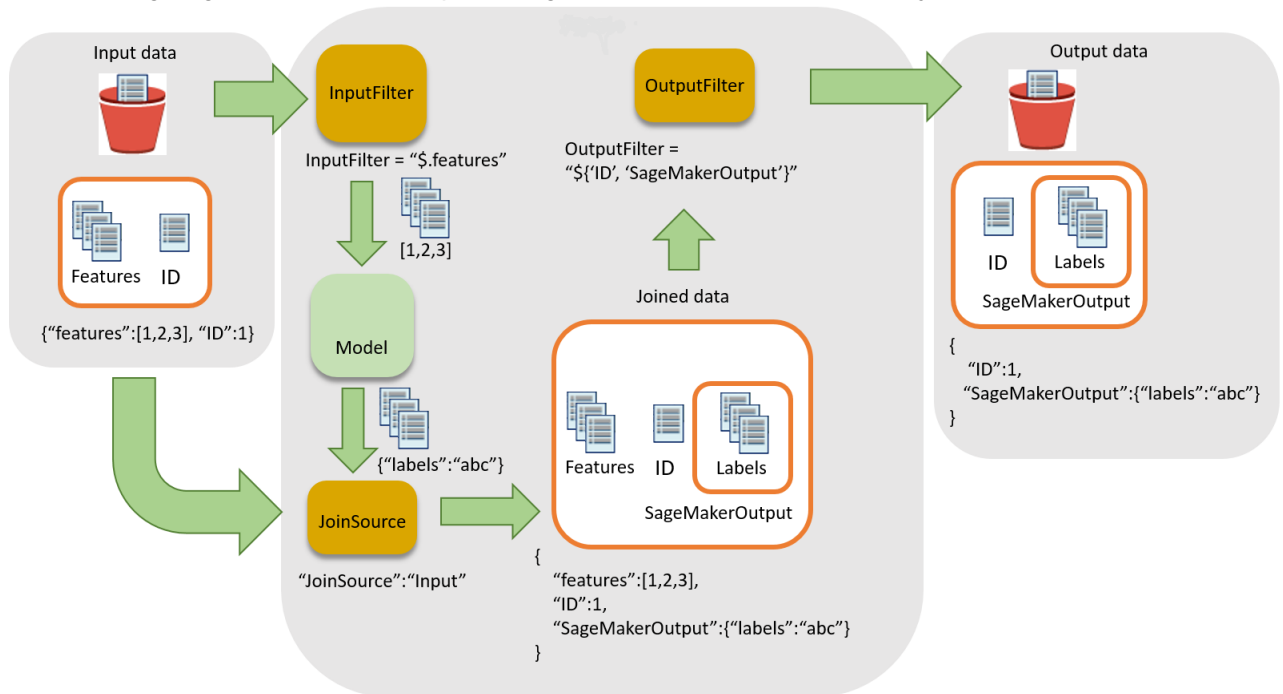
To join the prediction results with the input data, there are three main steps:

- Filter the input data that is not needed for inference before passing it to the batch transform job. Use the InputFilter parameter to determine which attributes to use as input for the model.
- Associate the input data with the inference results. Use JoinSource to combine the input data with the inference.

- Filter the joined data to retain the inputs needed to provide context for interpreting the predictions in the reports. Use OutputFilter to store the specified portion of the joined dataset in the output file.

Use Data Processing in Batch Transform Jobs

To process the data when creating a batch transform job with CreateTransformJob:

1. Specify the portion of the input to pass to the model with the `InputFilter` parameter in the `DataProcessing` data structure.
2. Join the raw input data with the transformed data with the `JoinSource` parameter.
3. Specify which portion of the joined input and transformed data from the batch transform job to include in the output file with the `OutputFilter` parameter.
4. Choose either JSON- or CSV-formatted files for input:
   - For JSON- or JSON Lines-formatted inputs, Amazon SageMaker either adds `SageMakerOutput` attribute to the input file or creates a new JSON output file with the attributes `SageMakerInput` and `SageMakerOutput`. For more information, see DataProcessing (p. 891).
   - For CSV-formatted input files, the joined input data is followed by the transformed data and the output is a CSV file.

If you use an algorithm with the DataProcessing structure, it must support your chosen format for *both* input and output files. For example, with the TransformOutput (p. 1030) field of the `CreateTransformJob` API, you must set both the ContentType and Accept parameters to one of the following values: `text/csv`, `application/json`, or `application/jsonlines`. The syntax for specifying columns in a CSV file and specifying attributes in a JSON file are different. Using the wrong syntax causes an error. For more information, see Examples (p. 353) For more information about input and output file formats for build-in algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 56).

The record delimiters for the input and output must also be consistent with the your chosen file input. The SplitType parameter indicates how to split the records in the input dataset. The AssembleWith parameter indicates how to reassemble the records for the output. If you set input and output formats to `text/csv`, you must also set the `SplitType` and `AssemblyType` parameters to `line`. If you set the input and output formats to `application/jsonlines`, you can set both `SplitType` and `AssemblyType` to either `none` or `line`.

For JSON files, the attribute name `SageMakerOutput` is reserved for output. The JSON input file can't have an attribute with this name. If it does, the data in the input file might be overwritten.

Supported JSONPath Operators

To filter and join the input data and inference, use a JSONPath subexpression. The following table lists the supported JSONPath operators.

JSONPath Operator	Description	Example
$	The root element to a query. This operator is required at the beginning of all path expressions.	`"$"`
.*<name>*	A dot-notated child element.	`"$.id"`
*	A wildcard. Use in place of an attribute name or numeric value.	`"$.id.*"`
`['<name>' (,'<name>')]`	A bracket-notated element or multiple child elements.	`"$['id','SageMakerOutput']"`

JSONPath Operator	Description	Example
[*<number>* (,*<number>*)]	An index or array of indexes. Negative index values are also supported. A −1 index refers to the last element in an array.	$[1],$[1,3,5]
[*<start>*:*<end>*]	An array slice operator. If you omit *<start>*, Amazon SageMaker uses the first element of the array. If you omit *<end>*, Amazon SageMaker uses the last element of the array.	$[2:5],$[:5],$[2:]

Note
Amazon SageMaker supports only a subset of the defined JSONPath operators. For more information about JSONPath operators, see JsonPath.

Examples

The following examples show some common ways to join input data with prediction results.

Topics

Output Only Inference Results

By default, the DataProcessing parameter doesn't join results with input and only outputs the inference results.

If you want to explicity specify in code not to join results with input, use the Amazon SageMaker Python SDK and specify these settings in a transformer call.

```
sm_transformer = sagemaker.transformer.Transformer(…)
sm_transformer.transform(…, input_filter="$", join_source= "None", output_filter="$")
```

The following code shows the default behavior. To output an inference only using the AWS SDK for Python, add it to your CreateTransformJob request:

```
{
    "DataProcessing": {
        "InputFilter": "$",
        "JoinSource": "None",
        "OutputFilter": "$"
    }
}
```

Output a Combination of Input Data and Results

If you are using the Amazon SageMaker Python SDK, combine the input data with the inference in the output file, specify "Input" for the JoinSource parameter in a transformer call.

```
sm_transformer = sagemaker.transformer.Transformer(…)
```

```
sm_transformer.transform(…, join_source= "Input")
```

If you are using the AWS SDK for Python (Boto 3), join all input data with the inference by adding the following code to your CreateTransformJob (p. 673) request.

```
{
    "DataProcessing":
    {
        "JoinSource": "Input"
    }
}
```

For JSON or JSON Lines, the results are in the `SageMakerOutput` key in the input JSON file. For example, if the input is a JSON file that contains the key-value pair `{"key":1}`, the the data transform result might be `{"label":1}`.

Amazon SageMaker stores both in the input file under `SageMakerInput` key.

```
{
    "key":1,
    "SageMakerOutput":{"label":1}
}
```

> **Note**
> The joined result for JSON must be a key-value pair object. If the input is not a key-value pair object, Amazon SageMaker creates a new JSON file. In the new JSON file, the input data is stored in the `SageMakerInput` key and the results are stored as the `SageMakerOutput` value.

For a CSV file, for example, if the record is `[1,2,3]`, and the label result is `[1]`, then the output file would contain `[1,2,3,1]`.

Output an ID Column with Results and Exclude the ID Column from the Input (CSV)

If you are using the Amazon SageMaker Python SDK, to include results or an ID column in the output, specify indexes of the joined dataset in a transformer call. For example, if your data includes five columns an the first one is the ID column, use the following transformer request.

```
sm_transformer = sagemaker.transformer.Transformer(…)
sm_transformer.transform(…, input_filter="$[1:]", join_source= "Input",
 output_filter="$[0,5:]")
```

If you are using the AWS SDK for Python (Boto 3), add the following code to your CreateTransformJob (p. 673) request.

```
{
    "DataProcessing": {
        "InputFilter": "$[1:]",
        "JoinSource": "Input",
        "OutputFilter": "$[0,5:]"
    }
}
```

To specify columns in Amazon SageMaker, index the array elements. The first column is 0, the second column is 1, and the sixth column is 5. To exclude the first column from the input, set InputFilter to `"$[1:]"`.

The OutputFilter parameter applies to the joined input and output. To index correctly, you must know the sizes of the input data combined with the inference. To include the first three columns from the input data with the output, set the OutputFilter to "$[0:2, 5:]". The colon ':' tells Amazon SageMaker to include all of the elements between two values. For example, 0:2 specifies the first three columns. If you omit the number after the colon, for example, "[0, 5:]", the subset ends at the last column in the joined data.

Output an ID Attribute with Results and Exclude the ID Attribute from the Input (JSON)

If you are using the Amazon SageMaker Python SDK, include results or an ID attribute in the output by specifying it in a transformer call. For example, if you store data under the features attribute and the record ID under the ID attribute, you would use the following transformer request.

```
sm_transformer = sagemaker.transformer.Transformer(…)
sm_transformer.transform(…, input_filter="$.features", join_source= "Input",
 output_filter="$['id','SageMakerOutput']")
```

If you are using the AWS SDK for Python (Boto 3), join all input data with the inference by adding the following code to your CreateTransformJob (p. 673) request.

```
{
    "DataProcessing": {
        "InputFilter": "$.features",
        "JoinSource": "Input",
        "OutputFilter": "$['id','SageMakerOutput']"
    }
}
```

Warning
The attribute name SageMakerOutput is reserved for the JSON output file. The JSON input file must not have an attribute with this name. If it does, the input file values might be overwritten with the inference.

Amazon SageMaker Elastic Inference (EI)

By using Amazon Elastic Inference (EI), you can speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as Amazon SageMaker hosted models, but at a fraction of the cost of using a GPU instance for your endpoint. EI allows you to add inference acceleration to a hosted endpoint for a fraction of the cost of using a full GPU instance. Add an EI accelerator in one of the available sizes to a deployable model in addition to a CPU instance type, and then add that model as a production variant to an endpoint configuration that you use to deploy a hosted endpoint. You can also add an EI accelerator to a Amazon SageMaker notebook instance so that you can test and evaluate inference performance when you are building your models.

Elastic Inference is supported in EI-enabled versions of TensorFlow and MXNet. To use any other deep learning framework, export your model by using ONNX, and then import your model into MXNet. You can then use your model with EI as an MXNet model. For information about importing an ONNX model into MXNet, see https://mxnet.incubator.apache.org/tutorials/onnx/super_resolution.html.

Topics
- How EI Works (p. 356)
- Choose an EI Accelerator Type (p. 356)

How EI Works

EI accelerators are network attached devices that work along with EC2 instances in your endpoint to accelerate your inference calls. When your model is deployed as an endpoint, ML frameworks use a combination of EC2 instance and accelerator resources to execute inference calls.

The following EI accelerator types are available. You can configure your endpoints or notebook instances with any EI accelerator type.

In the table, the throughput in teraflops (TFLOPS) is listed for both single-precision floating-point (F32) and half-precision floating-point (F16) operations. The memory in GB is also listed.

Accelerator Type	F32 Throughput in TFLOPS	F16 Throughput in TFLOPS	Memory in GB
ml.eia1.medium	1	8	1
ml.eia1.large	2	16	2
ml.eia1.xlarge	4	32	4

Choose an EI Accelerator Type

Consider the following factors when choosing an accelerator type for a hosted model:

- Models, input tensors and batch sizes influence the amount of accelerator memory you need. Start with an accelerator type that provides at least as much memory as the file size of your trained model.
- Demands on CPU compute resources, GPU-based acceleration, and CPU memory vary significantly between different kinds of deep learning models. The latency and throughput requirements of the application also determine the amount of compute and acceleration you need. Thoroughly test different configurations of instance types and EI accelerator sizes to make sure you choose the configuration that best fits the performance needs of your application.

Use EI in a Amazon SageMaker Notebook Instance

Typically, you build and test machine learning models in a Amazon SageMaker notebook before you deploy them for production. You can attach EI to your notebook instance when you create the notebook instance. You can set up an endpoint that is hosted locally on the notebook instance by using the local mode supported by TensorFlow and MXNet estimators and models in the Amazon SageMaker Python SDK to test inference performance. For instructions on how to attach EI to a notebook instance and set up a local endpoint for inference, see Attach EI to a Notebook Instance (p. 361).

Use EI on a Hosted Endpoint

When you are ready to deploy your model for production to provide inferences, you create a Amazon SageMaker hosted endpoint. You can attach EI to the instance where your endpoint is hosted to increase its performance at providing inferences. For instructions on how to attach EI to a hosted endpoint instance, see Use EI on Amazon SageMaker Hosted Endpoints (p. 363).

Frameworks that Support EI

EI is designed to be used with AWS enhanced versions of TensorFlow or Apache MXNet machine learning frameworks. These enhanced versions of the frameworks are automatically built into containers when you use the Amazon SageMaker Python SDK, or you can download them as binary files and import them in your own Docker containers. You can download the EI-enabled binary for TensorFlow from the Amazon S3 bucket at https://s3.console.aws.amazon.com/s3/buckets/amazonei-tensorflow. For information about building a container that uses the EI-enabled version of TensorFlow, see https://github.com/aws/sagemaker-tensorflow-container#building-the-sagemaker-elastic-inference-tensorflow-serving-container. You can download the EI-enabled binary for Apache MXNet from the public Amazon S3 bucket at https://s3.console.aws.amazon.com/s3/buckets/amazonei-apachemxnet. For information about buidling a container that uses the EI-enabled version of MXNet, see https://github.com/aws/sagemaker-mxnet-container#building-the-sagemaker-elastic-inference-mxnet-container.

To use EI in a hosted endpoint, you can use any of the following, depending on your needs.

- SageMaker Python SDK TensorFlow - if you want to use TensorFlow and you don't need to build a custom container.
- SageMaker Python SDK MXNet - if you want to use MXNet and you don't need to build a custom container.
- The low-level AWS Amazon SageMaker SDK for Python (Boto 3) - if you need to build a custom container.

Typically, you don't need to create a custom container unless your model is very complex and requires extensions to a framework that the Amazon SageMaker pre-built containers do not support.

Use EI with Amazon SageMaker Built-in Algorithms

Currently, the Image Classification Algorithm (p. 108) and Object Detection Algorithm (p. 199) built-in algorithms support EI. For an example that uses the Image Classification algorithm with EI, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/imageclassification_caltech/Image-classification-fulltraining.ipynb.

EI Sample Notebooks

The following Sample notebooks provide examples of using EI in Amazon SageMaker:

- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference_local.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference.ipynb

- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference_local.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_serving_using_elastic_inference_with_your_own_model/tensorflow_serving_pretrained_model_elastic_inference.ipynb

Set Up to Use EI

Use the instructions in this topic only if one of the following applies to you:

- You want to use a customized role or permission policy.
- You want to use a VPC for your hosted model or notebook instance.

> **Note**
> If you already have an execution role that has the `AmazonSageMakerFullAccess` managed policy attached (this is true for any IAM role that you create when you create a notebook instance, training job, or model in the console) and you are not connecting to an EI model or notebook instance in a VPC, you do not need to make any of these changes to use EI in Amazon SageMaker.

Topics

Set Up Required Permissions

To use EI in Amazon SageMaker, the role that you use to open a notebook instance or create a deployable model must have a policy with the required permissions attached. You can attach the `AmazonSageMakerFullAccess` managed policy, which contains the required permissions, to the role, or you can add a custom policy that has the required permissions. For information about creating an IAM role, see Creating a Role for an AWS Service (Console) in the *AWS Identity and Access Management User Guide*. For information about attaching a policy to a role, see Adding and Removing IAM Policies .

Add these permissions specifically for connecting EI in an IAM policy:

```
{
    "Effect": "Allow",
    "Action": [
        "elastic-inference:Connect",
        "ec2:DescribeVpcEndpoints"
    ],
    "Resource": "*"
}
```

The following IAM policy is the complete list of required permissions to use EI in Amazon SageMaker:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elastic-inference:Connect",
                "ec2:DescribeVpcEndpoints"
```

```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "cloudwatch:PutMetricData",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeleteScheduledAction",
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
                "application-autoscaling:DescribeScheduledActions",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:PutScheduledAction",
                "application-autoscaling:RegisterScalableTarget",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:GetBucketLocation",
```

```
                "s3:ListBucket",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "s3:ExistingObjectTag/SageMaker": "true"
                }
            }
        },
        {
            "Action": "iam:CreateServiceLinkedRole",
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/aws-service-role/sagemaker.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "sagemaker.amazonaws.com"
                }
            }
        }
    ]
}
```

Use a Custom VPC to Connect to EI

To use EI with Amazon SageMaker in a VPC, you need to create and configure two security groups, and set up a PrivateLink VPC interface endpoint. EI uses VPC interface endpoint to communicate with Amazon SageMaker endpoints in your VPC. The security groups you create are used to connect to the VPC interface endpoint.

Set up Security Groups to Connect to EI

To use EI within a VPC, you need to create two security groups:

- A security group to control access to the VPC interface endpoint that you will set up for EI.
- A security group that allows Amazon SageMaker to call into the first security group.

Complete the following steps to configure the two security groups:

1. Create a security group with no outbound connections. You will attach this to the VPC endpoint interface you create in the next section.

2. Create a second security group with no inbound connections, but with an outbound connection to the first security group.

3. Edit the first security group to allow inbound connections only to the second security group an all outbound connections.

For more information about VPC security groups, see Security Groups for Your VPC in the *Amazon Virtual Private Cloud User Guide*.

Set up a VPC Interface Endpoint to Connect to EI

To use EI with Amazon SageMaker in a custom VPC, you need to set up a VPC interface endpoint (PrivateLink) for the EI service.

- Set up a VPC interface endpoint (PrivateLink) for the EI. Follow the instructions at Creating an Interface Endpoint. In the list of services, choose **com.amazonaws.<region>.elastic-inference.runtime**. For **Security group**, make sure you select the first security group you created in the previous section to the endpoint.

- When you set up the interface endpoint, choose all of the Availability Zones where EI is available. EI fails if you do not set up at least two Availability Zones. For information about VPC subnets, see VPCs and Subnets.

Attach EI to a Notebook Instance

To test and evaluate inference performance using EI, you can attach EI to a notebook instance when you create or update a notebook instance. You can then use EI in local mode to host a model at an endpoint hosted on the notebook instance. You should test various sizes of notebook instances and EI accelerators to evaluate the configuration that works best for your use case.

Set Up to Use EI

To use EI locally in a notebook instance, create a notebook instance with an EI instance. To do this:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker

2. In the navigation pane, choose **Notebook instances**.

3. Choose **Create notebook instance**.

4. For **Notebook instance name**, provide a unique name for your notebook instance.

5. For **notebook instance type**, choose a CPU instance such as ml.t2.medium.

6. For **Elastic Inference (EI)**, choose an instance from the list, such as **ml.eia1.medium**.

7. For **IAM role**, choose an IAM role that has the required permissions to use Amazon SageMaker and EI.

8. (Optional) For **VPC - Optional**, if you want the notebook instance to use a VPC, choose one from the available list, otherwise leave it as **No VPC**. If you use a VPC follow the instructions at Use a Custom VPC to Connect to EI (p. 360).

9. (Optional) For **Lifecycle configuration - optional**, either leave it as **No configuration** or choose a lifecycle configuration. For more information, see Customize a Notebook Instance (p. 40).

10. (Optional) For **Encryption key - optional**, Optional) If you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.

11. (Optional) For **Volume Size In GB - optional**, leave the default value of 5.

12. (Optional) For **Tags**, add tags to the notebook instance. A tag is a label you assign to help manage your notebook instances. A tag consists of a key and a value both of which you define.

13. Choose **Create Notebook Instance**.

After you create your notebook instance with EI attached, you can create a Jupyter notebook and set up an EI endpoint that is hosted locally on the notebook instance.

Topics

- Use EI in Local Mode in Amazon SageMaker (p. 362)

Use EI in Local Mode in Amazon SageMaker

To use EI locally in an endpoint hosted on a notebook instance, use local mode with the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet estimators or models. For more information about local mode support in the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview.

Topics

- Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models (p. 362)
- Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models (p. 362)

Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models

To use EI with TensorFlow in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator or a model object. For more information about Amazon SageMaker Python SDK TensorFlow estimators and models, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

The following code shows how to use local mode with an estimator object. To call the `deploy` method, you must have previously either:

- Trained the model by calling the `fit` method of an estimator.
- Pass a model artifact when you initialize the model object.

```
# Deploys the model to a local endpoint
tf_predictor = tf_model.deploy(initial_instance_count=1,
                               instance_type='local',
                               accelerator_type='local_sagemaker_notebook')
```

For a complete example of using EI in local mode with TensorFlow, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference_local.ipynb

Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models

To use EI with MXNet in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator or a model object. For more information about Amazon SageMaker Python SDK MXNet estimators and models, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst.

The following code shows how to use local mode with an estimator object. You must have previously called the `fit` method of the estimator to train the model.

```
# Deploys the model to a local endpoint
mxnet_predictor = mxnet_estimator.deploy(initial_instance_count=1,
                                          instance_type='local',
                                          accelerator_type='local_sagemaker_notebook')
```

For a complete example of using EI in local mode with MXNet, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference_local.ipynb .

Use EI on Amazon SageMaker Hosted Endpoints

To use Elastic Inference (EI) in Amazon SageMaker with a hosted endpoint for real-time inference, specify an EI accelerator when you create the deployable model to be hosted at that endpoint. You can do this in one of the following ways:

- Use the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet and the Amazon SageMaker pre-built containers for TensorFlow and MXNet
- Build your own container, and use the low-level Amazon SageMaker API (Boto 3). You will need to import the EI-enabled version of either TensorFlow or MXNet from the provided Amazon S3 locations into your container, and use one of those versions to write your training script.
- Use either the Image Classification Algorithm (p. 108) or Object Detection Algorithm (p. 199) build-in algorithms, and use Boto 3 to run your training job and create your deployable model and hosted endpoint.

Topics
- Use EI with an Amazon SageMaker TensorFlow Container (p. 363)
- Use EI with an Amazon SageMaker MXNet Container (p. 364)
- Use EI with Your Own Container (p. 364)

Use EI with an Amazon SageMaker TensorFlow Container

To use TensorFlow with EI in Amazon SageMaker, you need to call the `deploy` method of either the Estimator or Model objects. You then specify an accelerator type using the accelerator_type input argument. For information on using TensorFlow in the Amazon SageMaker Python SDK, see: https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

Use an Estimator Object

To use an estimator object with EI, include the `accelerator_type` input argument when you use the deploy method. The estimator returns a predictor object which we call its deploy method as shown in the example code:

```
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                             instance_type='ml.m4.xlarge',
                             accelerator_type='ml.eia1.medium')
```

Use a Model Object

To use a model object with EI, include the `accelerator_type` input argument when you use the deploy method. The estimator returns a predictor object which we call its deploy method as shown in the example code:

```
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
                              instance_type='ml.m4.xlarge',
                              accelerator_type='ml.eia1.medium')
```

Use EI with an Amazon SageMaker MXNet Container

To use MXNet with EI in Amazon SageMaker, you need to call the `deploy` method of either the Estimator or Model objects. You then specify an accelerator type using the accelerator_type input argument. For information on using MXNet in the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst

Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

Use an Estimator Object

To use an estimator object with EI, include the `accelerator_type` input argument when you use the deploy method. The estimator returns a predictor object which we call its deploy method as shown in the example code:

```
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                              instance_type='ml.m4.xlarge',
                              accelerator_type='ml.eia1.medium')
```

Use a Model Object

To use a model object with EI, include the `accelerator_type` input argument when you use the deploy method. The estimator returns a predictor object which we call its deploy method as shown in the example code:

```
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
                              instance_type='ml.m4.xlarge',
                              accelerator_type='ml.eia1.medium')
```

For a complete example of using EI with MXNet in Amazon SageMaker, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference.ipynb

Use EI with Your Own Container

To use EI with a model in a custom container that you build, use the low-level Amazon SageMaker SDK for Python (Boto 3). download and import the AWS EI-enabled versions of TensorFlow or Apache MXNet machine learning frameworks, and write your training script using those frameworks.

Import the EI Version of TensorFlow or MXNet into Your Docker Container

To use EI with your own container, you need to import either the Amazon EI TensorFlow Serving library or the Amazon EI Apache MXNet library into your container. The EI-enabled versions of TensorFlow and

MXNet are currently available as binary files stored in Amazon S3 locations. You can download the EI-enabled binary for TensorFlow from the Amazon S3 bucket at https://s3.console.aws.amazon.com/s3/buckets/amazonei-tensorflow. For information about building a container that uses the EI-enabled version of TensorFlow, see https://github.com/aws/sagemaker-tensorflow-container#building-the-sagemaker-elastic-inference-tensorflow-serving-container. You can download the EI-enabled binary for Apache MXNet from the public Amazon S3 bucket at https://s3.console.aws.amazon.com/s3/buckets/amazonei-apachemxnet. For information about building a container that uses the EI-enabled version of MXNet, see https://github.com/aws/sagemaker-mxnet-container#building-the-sagemaker-elastic-inference-mxnet-container.

Create an EI Endpoint with Boto 3

To create an endpoint by using Boto 3, you first create an endpoint configuration. The endpoint configuration specifies one or more models (called production variants) that you want to host at the endpoint. To attach EI to one or more of the production variants hosted at the endpoint, you specify one of the EI instance types as the `AcceleratorType` field for that `ProductionVariant`. You then pass that endpoint configuration when you create the endpoint.

Create an Endpoint Configuration

To use EI, you need to specify an accelerator type in the endpoint configuration:

```
# Create Endpoint Configuration
from time import gmtime, strftime

endpoint_config_name = 'ImageClassificationEndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S",
 gmtime())
print(endpoint_config_name)
create_endpoint_config_response = sagemaker.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.m4.xlarge',
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic',
        'AcceleratorType':'ml.eia1.medium'}])

print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])
```

Create an Endpoint

After you create an endpoint configuration with an accelerator type, you can proceed to create an endpoint.

```
endpoint_name = 'ImageClassificationEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
endpoint_response = sagemaker.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
```

After the endpoint is created you can invoke it using the invoke_endpoint method in a boto3 runtime object as you would any other endpoint.

Automatically Scale Amazon SageMaker Models

Amazon SageMaker supports automatic scaling for production variants. *Automatic scaling* dynamically adjusts the number of instances provisioned for a production variant in response to changes in your

workload. When the workload increases, automatic scaling brings more instances online. When the workload decreases, automatic scaling removes unnecessary instances so that you don't pay for provisioned variant instances that you aren't using.

To use automatic scaling for a production variant, you define and apply a scaling policy that uses Amazon CloudWatch metrics and target values that you assign. Automatic scaling uses the policy to adjust the number of instances up or down in response to actual workloads.

You can use the AWS Management Console to apply a scaling policy based on a predefined metric. A *predefined metric* is defined in an enumeration so that you can specify it by name in code or use it in the AWS Management Console. Alternatively, you can use either the AWS Command Line Interface (AWS CLI) or the Application Auto Scaling API to apply a scaling policy based on a predefined or custom metric. We strongly recommend that you load test your automatic scaling configuration to ensure that it works correctly before using it to manage production traffic.

For information about deploying trained models as endpoints, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26).

Topics
- Automatic Scaling Components (p. 366)
- Before You Begin (p. 368)
- Related Topics (p. 369)
- Configure Automatic Scaling for a Production Variant (p. 369)
- Edit a Scaling Policy (p. 375)
- Delete a Scaling Policy (p. 375)
- Update Endpoints that Use Automatic Scaling (p. 377)
- Load Testing for Production Variant Automatic Scaling (p. 377)
- Best Practices for Configuring Automatic Scaling (p. 378)

Automatic Scaling Components

To adjust the number of instances hosting a production variant, Amazon SageMaker automatic scaling uses a scaling policy . Automatic scaling has the following components:

- Required permissions—Permissions that are required to perform automatic scaling actions.
- A service-linked role—An AWS Identity and Access Management (IAM) role that is linked to a specific AWS service. A service-linked role includes all of the permissions that the service requires to call other AWS services on your behalf. Amazon SageMaker automatic scaling automatically generates this role, `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint`, for you.
- A target metric—The Amazon CloudWatch metric that Amazon SageMaker automatic scaling uses to determine when and how much to scale.
- Minimum and maximum capacity—The minimum and maximum number of instances to use for scaling the variant.
- A cool down period—The amount of time, in seconds, after a scale-in or scale-out activity completes before another scale-out activity can start.

Required Permissions for Automatic Scaling

The `SagemakerFullAccessPolicy` IAM policy has all of the permissions required to perform automatic scaling actions. For more information about Amazon SageMaker IAM roles, see Amazon SageMaker Roles (p. 496).

If you are using a custom permission policy, you must include the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
  "sagemaker:DescribeEndpoint",
  "sagemaker:DescribeEndpointConfig",
  "sagemaker:UpdateEndpointWeightsAndCapacities"
 ],
 "Resource": "*"
}
                {
    "Action": [
        "application-autoscaling:*"
    ],
    "Effect": "Allow",
    "Resource": "*"
}

{
 "Action": "iam:CreateServiceLinkedRole",
 "Effect": "Allow",
 "Resource":
 "arn:aws:iam::*:role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
 "Condition": {
  "StringLike": { "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com" }
 }
}

{
 "Effect": "Allow",
 "Action": [
  "cloudwatch:PutMetricAlarm",
  "cloudwatch:DescribeAlarms",
  "cloudwatch:DeleteAlarms"
 ],
 "Resource": "*"
}
```

Service-Linked Role for Automatic Scaling

A service-linked role is a unique type of IAM role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all of the permissions that the service requires to call other AWS services on your behalf. Automatic scaling uses the `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint` service-linked role. For more information, see Service-Linked Roles for Application Auto Scaling in the *Application Auto Scaling User Guide*.

Target Metric for Automatic Scaling

Amazon SageMaker automatic scaling uses target-tracking scaling policies. You configure the *target-tracking scaling policy* by specifying a predefined or custom metric and a target value for the metric . For more information, see Target Tracking Scaling Policies.

Amazon CloudWatch alarms trigger the scaling policy , which calculate how to adjust scaling based on the metric and target value that you set. The scaling policy adds or removes endpoint instances as required to keep the metric at, or close to, the specified target value. In addition, a target-tracking scaling policy also adjusts to fluctuations in the metric when a workload changes. The policy minimizes rapid fluctuations in the number of available instances for your variant.

For example, a scaling policy that uses the predefined `InvocationsPerInstance` metric with a target value of 70 can keep `InvocationsPerInstance` at, or close to 70.

Minimum and Maximum Capacity for Automatic Scaling

You can specify the maximum number of endpoint instances that Application Auto Scaling manages for the variant. The maximum value must be equal to or greater than the value specified for the minimum number of endpoint instances. Amazon SageMaker automatic scaling does not enforce a limit for this value.

You can also specify the minimum number of instances that Application Auto Scaling manages for the variant. This value must be at least 1, and equal to or less than the value specified for the maximum number of variant instances.

To determine the minimum and maximum number of instances that you need for typical traffic, test your automatic scaling configuration with the expected rate of traffic to your variant.

Cooldown Period for Automatic Scaling

Tune the responsiveness of a target-tracking scaling policy by adding a cooldown period. A *cooldown period* controls when your variant is scaled in and out by blocking subsequent scale-in or scale-out requests until the period expires. This slows the deletion of variant instances for scale-in requests, and the creation of variant instances for scale-out requests. A cooldown period helps to ensure that it doesn't launch or terminate additional instances before the previous scaling activity takes effect. After automatic scaling dynamically scales using a scaling policy, it waits for the cooldown period to complete before resuming scaling activities.

You configure the cooldown period in your automatic scaling policy. You can specify the following cooldown periods:

- A scale-in activity reduces the number of variant instances. A scale-in cooldown period specifies the amount of time, in seconds, after a scale-in activity completes before another scale-in activity can start.
- A scale-out activity increases the number of variant instances. A scale-out cooldown period specifies the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start.

If you don't specify a scale-in or a scale-out cooldown period automatic scaling use the default, which is 300 seconds for each.

If instances are being added or removed too quickly when you test your automatic scaling configuration, consider increasing this value. You can see this behavior if the traffic to your variant has a lot of spikes, or if you have multiple automatic scaling policies defined for a variant.

If instances are not being added quickly enough to address increased traffic, consider decreasing this value.

Before You Begin

Before you can use automatically scaled model deployment, create an Amazon SageMaker model deployment. For more information about deploying a model endpoint, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26).

When automatic scaling adds a new variant instance, it is the same instance class as the one used by the primary instance.

Related Topics

- **What Is Application Auto Scaling?**

Configure Automatic Scaling for a Production Variant

You can configure automatic scaling for a variant with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Topics

- Configure Automatic Scaling for a Production Variant (Console) (p. 369)
- Configure Automatic Scaling for a Production Variant (AWS CLI or the Application Auto Scaling API) (p. 370)

Configure Automatic Scaling for a Production Variant (Console)

To configure automatic scaling for a production variant (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose **Endpoints**.
3. Choose the endpoint that you want to configure.
4. For **Endpoint runtime settings**, choose the variant that you want to configure.
5. For **Endpoint runtime settings**, choose **Configure auto scaling**.

    The **Configure variant automatic scaling** page appears.
6. For **Minimum capacity**, type the minimum number of instances that you want the scaling policy to maintain. At least 1 instance is required.
7. For **Maximum capacity**, type the maximum number of instances that you want the scaling policy to maintain.
8. For the target value, type the average number of invocations per instance per minute for the variant. To determine this value, follow the guidelines in Load Testing (p. 377).

    Application Auto Scaling adds or removes instances to keep the metric close to the value that you specify.
9. For **Scale-in cool down (seconds)** and **Scale-out cool down (seconds)**, type the number seconds for each cool down period. Assuming that the order in the list is based on either most important to less important of first applied to last applied.
10. Select **Disable scale in** to prevent the scaling policy from deleting variant instances if you want to ensure that your variant scales out to address increased traffic, but are not concerned with removing instances to reduce costs when traffic decreases, disable scale-in activities.

    Scale-out activities are always enabled so that the scaling policy can create endpoint instances as needed.
11. Choose **Save**.

This procedure registers a variant as a scalable target with Application Auto Scaling. When you register a variant, Application Auto Scaling performs validation checks to ensure the following:

- The variant exists
- The permissions are sufficient

- You aren't registering a variant with an instance that is a burstable performance instance such as T2

    Note
    Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable
    instances such as T2, because they already allow for increased capacity under increased
    workloads. For information about burstable performance instances, see Amazon EC2 Instance
    Types.

Configure Automatic Scaling for a Production Variant (AWS CLI or the Application Auto Scaling API)

With the AWS CLI or the Application Auto Scaling API, you can configure automatic scaling based on
either a predefined or a custom metric.

Register a Production Variant

To define the scaling limits for the variant, register your variant with Application Auto Scaling.
Application Auto Scaling dynamically scales the number of variant instances.

To register your variant, you can use either the AWS CLI or the Application Auto Scaling API.

When you register a variant, Application Auto Scaling performs validation checks to ensure the
following:

- The variant resource exists
- The permissions are sufficient
- You aren't registering a variant with an instance that is a Burstable Performance Instance such as T2

    Note
    Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable
    instances such as T2, because burstable instances already allow for increased capacity under
    increased workloads. For information about Burstable Performance Instances, see Amazon
    EC2 Instance Types.

Register a Production Variant (AWS CLI)

To register your endpoint, use the `register-scalable-target` AWS CLI command with the following
parameters:

- `--service-namespace`—Set this value to `sagemaker`.
- `--resource-id`—The resource identifier for the production variant. For this parameter, the resource
  type is `endpoint` and the unique identifier is the name of the variant. For example `endpoint/`
  `MyEndpoint/variant/MyVariant`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `--min-capacity`—The minimum number of instances that Application Auto Scaling must manage
  for this endpoint. Set `min-capacity` to at least 1. It must be equal to or less than the value specified
  for `max-capacity`.
- `--max-capacity`—The maximum number of instances that Application Auto Scaling should manage.
  Set `max-capacity` to a minimum of 1, It must be equal to or greater than the value specified for
  `min-capacity`.

Example

The following example shows how to register an endpoint variant named `MyVariant` that is dynamically
scaled to have one to eight instances:

```
aws application-autoscaling register-scalable-target \
    --service-namespace sagemaker \
    --resource-id endpoint/MyEndPoint/variant/MyVariant \
    --scalable-dimension sagemaker:variant:DesiredInstanceCount \
    --min-capacity 1 \
    --max-capacity 8
```

Register a Production Variant (Application Auto Scaling API)

To register your endpoint variant with Application Auto Scaling, use the `RegisterScalableTarget` Application Auto Scaling API action with the following parameters:

- `ServiceNamespace`—Set this value to `sagemaker`.
- `ResourceID`—The resource identifier for the production variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant, for example `endpoint/MyEndPoint/variant/MyVariant`.
- `ScalableDimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `MinCapacity`—The minimum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or less than the value specified for `MaxCapacity`.
- `MaxCapacity`—The maximum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or greater than the value specified for `MinCapacity`.

Example

The following example shows how to register an Amazon SageMaker production variant that is dynamically scaled to use one to eight instances:

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndPoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

Define a Target-Tracking Scaling Policy

To specify the metrics and target values for a scaling policy, you configure a target-tracking scaling policy. You can use either a predefined metric or a custom metric.

Scaling policy configuration is represented by a JSON block. You save your scaling policy configuration as a JSON block in a text file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see `TargetTrackingScalingPolicyConfiguration` in the *Application Auto Scaling API Reference*.

The following options are available for defining a target-tracking scaling policy configuration.

Topics

Use a Predefined Metric

To quickly define a target-tracking scaling policy for a variant, use the `SageMakerVariantInvocationsPerInstance` predefined metric. `SageMakerVariantInvocationsPerInstance` is the average number of times per minute that each instance for a variant is invoked. We strongly recommend using this metric.

To use a predefined metric in a scaling policy, create a target tracking configuration for your policy. In the target tracking configuration, include a `PredefinedMetricSpecification` for the predefined metric and a `TargetValue` for the target value of that metric.

Example

The following example is a typical policy configuration for target-tracking scaling for a variant. In this configuration, we use the `SageMakerVariantInvocationsPerInstance` predefined metric to adjust the number of variant instances so that each instance has a `InvocationsPerInstance` metric of 70.

```
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
    }
}
```

Use a Custom Metric

If you need to define a target-tracking scaling policy that meets your custom requirements, define a custom metric. You can define a custom metric based on any production variant metric that changes in proportion to scaling.

Not all Amazon SageMaker metrics work for target tracking. The metric must be a valid utilization metric, and it must describe how busy an instance is. The value of the metric must increase or decrease in inverse proportion to the number of variant instances. That is, the value of the metric should decrease when the number of instances increases.

> **Important**
> Before deploying automatic scaling in production, you must test automatic scaling with your custom metric.

Example

The following example is a target-tracking configuration for a scaling policy. In this configuration, for a variant named `my-variant`, a custom metric adjusts the variant based on an average CPU utilization of 50 percent across all instances.

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
```

```
        "MetricName": "CPUUtilization",
        "Namespace": "/aws/sagemaker/Endpoints",
        "Dimensions": [
            {"Name": "EndpointName", "Value": "my-endpoint" },
            {"Name": "VariantName","Value": "my-variant"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

Add a Cooldown Period

To add a cooldown period for scaling out your variant, specify a value, in seconds, for `ScaleOutCooldown` . Similarly, to add a cooldown period for scaling in your variant, add a value, in seconds, for `ScaleInCooldown` . For more information about `ScaleInCooldown` and `ScaleOutCooldown`, see `TargetTrackingScalingPolicyConfiguration` in the *Application Auto Scaling API Reference*.

Example

The following is an example of a target-tracking policy configuration for a scaling policy. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined metric is used to adjust a variant based on an average of 70 across all instances of that variant. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

```
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

Disable Scale-in Activity

You can prevent the target-tracking scaling policy configuration from scaling in your variant by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting instances, while still allowing it to create them as needed.

To enable or disable scale-in activity for your variant, specify a Boolean value for `DisableScaleIn`. For more information about `DisableScaleIn`, see `TargetTrackingScalingPolicyConfiguration` in the *Application Auto Scaling API Reference*.

Example

The following is an example of a target-tracking configuration for a scaling policy. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined metric adjusts a variant based on an average of 70 across all instances of that variant. The configuration disables scale-in activity for the scaling policy.

```
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
    },
```

```
        "DisableScaleIn": true
}
```

Apply a Scaling Policy to a Production Variant

After registering your variant and defining a scaling policy, apply the scaling policy to the registered variant. To apply a scaling policy to a variant, you can use the AWS CLI or the Application Auto Scaling API.

Apply a Scaling Policy to a Production Variant (AWS CLI)

To apply a scaling policy to your variant, use the `put-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name`—The name of the scaling policy.
- `--policy-type`—Set this value to `TargetTrackingScaling`.
- `--resource-id`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example `endpoint/MyEndpoint/variant/MyVariant`.
- `--service-namespace`—Set this value to `sagemaker`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `--target-tracking-scaling-policy-configuration`—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses with Application Auto Scaling to apply a target-tracking scaling policy named `myscalablepolicy` to a variant named `myscalablevariant`. The policy configuration is saved in a file named `config.json`.

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id endpoint/MyEndpoint/variant/MyVariant \
    --service-namespace sagemaker \
    --scalable-dimension sagemaker:variant:DesiredInstanceCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

Apply a Scaling Policy to a Production Variant (Application Auto Scaling API)

To apply a scaling policy to a variant with the Application Auto Scaling API, use the `PutScalingPolicy` Application Auto Scaling API action with the following parameters:

- `PolicyName`—The name of the scaling policy.
- `ServiceNamespace`—Set this value to `sagemaker`.
- `ResourceID`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- `ScalableDimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `PolicyType`—Set this value to `TargetTrackingScaling`.
- `TargetTrackingScalingPolicyConfiguration`—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses Application Auto Scaling to apply a target-tracking scaling policy named `myscalablepolicy` to a variant named `myscalablevariant`. It uses a policy configuration based on the `SageMakerVariantInvocationsPerInstance` predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 70.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
        }
    }
}
```

Edit a Scaling Policy

You can edit a variant scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Edit a Scaling Policy (Console)

To edit a scaling policy with the AWS Management Console, use the same procedure that you used to Configure Automatic Scaling for a Production Variant (Console) (p. 369).

Edit a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- With the AWS CLI, specify the name of the policy that you want to edit in the `--policy-name` parameter. Specify new values for the parameters that you want to change.
- With the Application Auto Scaling API, specify the name of the policy that you want to edit in the `PolicyName` parameter. Specify new values for the parameters that you want to change.

For more information, see Apply a Scaling Policy to a Production Variant (p. 374).

Delete a Scaling Policy

You can delete a scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Delete a Scaling Policy (Console)

To delete an automatic scaling policy for a variant (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose **Endpoints**.
3. Choose the endpoint for which you want to delete automatic scaling.
4. For **Endpoint runtime settings**, choose the variant that you want to configure.
5. Choose **Configure auto scaling**.
6. Choose **Deregister auto scaling**.

Delete a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to delete a scaling policy from a variant.

Delete a Scaling Policy (AWS CLI)

To delete a scaling policy from a variant, use the `delete-scaling-policy` AWS CLI command with the following parameters:

*   `--policy-name`—The name of the scaling policy.
*   `--resource-id`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
*   `--service-namespace`—Set this value to `sagemaker`.
*   `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.

Example

The following example deletes a target-tracking scaling policy named `myscalablepolicy` from a variant named `myscalablevariant`.

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id endpoint/MyEndpoint/variant/MyVariant \
    --service-namespace sagemaker \
    --scalable-dimension sagemaker:variant:DesiredInstanceCount
```

Delete a Scaling Policy (Application Auto Scaling API)

To delete a scaling policy from your variant, use the `DeleteScalingPolicy` Application Auto Scaling API action with the following parameters:

*   `PolicyName`—The name of the scaling policy.
*   `ServiceNamespace`—Set this value to `sagemaker`.
*   `ResourceID`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant,. For example, `endpoint/MyEndpoint/variant/MyVariant`.
*   `ScalableDimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.

Example

The following example uses the Application Auto Scaling API to delete a target-tracking scaling policy named `myscalablepolicy` from a variant named `myscalablevariant`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount"
}
```

Update Endpoints that Use Automatic Scaling

When you update Amazon SageMaker endpoints that have automatic scaling applied, complete the following steps:

To update an endpoint that has automatic scaling applied

1. Deregister the endpoint as a scalable target by calling DeregisterScalableTarget.
2. Because you turn off automatic scaling before you update the endpoint, you might want to take the additional precaution of increasing the number of instances for your endpoint during the update. To do this, update the instance counts for the production variants hosted at the endpoint by calling UpdateEndpointWeightsAndCapacities (p. 842).
3. Call DescribeEndpoint (p. 709) repeatedly until the value of the `EndpointStatus` field of the response is `InService`.
4. Call DescribeEndpointConfig (p. 712) to get the values of the current endpoint config.
5. Create a new endpoint config by calling CreateEndpointConfig (p. 635). For the `InitialInstanceCount` field of each production variant, specify the corresponding value of `DesiredInstanceCount` from the response to the previous call to DescribeEndpoint (p. 709). For all other values, use the values that you got as the response when you called DescribeEndpointConfig (p. 712) in the previous step.
6. Update the endpoint by calling UpdateEndpoint (p. 840). Specify the endpoint config you created in the previous step as the `EndpointConfig` field.
7. Re-enable automatic scaling by calling RegisterScalableTarget.

Load Testing for Production Variant Automatic Scaling

Perform load tests to choose an automatic scaling configuration that works the way you want.

For an example of load testing to optimize automatic scaling for a Amazon SageMaker endpoint, see Load test and optimize an Amazon SageMaker endpoint using automatic scaling.

The following guidelines for load testing assume you are using an automatic scaling policy that uses the predefined target metric `SageMakerVariantInvocationsPerInstance`.

Topics

Determine the Performance Characteristics of a Production Variant

Perform load testing to find the peak `InvocationsPerInstance` that your variant instance can handle, and the latency of requests, as concurrency increases.

This value depends on the instance type chosen, payloads that clients of your variant typically send, and the performance of any external dependencies your variant has.

To find the peak requests-per-second (RPS) your variant can handle and latency of requests

1. Set up an endpoint with your variant using a single instance. For information about how to set up an endpoint, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26).
2. Use a load testing tool to generate an increasing number of parallel requests, and monitor the RPS and model latency in the out put of the load testing tool.

   Note
   You can also monitor requests-per-minute instead of RPS. In that case don't multiply by 60 in the equation to calculate `SageMakerVariantInvocationsPerInstance` shown below.

   When the model latency increases or the proportion of successful transactions decreases, this is the peak RPS that your variant can handle.

Calculate the Target SageMakerVariantInvocationsPerInstance

After you find the performance characteristics of the variant, you can determine the maximum RPS we should allow to be sent to an instance. The threshold used for scaling must be less than this maximum value. Use the following equation in combination with load testing to determine the correct value for the `SageMakerVariantInvocationsPerInstance` target metric in your automatic scaling configuration.

```
SageMakerVariantInvocationsPerInstance = (MAX_RPS * SAFETY_FACTOR) * 60
```

Where `MAX_RPS` is the maximum RPS that you determined previously, and `SAFETY_FACTOR` is the safety factor that you chose to ensure that your clients don't exceed the maximum RPS. Multiply by 60 to convert from RPS to invocations-per-minute to match the per-minute CloudWatch metric that Amazon SageMaker uses to implement automatic scaling (you don't need to do this if you measured requests-per-minute instead of requests-per-second).

   Note
   Amazon SageMaker recommends that you start testing with a `SAFETY_FACTOR` of 0.5. Test your automatic scaling configuration to ensure it operates in the way you expect with your model for both increasing and decreasing customer traffic on your endpoint.

Best Practices for Configuring Automatic Scaling

When configuring automatic scaling, consider the following general guidelines.

Testing Your Automatic Scaling Configuration

It is important that you test your automatic scaling configuration to confirm that it works with your model the way you expect it to.

Updating Endpoints Configured for Automatic Scaling

When you update an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If the update would change the instance type for any variant that is a target for automatic scaling, the update fails.

In the AWS Management Console, you see a warning that you must deregister the variant from automatic scaling before you can update it. If you are trying to update the endpoint by calling the UpdateEndpoint (p. 840) API, the call fails. Before you update the endpoint, delete any scaling policies configured for it by calling the DeleteScalingPolicy Application Auto Scaling API action, then call DeregisterScalableTarget to deregister the variant as a scalable target. After you update the endpoint, you can register the variant as a scalable target and attach an automatic scaling policy to the updated variant.

There is one exception. If you change the model for a variant that is configured for automatic scaling, Amazon SageMaker automatic scaling allows the update. This is because changing the model doesn't typically affect performance enough to change automatic scaling behavior. If you do update a model for a variant configured for automatic scaling, ensure that the change to the model doesn't significantly affect performance and automatic scaling behavior.

For instructions on how to update an endpoint that uses automatic scaling, see Update Endpoints that Use Automatic Scaling (p. 377).

Deleting Endpoints Configured for Automatic Scaling

If you delete an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If any are and you have permission to deregister the variant, Application Auto Scaling deregisters those variants as scalable targets without notifying you. If you use a custom permission policy that doesn't provide permission for the DeleteScalingPolicy and DeregisterScalableTarget actions, you must delete automatic scaling policies and deregister scalable targets and before deleting the endpoint.

> **Note**
> You, as an IAM user, might not have sufficient permission to delete an endpoint if another IAM user configured automatic scaling for a variant on that endpoint.

Using Step Scaling Policies

Although Amazon SageMaker automatic scaling supports using Application Auto Scaling step scaling policies, we recommend using target tracking policies, instead. For information about using Application Auto Scaling step scaling policies, see Step Scaling Policies.

Scaling In When There Is No Traffic

If a variant's traffic becomes zero, Amazon SageMaker automatic scaling doesn't scale down. This is because Amazon SageMaker doesn't emit metrics with a value of zero.

As a workaround, do either of the following:

* Send requests to the variant until automatic scaling scales down to the minimum capacity

- Change the policy to reduce the maximum provisioned capacity to match the minimum provisioned capacity

Troubleshoot Amazon SageMaker Model Deployments

If you encounter an issue when deploying machine learning models in Amazon SageMaker, see the following guidance.

Topics

Detection Errors in the Active CPU Count

If you deploy an Amazon SageMaker model with a Linux Java Virtual Machine (JVM), you might encounter detection errors that prevent using available CPU resources. This issue affects some JVMs that support Java 8 and Java 9, and most that support Java 10 and Java 11. These JVMs implement a mechanism that detects and handles the CPU count and the maximum memory available when running a model in a Docker container, and, more generally, within Linux `taskset` commands or control groups (cgroups). Amazon SageMaker deployments take advantage of some of the settings that the JVM uses for managing these resources. Currently, this causes the container to incorrectly detect the number of available CPUs.

Amazon SageMaker doesn't limit access to CPUs on an instance. However, the JVM might detect the CPU count as `1` when more CPUs are available for the container. As a result, the JVM adjusts all of its internal settings to run as if only `1` CPU core is available. These settings affect garbage collection, locks, compiler threads, and other JVM internals that negatively affect the concurrency, throughput, and latency of the container.

For an example of the misdetection, in a container configured for Amazon SageMaker that is deployed with a JVM that is based on Java8_191 and that has four available CPUs on the instance, run the following command to start your JVM:

```
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
```

This generates the following output:

```
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Many of the JVMs affected by this issue have an option to disable this behavior and reestablish full access to all of the CPUs on the instance. Disable the unwanted behaviour and establish full access to all instance CPUs by including the `-XX:-UseContainerSupport` parameter when starting Java applications. For example, run the `java` command to start your JVM as follows:

```
java -XX:-UseContainerSupport -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
```

This generates the following output:

```
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: sched_getaffinity processor count: 4
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Check whether the JVM used in your container supports the `-XX:-UseContainerSupport` parameter. If it does, always pass the parameter when you start your JVM. This provides access to all of the CPUs in your instances.

You might also encounter this issue when indirectly using a JVM in Amazon SageMaker containers. For example, when using a JVM to support SparkML Scala. The `-XX:-UseContainerSupport` parameter also affects the output returned by the Java `Runtime.getRuntime().availableProcessors()` API.

Best Practices for Deploying Amazon SageMaker Models

This topic provides guidance on best practices for deploying machine learning models in Amazon SageMaker.

Topics

Deploy Multiple Instances Across Avalibility Zones

Create robust endpoints when hosting your model. Amazon SageMaker endpoints can help protect your application from Availability Zone outages and instance failures. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones. For this reason, we strongly recommended that you deploy multiple instances for each production endpoint.

If you are using an Amazon Virtual Private Cloud (VPC), configure the VPC with at least two Subnets, each in a different Availability Zone. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones.

In general, to achieve more reliable performance, use more small Instance Types in different Availability Zones to host your endpoints.

Hosting Instance Storage Volumes

When you create an endpoint, Amazon SageMaker attaches an Amazon EBS storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type. The following table shows the size of the storage volume that Amazon SageMaker attaches for each instance type.

Instance Type	Storage Volume in GB
ml.t2.medium	2
ml.t2.large	4
ml.t2.xlarge	8
ml.t2.2xlarge	16
ml.m4.xlarge	8
ml.m4.2xlarge	16
ml.m4.4xlarge	30
ml.m4.10xlarge	30
ml.m4.16xlarge	30
ml.m5.large	4
ml.m5.xlarge	8
ml.m5.2xlarge	16
ml.m5.4xlarge	30
ml.m5.12xlarge	30
ml.m5.24xlarge	30
ml.c4.large	4
ml.c4.xlarge	4
ml.c4.2xlarge	8
ml.c4.4xlarge	15
ml.c4.8xlarge	30
ml.c5.large	2
ml.c5.xlarge	4
ml.c5.2xlarge	8
ml.c5.4xlarge	16
ml.c5.9xlarge	30
ml.c5.18xlarge	30
ml.p2.xlarge	30
ml.p2.8xlarge	30
ml.p2.16xlarge	30

Instance Type	Storage Volume in GB
ml.g5.2xlarge	300
ml.g5.8xlarge	300
ml.g5.16xlarge	300

Use Your Own Algorithms or Models with Amazon SageMaker

Amazon SageMaker makes extensive use of *Docker containers* for build and runtime tasks. Before using your own algorithm or model with Amazon SageMaker, you need to understand how Amazon SageMaker manages and runs them. Amazon SageMaker provides pre-built Docker images for its built-in algorithms and the deep learning frameworks supported used for training and inference. By using containers, you can train machine learning algorithms and deploy models quickly and reliably at any scale. Docker is a program that performs operating-system-level virtualization for installing, distributing, and managing software. It packages applications and their dependencies into virtual containers that provide isolation, portability, and security.

You can put scripts, algorithms, and inference code for your machine learning models into containers. The container includes the runtime, system tools, system libraries, and other code required to train your algorithms or deploy your models. This gives you the flexibility to use almost any script or algorithm code with Amazon SageMaker, regardless of runtime or implementation language. The code that runs in containers is effectively isolated from its surroundings, ensuring a consistent runtime, regardless of where the container is deployed. After packaging your training code, inference code, or both into Docker containers, you can create algorithm resources and model package resources for use in Amazon SageMaker or to publish on AWS Marketplace. With Docker, you can ship code faster, standardize application operations, seamlessly move code, and economize by improving resource utilization.

You create Docker containers from *images* that are saved in a *repository*. You build the images from scripted instructions provided in a *Dockerfile*. To use Docker containers in Amazon SageMaker, the scripts that you use must satisfy certain requirements. For information about the requirements, see and .

Scenarios for Running Scripts, Training Algorithms, or Deploying Models with Amazon SageMaker

Amazon SageMaker always uses Docker containers when running scripts, training algorithms or deploying models. However, your level of engagement with containers varies depending on whether you are using a built-in algorithm provided by Amazon SageMaker or a script or model that you have developed yourself. If you're using your own code, it also depends on the language and framework or environment used to develop it, and any other the dependencies it requires to run. In particular, it depends on whether you use the Amazon SageMaker Python SDK or AWS SDK for Python (Boto3) or some other SDK. Amazon SageMaker provides containers for its built-in algorithms and pre-built Docker images for some of the most common machine learning frameworks. You can use the containers and images as provided or extend them to cover more complicated use cases. You can also create your own container images to manage more advanced use cases not addressed by the containers provided by Amazon SageMaker.

There are four main scenarios for running scripts, algorithms, and models in the Amazon SageMaker environment. The last three describe the scenarios covered here: the ways you can use containers to *bring your own script, algorithm or model*.

- **Use a built-in algorithm**. Containers are used behind the scenes when you use one of the Amazon SageMaker built-in algorithms, but you do not deal with them directly. You can train and deploy

these algorithms from the Amazon SageMaker console, the AWS Command Line Interface (AWS CLI), a Python notebook, or the Amazon SageMaker Python SDK. The built-in algorithms available are itemized and described in the Use Amazon SageMaker Built-in Algorithms (p. 56) topic. For an example of how to train and deploy a built-in algorithm using Jupyter Notebook running in an Amazon SageMaker notebook instance, see the Get Started (p. 16) topic.

- **Use pre-built container images**.Amazon SageMaker provides pre-built containers to supports deep learning frameworks such as Apache MXNet, TensorFlow, PyTorch, and Chainer. It also supports machine learning libraries such a scikit-learn and SparkML by providing pre-built Docker images. If you use the Amazon SageMaker Python SDK, they are deployed using their respective Amazon SageMaker SDK `Estimator` class. In this case, you can supply the Python code that implements your algorithm and configure the pre-built image to access your code as an entry point. For a list of deep learning frameworks currently supported by Amazon SageMaker and samples that show how to use their pre-build container images, see Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 398). For information on the scikit-learn and SparkML pre-built container images, see Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML (p. 401). For more information about using frameworks with the Amazon SageMaker Python SDK, see their respective topics in Use Machine Learning Frameworks with Amazon SageMaker (p. 440).

- **Extend a pre-built container image**. If you have additional functional requirements for an algorithm or model that you developed in a framework that a pre-built Amazon SageMaker Docker image doesn't support, you can modify an Amazon SageMaker image to satisfy your needs. For an example, see Extending our PyTorch containers.

- **Build your own custom container image**: If there is no pre-built Amazon SageMaker container image that you can use or modify for an advanced scenario, you can package your own script or algorithm to use with Amazon SageMaker.You can use any programming language or framework to develop your container. For an example that shows how to build your own containers to train and host an algorithm, see Bring Your Own R Algorithm.

The next topic provides a brief introduction to Docker containers. Amazon SageMaker has certain contractual requirements that a container must satisfy to be used with it. The following topic describes the Amazon SageMaker Containers library that can be used to create Amazon SageMaker-compatible containers, including a list of the environmental variables it defines and may require. Then a tutorial that shows how to get started by using Amazon SageMaker Containers to train a Python script. After the tutorial, topics:

- Describe the pre-built Docker containers provided by Amazon SageMaker for deep learning frameworks and other libraries.

- Provide examples of how to deploy containers for the various scenarios.

Subsequent sections describe in more detail the contractual requirements to use Docker with Amazon SageMaker to train your custom algorithms and to deploy your inference code to make predictions. There are two ways to make predictions when deploying a model. First, to get individual, real-time predictions, you can make inferences with a hosting services. Second, to get predictions for an entire dataset, you can use a batch transform. The final sections describe how to create algorithm and model package resources for use in your Amazon SageMaker account or to publish on AWS Marketplace.

Docker Container Basics

Docker containers provide isolation, portability, and security. They simplify the creation of highly distributed systems and save money by improving resource utilization. Docker relies on Linux kernel functionality to provide a lightweight virtualization to package applications into an image that is totally self-contained. Docker uses a file, called a Dockerfile, to specify how the image is assembled. When you have an image, you use Docker to build and run a container based on that image.

You can build your Docker images from scratch or base them on other Docker images that you or others have built. Images are stored in repositories that are indexed and maintained by registries. An image can be pushed into or pulled out of a repository using its registry address, which is similar to a URL. Docker Hub is a registry hosted by Docker, Inc. that provides publicly available repositories. AWS provides the Amazon Elastic Container Service (Amazon ECS), a highly scalable, fast container management service. With Amazon ECS, you can deploy any kind of code in Amazon SageMaker. You can also create a logical division of labor by creating a deployment team that handles DevOps and infrastructure, and that maintains the container, and a data science team that creates the algorithms and models that are later added to a container.

Docker builds images by reading the instructions from a Dockerfile text file that contains all of the commands, in order, that are needed to build the image. A Dockerfile adheres to a specific format and set of instructions. For move information, see Dockerfile reference. Dockerfiles used in Amazon SageMaker must also satisfy additional requirements regarding the environmental variables, directory structure, timeouts, and other common functionality. For information, see Use Your Own Training Algorithms (p. 404) and Use Your Own Inference Code (p. 408).

For general information about Docker containers managed by Amazon ECS, see Docker Basics for Amazon ECS in the *Amazon Elastic Container Service Developer Guide*.

For more information about writing Dockerfiles to build images, see Best practices for writing Dockerfiles.

For general information about Docker, see the following:

- Docker home page
- Docker overview
- Getting Started with Docker
- Dockerfile reference

Amazon SageMaker Containers: a Library to Create Docker Containers

Amazon SageMaker Containers is a library that implements the functionality that you need to create containers to run scripts, train algorithms, or deploy models that are compatible with Amazon SageMaker. To install this library, use a `RUN pip install sagemaker-containers` command in your Dockerfile. The library defines the locations for storing code and other resources when you install it. Your Dockerfile must also copy the code to be run into the location expected by an Amazon SageMaker-compatible container and define the entry point containing the code to run when the container is started. The library also defines other information that a container needs to manage deployments for training and inference. After you build a Docker image, you can push it to the Amazon Elastic Container Registry (Amazon ECR). To create a container, you can pull the image from Amazon ECR and build the container using the `docker build` command.

The following high-level schematic shows how the files are organized in an Amazon SageMaker-compatible container created with the Amazon SageMaker Containers library.
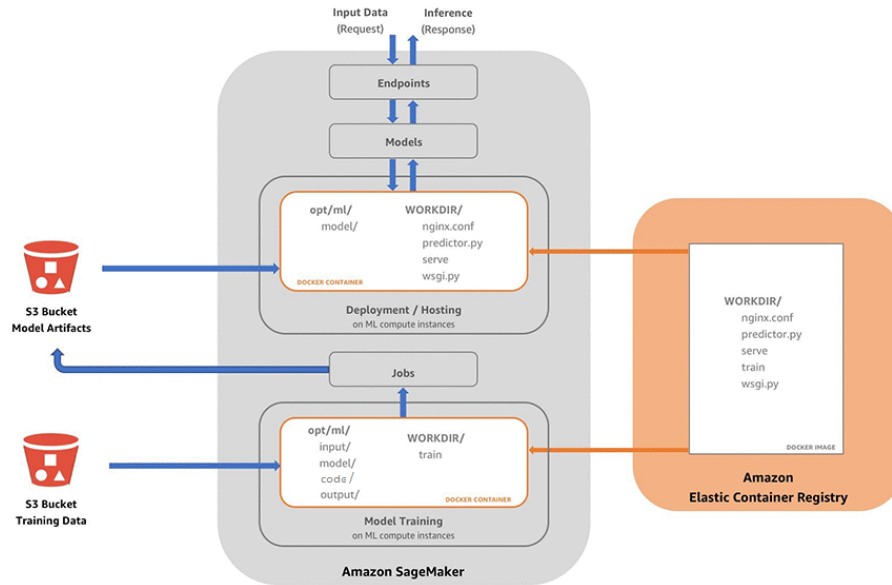
When Amazon SageMaker trains a model, it creates a number of files in the container's `/opt/ml` directory.

```
/opt/ml
### input
#    ### config
#    #    ### hyperparameters.json
#    #    ### resourceConfig.json
#    ### data
#        ### <channel_name>
#            ### <input data>
### model
#
### code
#    ### <script files>
#
### output
    ### failure
```

When you run a model *training* job, the Amazon SageMaker container has a `/opt/ml/input/` directory that contains JSON files that configure the hyperparameters for the algorithm and the network layout used for distributed training. The directory also contains files that specify the channels through which Amazon SageMaker accesses the data in Amazon Simple Storage Service (Amazon S3). Place scripts to run in the `/opt/ml/code/` directory. The `/opt/ml/model/` directory contains the model generated by your algorithm in a singe file or an entire directory tree in any format. You can also send information about why a training job failed to the `/opt/ml/output/` directory. Amazon SageMaker packages files in this directory into a compressed tar archive file.

When you *host* a trained model on Amazon SageMaker to make inferences, you deploy the model to an HTTP endpoint. The model makes realtime predictions in response to inference requests. The container must contain a serving stack to process these requests. The five files used in the standard Python serving stack by Amazon SageMaker are installed in the container's `WORKDIR`. You can choose a different toolset to deploy an HTTP endpoint and, therefore, could have a different layout. If you're writing in a programming language other than Python, you will have a different layout, the nature of which will depend on the frameworks and tools that you choose. The Python serving stack in the `WORKDIR` directory contains the following files:

- **nginx.conf**– The configuration file for the nginx front end.

- **predictor.py**– The program that implements the Flask web server and the decision tree predictions for this application. You need to customize the code that performs prediction for your application.
- **serve** – The program started when the container is started for hosting. This file simply launches the Gunicorn server, which runs multiple instances of the Flask application defined in `predictor.py`.
- **train** – The program that is invoked when you run the container for training. To implement your training algorithm, you modify this program.
- **wsgi.py** – A small wrapper used to invoke the Flask application.

In the container, the model files are in the same place that they were written to during training.

```
/opt/ml
### model
    ### <model files>
```

For more information, see Use Your Own Inference Code (p. 408)

You can provide separate Docker images for the training algorithm and inference code, as shown in the figure. Or you can use a single Docker image for both. When creating Docker images for use with Amazon SageMaker, consider the following:

- Providing two Docker images can increase storage requirements and cost because common libraries might be duplicated.
- In general, smaller containers start faster for both training and hosting. Models train faster and the hosting service can react to increases in traffic by automatically scaling more quickly.
- You might be able to write an inference container that is significantly smaller than the training container. This is especially common when you use GPUs for training, but your inference code is optimized for CPUs.
- Amazon SageMaker requires that Docker containers run without privileged access.
- Docker containers might send messages to the `Stdout` and `Stderr` files. Amazon SageMaker sends these messages to Amazon CloudWatch logs in your AWS account.

Environmental Variables used by Amazon SageMaker Containers to Define Entry Points

When creating a Dockerfile, you must define an entry point that specifies the location of the code to run when the container starts. Amazon SageMaker Containers does this by setting an `ENV` environment variable. The environment variable that you need to set depends on the job you want to do:

- To run a script,specify the `SAGEMAKER_PROGRAM ENV` variable.
- To train an algorithm, specify the `SAGEMAKER_TRAINING_MODULE ENV` variable.
- To host a model, specify the `SAGEMAKER_SERVING_MODULE ENV` variable.

You can use the Amazon SageMaker containers SDK package to set environment variables.

SAGEMAKER_PROGRAM

Train scripts similar to those you would use outside Amazon SageMaker using Amazon SageMaker Script Mode. It supports Python and Shell scripts: Amazon SageMaker uses the Python interpreter for any script with the `.py` suffix.Amazon SageMaker uses the Shell interpreter to execute any other script.

When running a program to specify the entry point for Script Mode, set the **SAGEMAKER_PROGRAM** environmental variable. The script must be located in the `/opt/ml/code` folder.

For example, the container used in the example in Get Started: Use Amazon SageMaker Containers to Run a Python Script (p. 396) sets this `ENV` as follows.

```
ENV SAGEMAKER_PROGRAM train.py
```

The Amazon SageMaker PyTorch container sets the `ENV` variable as follows.

```
ENV SAGEMAKER_PROGRAM cifar10.py
```

In the example, cifar10.py is the program that implements the training algorithm and handles loading the model for inferences. For more information, see the Extending our PyTorch containers notebook.

`SAGEMAKER_TRAINING_MODULE`

When training an algorithm, specify the location of the module that contains the training logic by setting the **`SAGEMAKER_TRAINING_MODULE`** environment variable. An Amazon SageMaker container invokes this module when the container starts training. For example, you set this environment variable in MXNet as follows.

```
ENV SAGEMAKER_TRAINING_MODULE sagemaker_mxnet_container.training:main
```

For TensorFlow, set this `environmant variable` as follows.

```
ENV SAGEMAKER_TRAINING_MODULE sagemaker_tensorflow_container.training:main
```

The code that implements this logic is in Amazon SageMaker Containers.

`SAGEMAKER_SERVING_MODULE`

To locate the module that contains the hosting logic when deploying a model, set the **`SAGEMAKER_SERVING_MODULE`** environmental variable. An Amazon SageMaker container invokes this module when it starts hosting.

```
ENV SAGEMAKER_SERVING_MODULE sagemaker_mxnet_container.serving:main
```

The code that implements this logic is in the: Amazon SageMaker Containers.

Environmental Variables used by Amazon SageMaker Containers Important for Running User Scripts

When you write a script to run in a container, you are likely to use the following build-time environment variables. Amazon SageMaker Containers sets some of these variable values by default.

- **`SM_MODEL_DIR`**

```
SM_MODEL_DIR=/opt/ml/model
```

When the training job finishes, Amazon SageMaker deletes the container, including its file system, except for the files in the `/opt/ml/model` and `/opt/ml/output` folders. Use `/opt/ml/model` to save the model checkpoints. Amazon SageMaker uploads these checkpoints to the default S3 bucket. Examples:

```
# Using it in argparse
```

```
parser.add_argument('model_dir', type=str, default=os.environ['SM_MODEL_DIR'])

# Using it as a variable
model_dir = os.environ['SM_MODEL_DIR']

# Saving checkpoints to the model directory in Chainer
serializers.save_npz(os.path.join(os.environ['SM_MODEL_DIR'], 'model.npz'), model)
```

For more information, see How Amazon SageMaker Processes Training Output.

- **SM_CHANNELS**

```
SM_CHANNELS='["testing","training"]'
```

The SM_CHANNELS environmental variable contains the list of input data channels for the container. When you train a model, you can partition your training data into different logical "channels". Common channels are: training, testing,and evaluation, or images and labels. SM_CHANNELS includes the name of the channels that are in the container as a JSON encoded list.

Examples:

```
import json

# Using it in argparse
parser.add_argument('channel_names', type=int,
 default=json.loads(os.environ['SM_CHANNELS'])))

# Using it as a variable
channel_names = json.loads(os.environ['SM_CHANNELS']))
```

- **SM_CHANNEL_{channel_name}**

```
SM_CHANNEL_TRAINING='/opt/ml/input/data/training'
SM_CHANNEL_TESTING='/opt/ml/input/data/testing'
```

The **SM_CHANNEL_{channel_name}** environmental variable contains the directory where the channel named channel_name is located in the container.

Examples:

```
import json

parser.add_argument('--train', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
parser.add_argument('--test', type=str, default=os.environ['SM_CHANNEL_TESTING'])


args = parser.parse_args()

train_file = np.load(os.path.join(args.train, 'train.npz'))
test_file = np.load(os.path.join(args.test, 'test.npz'))
```

- **SM_HPS**

```
SM_HPS='{"batch-size": "256", "learning-rate": "0.0001","communicator": "pure_nccl"}'
```

The **SM_HPS** environmental variable contains a JSON encoded dictionary with the hyperparameters that you have provided.

Example:

```
import json

hyperparameters = json.loads(os.environ['SM_HPS']))
# {"batch-size": 256, "learning-rate": 0.0001, "communicator": "pure_nccl"}
```

- **SM_HP_ {hyperparameter_name}**

```
SM_HP_LEARNING-RATE=0.0001
SM_HP_BATCH-SIZE=10000
SM_HP_COMMUNICATOR=pure_nccl
```

The SM_HP_ {hyperparameter_name environmental variable contains the value of the hyperparameter named hyperparameter_name.

Examples:

```
learning_rate = float(os.environ['SM_HP_LEARNING-RATE'])
batch_size = int(os.environ['SM_HP_BATCH-SIZE'])
comminicator = os.environ['SM_HP_COMMUNICATOR']
```

- **SM_CURRENT_HOST**

```
SM_CURRENT_HOST=algo-1
```

The SM_CURRENT HOST contains the name of the current container on the container network.

Examples:

```
# Using it in argparse
parser.add_argument('current_host', type=str, default=os.environ['SM_CURRENT_HOST'])

# Using it as a variable
current_host = os.environ['SM_CURRENT_HOST']
```

- **SM_HOSTS**

```
SM_HOSTS='["algo-1","algo-2"]'
```

The SM_HOSTS environmental variable contains a JSON-encoded list of all of the hosts.

Example:

```
import json

# Using it in argparse
parser.add_argument('hosts', type=nargs, default=json.loads(os.environ['SM_HOSTS']))

# Using it as variable
hosts = json.loads(os.environ['SM_HOSTS'])
```

- **SM_NUM_GPUS**

```
SM_NUM_GPUS=1
```

The SM_NUM_GPUS environmental variable contains the number of GPUs available in the current container.

Examples:

```
# Using it in argparse
parser.add_argument('num_gpus', type=int, default=os.environ['SM_NUM_GPUS'])

# Using it as a variable
num_gpus = int(os.environ['SM_NUM_GPUS'])
```

Reference: Amazon SageMaker Containers Environmental Variables

The following build-time environment variables are also defined by default when you use the Amazon SageMaker Containers.

- **SM_NUM_CPUS**

```
SM_NUM_CPUS=32
```

The `SM_NUM_CPUS` environment variable contains the number of CPUs available in the current container.

Example:

```
# Using it in argparse
parser.add_argument('num_cpus', type=int, default=os.environ['SM_NUM_CPUS'])

# Using it as a variable
num_cpus = int(os.environ['SM_NUM_CPUS'])
```

- **SM_LOG_LEVEL**

```
SM_LOG_LEVEL=20
```

The `SM_LOG_LEVEL` environment variable contains the current log level in the container.

Example:

```
import logging

logger = logging.getLogger(__name__)

logger.setLevel(int(os.environ.get('SM_LOG_LEVEL', logging.INFO)))
```

- **SM_NETWORK_INTERFACE_NAME**

```
SM_NETWORK_INTERFACE_NAME=ethwe
```

The `SM_NETWORK_INTERFACE_NAME` environment variable contains the name of the network interface, which is used for distributed training.

Example:

```
# Using it in argparse
```

```
parser.add_argument('network_interface', type=str,
 default=os.environ['SM_NETWORK_INTERFACE_NAME'])

# Using it as a variable
network_interface = os.environ['SM_NETWORK_INTERFACE_NAME']
```

- **SM_USER_ARGS**

```
SM_USER_ARGS='["--batch-size","256","--learning_rate","0.0001","--
communicator","pure_nccl"]'
```

The `SM_INPUT_DIR` environment variable contains a JSON-encoded list of the script arguments provided for training.

- **SM_INPUT_DIR**

```
SM_INPUT_DIR=/opt/ml/input/
```

The `SM_INPUT_DIR` environment variable contains the path of the input directory, `/opt/ml/input/`. This is the directory where Amazon SageMaker saves input data and configuration files before and during training.

- **SM_INPUT_CONFIG_DIR**

```
SM_INPUT_DIR=/opt/ml/input/config
```

The `SM_INPUT_CONFIG_DIR` environment variable contains the path of the input config directory, `/opt/ml/input/config/`. This is the directory where standard Amazon SageMaker configuration files are located.

When training starts, Amazon SageMaker creates the following files in this directory:

- `hyperparameters.json` – Contains the hyperparameters specified in the `CreateTrainingJob` request.
- `inputdataconfig.json` – Contains the data channel information that you specified in the `InputDataConfig` parameter in a `CreateTrainingJob` request.
- `resourceconfig.json` – Contains the name of the current host and all host containers used in the training.

For more information, see: Using Your Own Training Algorithms.

- **SM_OUTPUT_DATA_DIR**

```
SM_OUTPUT_DATA_DIR=/opt/ml/output/data/algo-1
```

The `SM_OUTPUT_DATA_DIR` environment variable contains the directory where the algorithm writes non-model training artifacts, such as evaluation results. Amazon SageMaker retains these artifacts. As it runs in a container, your algorithm generates output, including the status of the training job and model, and the output artifacts. Your algorithm should write this information to this directory.

- **SM_RESOURCE_CONFIG**

```
SM_RESOURCE_CONFIG='{"current_host":"algo-1","hosts":["algo-1","algo-2"]}'
```

The `SM_RESOURCE_CONFIG` environment variable contains the contents of the `resourceconfig.json` file located in the `/opt/ml/input/config/` directory. It has the following keys:

- `current_host` – The name of the current container on the container network. For example, `"algo-1"`.

- `hosts` – The list of names of all of the containers on the container network, sorted lexicographically. For example, `["algo-1", "algo-2", "algo-3"]` for a three-node cluster.

For more information about the `resourceconfig.json` file, see: Distributed Training Configuration.

- **SM_INPUT_DATA_CONFIG**

```
SM_INPUT_DATA_CONFIG='{
    "testing": {
        "RecordWrapperType": "None",
        "S3DistributionType": "FullyReplicated",
        "TrainingInputMode": "File"
    },
    "training": {
        "RecordWrapperType": "None",
        "S3DistributionType": "FullyReplicated",
        "TrainingInputMode": "File"
    }
}'
```

The `SM_INPUT_DATA_CONFIG` environment variable contains the input data configuration of the `inputdataconfig.json` file located in the `/opt/ml/input/config/` directory.

For more information about the `resourceconfig.json` file, see Distributed Training Configuration.

- **SM_TRAINING_ENV**

```
SM_TRAINING_ENV='
{
    "channel_input_dirs": {
        "test": "/opt/ml/input/data/testing",
        "train": "/opt/ml/input/data/training"
    },
    "current_host": "algo-1",
    "framework_module": "sagemaker_chainer_container.training:main",
    "hosts": [
        "algo-1",
        "algo-2"
    ],
    "hyperparameters": {
        "batch-size": 10000,
        "epochs": 1
    },
    "input_config_dir": "/opt/ml/input/config",
    "input_data_config": {
        "test": {
            "RecordWrapperType": "None",
            "S3DistributionType": "FullyReplicated",
            "TrainingInputMode": "File"
        },
        "train": {
            "RecordWrapperType": "None",
            "S3DistributionType": "FullyReplicated",
            "TrainingInputMode": "File"
        }
    },
    "input_dir": "/opt/ml/input",
    "job_name": "preprod-chainer-2018-05-31-06-27-15-511",
    "log_level": 20,
    "model_dir": "/opt/ml/model",
```

```
    "module_dir": "s3://sagemaker-{aws-region}-{aws-id}/{training-job-name}/source/
sourcedir.tar.gz",
    "module_name": "user_script",
    "network_interface_name": "ethwe",
    "num_cpus": 4,
    "num_gpus": 1,
    "output_data_dir": "/opt/ml/output/data/algo-1",
    "output_dir": "/opt/ml/output",
    "resource_config": {
        "current_host": "algo-1",
        "hosts": [
            "algo-1",
            "algo-2"
        ]
    }
}'
```

The `SM_TRAINING_ENV` environment variable provides all of the training information as a JSON-encoded dictionary.

Additional Information for Scripts

Scripts can assign values for the hyperparameters of an algorithm. The interpreter passes all hyperparameters specified in the training job to the entry point as script arguments. For example, it passes the training job hyperparameters as follow.:

```
{"HyperParameters": {"batch-size": 256, "learning-rate": 0.0001, "communicator":
 "pure_nccl"}}
```

When an entry point needs additional information from the container that isn't available in hyperparameters, Amazon SageMaker Containers writes this information as environment variables that are available in the script. For example, the following training job includes the `training` and `testing` channels.

```
from sagemaker.pytorch import PyTorch

estimator = PyTorch(entry_point='train.py', ...)

estimator.fit({'training': 's3://bucket/path/to/training/data',
               'testing': 's3://bucket/path/to/testing/data'})
```

The environment variable SM_CHANNEL_`{channel_name}` provides the path where the channel is located.

```
import argparse
import os

if __name__ == '__main__':
  parser = argparse.ArgumentParser()

  ...

  # reads input channels training and testing from the environment variables
  parser.add_argument('--training', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
  parser.add_argument('--testing', type=str, default=os.environ['SM_CHANNEL_TESTING'])

  args = parser.parse_args()
  ...
```

Get Started: Use Amazon SageMaker Containers to Run a Python Script

To run an arbitrary script-based program in a Docker container using the Amazon SageMaker Containers, build a Docker container with an Amazon SageMaker notebook instance, as follows:

1. Create the notebook instance.
2. Create and upload the Dockerfile and Python scripts.
3. Build the container.
4. Test the container locally.
5. Clean up the resources.

To create an Amazon SageMaker notebook instance

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

2. Choose **Notebook** , **Notebook instances**, and **Create notebook instance**.

3. On the **Create notebook instance** page, provide the following information:

    a. For **Notebook instance name**, enter `RunScriptNotebookInstance`.

    b. For **Notebook Instance type**, choose `ml.t2.medium`.

    c. For **IAM role**, choose **Create a new role**.

        i. Choose **Create a new role**.

        ii. On the **Create an IAM role** page, choose **Specific S3 buckets**, specify an S3 bucket named `sagemaker-run-script`,. and then choose **Create role**.

            Amazon SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-`*`YYYYMMDD`*`T`*`HHmmSS`*. For example, `AmazonSageMaker-ExecutionRole-20190429T110788`. Record the role name because you'll need it later.

    d. For **Root Access**, accept the default, **Enabled**.

    e. Choose **Create notebook instance**.

        It takes a few minutes for Amazon SageMaker to launch an ML compute instance—in this case, a notebook instance—and attach an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see the CreateNotebookInstance (p. 656) API.

4. When the status of the notebook instance is `InService`, from **Actions**, choose **Open Jupyter**.

    For **New**, choose `conda_tensorflw_p36`. This is the kernel you need.

5. To name the notebook, choose **File**, **Rename**, enter t`Run-Python-Script`, and then choose **Rename**.

To create and upload the Dockerfile and Python scripts

1. In the editor of your choice, create the following Dockerfile text file locally and save it with the file name "Dockerfile" without an extension. The `docker build` command expects by default to find a file with precisely this name in the dockerfile directory. For example, in Notepad, you can save a text file without an extension by choosing **File**, **Save As** and choosing `All types(*.*)`.

```
FROM tensorflow/tensorflow:2.0.0a0
```

```
RUN pip install sagemaker-containers

# Copies the training code inside the container
COPY train.py /opt/ml/code/train.py

# Defines train.py as script entrypoint
ENV SAGEMAKER_PROGRAM train.py
```

The Dockerfile script performs the following tasks:

- `FROM tensorflow/tensorflow:2.0.0a0` downloads the TensorFlow library used to run the Python script.
- `RUN pip install sagemaker-containers`Amazon SageMaker Containers contains the common functionality necessary to create a container compatible with Amazon SageMaker.
- `COPY train.py /opt/ml/code/train.py` copies the script to the location inside the container that is expected by Amazon SageMaker. The script must be located in this folder.
- `ENV SAGEMAKER_PROGRAM train.py` defines train.py as the name of the entrypoint script that is located in the /opt/ml/code folder for the container. This is the only environmental variable that you must specify when, you are using your own container.

2. In the editor of your choice, create and save the following `train.py` text file locally.

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=1)

model.evaluate(x_test, y_test)
```

3. To upload the Dockerfile to a dockerfile directory, choose **Open JupyterLab**, choose the **File Browser** icon, and then choose the **New Folder** icon. This creates a new directory named **dockerfile**.

4. Double-click the new `dockerfile` folder, choose the **Upload Files** icon, navigate to where you saved your Dockerfile and `train.py` script files, and upload them to the `dockerfile` folder.

To build the container

1. The Jupyter Notebook opens in the SageMaker directory. The Docker build command must be run from the `dockerfile` directory you created. Run the following command to change into the `dockerfile` directory:

```
cd dockerfile
```

This returns your current directory: `/home/ec2-user/SageMaker/dockerfile`

2. To build the Docker container, run the following Docker build command, including the final period.

```
!docker build -t tf-2.0 .
```

To test the container locally

1. Use Local Mode the test the container locally. Replace the `'SageMakerRole'` value with the ARN for the role with the IAM role you created when configuring the notebook instance. The ARN should look like: `'arn:aws:iam::109225375568:role/service-role/AmazonSageMaker-ExecutionRole-20190429T110788'`.

```
from sagemaker.estimator import Estimator

estimator = Estimator(image_name='tf-2.0',
    role='SageMakerRole',
    train_instance_count=1,
    train_instance_type='local')

estimator.fit()
```

This test outputs the training environment configuration, the values used for the environmental variables, the source of the data, and the loss and accuracy obtained during training.

2. After using Local Mode, you can push the image to Amazon Elastic Container Registry and use it to run training jobs. For an example that shows how to complete these tasks, see Building Your Own TensorFlow Container

To clean up resources when done with the get started example

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/, stop and then delete the notebook instance.

2. Open the Amazon S3console at https://console.aws.amazon.com/s3 and delete the bucket that you created for storing model artifacts and the training dataset.

3. Open the IAM console at https://console.aws.amazon.com/iam/ and delete the IAM role. If you created permission policies, you can delete them, too.

> **Note**
> The Docker container shuts down automatically after it has run. You don't need to delete it.

Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch

Amazon SageMaker provides prebuilt Docker images that include deep learning framework libraries and other dependencies needed for training and inference. With the SageMaker Python SDK, you can train and deploy models using one of these popular deep learning frameworks. For instructions on installing and using the SDK, see Amazon SageMaker Python SDK.

The following table provides links to the GitHub repositories that contain the source code and Dockerfiles for each framework and for TensorFlow and MXNet Serving. The instructions linked are for using the Python SDK estimators to run your own training algorithms on Amazon SageMaker and your own models on Amazon SageMaker hosting.

Framework	Prebuilt Docker Image Source Code	Instructions
`TensorFlow`	Amazon SageMaker TensorFlow Containers Amazon SageMaker TensorFlow Serving Container	Using TensorFlow with the SageMaker Python SDK
`MXNet`	Amazon SageMaker MXNet Containers Amazon SageMaker MXNet Serving Container	Using MXNet with the SageMaker Python SDK
`Chainer`	Amazon SageMaker Chainer SageMaker Containers	Chainer SageMaker Estimators and Models
`PyTorch`	Amazon SageMaker PyTorch Containers	SageMaker PyTorch Estimators and Models

If you are not using the Amazon SageMaker Python SDK and one of its estimators to manage the container, you have to retrieve the relevant pre-built container. The Amazon SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). To pull an image from an Amazon ECR repo or to push an image to an Amazon ECR repo, use the fullname registry address of the image. Amazon SageMaker uses the following URL patterns for the container image registery addresses:

```
<account_id>.dkr.ecr.<region>.amazonaws.com/<ECR repo name>:<framework
version>-<processing unit type>-<python version>
```

The following table itemizes the supported values for each of the components in the URL registry addresses and how they are associated.

Here, for example, are some of the most common use cases for the deep learning frameworks supported the Amazon SageMaker:

- If you want to use **TensorFlow 1.13** or later to **train** a model with Python 3:

  ```
  763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-training::1.14-gpu-py3
  ```
- If you want to use **TensorFlow 1.14** or later to **train** a model with Python 2:

  ```
  763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-training::1.14-gpu-py2
  ```
- If you want to use **TensorFlow 1.14** or later for **inference** with Python 3:

  ```
  763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-inference::1.14-gpu-
  py3
  ```
- If you want to use **MxNet 1.4.1** or later to **train** a model or for **inference** with Python 3:

  ```
  763104351884.dkr.ecr.<region>.amazonaws.com/mxnet-training:1.4.1-gpu-py3
  ```

  ```
  763104351884.dkr.ecr.<region>.amazonaws.com/mxnet-inference:1.4.1-gpu-py3
  ```
- If you want to use **Chainer** or **PyTorch** to **train** a model or for **inference** with Python 2 or 3:

  ```
  520713654638.dkr.ecr.<region>.amazonaws.com/sagemaker-chainer:5.0.0-
  gpu-<python version>
  ```

  ```
  520713654638.dkr.ecr.<region>.amazonaws.com/sagemaker-pytorch:1.1.0-
  gpu-<python version>
  ```

URL Component	Description	Supported Values
`<account_id>`	Specifies the ID for Amazon SageMaker accounts that contain the pre-built containers.	`763104351884` `871362719292` `520713654638` `057415533634`
`<region>`	Specifies the AWS regions that contain the Amazon SageMaker accounts.	- Accounts `763104351884` and `871362719292` are located in: us-west-1, us-west-2, us-east-1, us-east-2, ap-northeast-1, ap-northeast-2, ap-southeast-1, ap-southeast-2, ap-south-1, eu-west-1, eu-west-2, eu-central-1, ca-central-1 - Accounts `520713654638` and `057415533634` are located in: ap-east-1

`<ECR repo name>`	Specifies the name of the public repository owned by Amazon SageMaker in the Amazon ECR.	Python 3 containers for TensorFlow-1.13 and later and for MXNet-1.4.1 and later in accounts `763104351884` and `871362719292`:
		
		    `tensorflow-training` (also for Python 2) `tensorflow-inference` `mxnet-training` `mxnet-inference`
		Other Python 2 and Python 3 containers in accounts `520713654638` and `057415533634`:
		
		    `sagemaker-tensorflow-scriptmode` (for Python 2 only) `sagemaker-tensorflow-serving-eia` `sagemaker-mxnet` (for Python 2 only) `sagemaker-mxnet-serving` (for Python 2 only) `sagemaker-mxnet-serving-eia` `sagemaker-chainer` `sagemaker-pytorch`

`<framework version>`	Specifies the framework and links to documentation for the estimators for each of the frameworks that explains how to specify the supported versions.	- TensorFlow: TensorFlow SageMaker Estimators - MXNet: MXNet SageMaker Estimators - Chainer: Chainer SageMaker Estimators - PyTorch: PyTorch SageMaker Estimators
`<processing unit type>`	Specifies whether to use a GPU or CPU for training or hosting.	`cpu` `gpu`
`<python version>`	Specifies the version of Python used. (Optional if you are using	`py2` `py3`

URL Component	Description	Supported Values
	the `tensorflow-inference` container for serving.)	

•

Amazon SageMaker also provides prebuilt Docker images for scikit-learn and Spark ML. For information about Docker images that enable using scikit-learn and Spark ML solutions in Amazon SageMaker, see Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML (p. 401).

You can use prebuilt containers to deploy your custom models or models that you have purchased on AWS Marketplace that have been trained in a framework other than Amazon SageMaker. For an overview of the process of bringing the trained model artifacts into Amazon SageMaker and hosting them at an endpoint, see Bring Your Own Pretrained MXNet or TensorFlow Models into Amazon SageMaker.

You can customize these prebuilt containers or extend them to handle any additional functional requirements for your algorithm or model that the prebuilt Amazon SageMaker Docker image doesn't support. For an example, see Extending Our PyTorch Containers.

Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML

Amazon SageMaker provides prebuilt Docker images that install the scikit-learn and Spark ML libraries and the dependencies they need to build Docker images that are compatible with Amazon SageMaker using the Amazon SageMaker Python SDK. With the SDK, you can use scikit-learn for machine learning tasks and use Spark ML to create and tune machine learning pipelines. For instructions on installing and using the SDK, see SageMaker Python SDK. The following table contains links to the GitHub repositories with the source code and the Dockerfiles for scikit-learn and Spark ML frameworks and to instructions that show how use the Python SDK estimators to run your own training algorithms on Amazon SageMaker Learner and your own models on Amazon SageMaker Hosting.

Library	Prebuilt Docker Image Source Code	Instructions
scikit-learn	SageMaker Scikit-learn Containers	Using Scikit-learn with the Amazon SageMaker Python SDK
Spark ML	SageMaker Spark ML Serving Containers	SparkML Serving

If you are not using the SM Python SDK and one of its estimators to manage the container, you have to retrieve the relevant pre-build container. The Amazon SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). You can push or pull them using their fullname registry addresses. Amazon SageMaker uses the following Docker Image URL patterns for scikit-learn and Spark M:

• `<ACCOUNT_ID>`.dkr.ecr.`<REGION_NAME>`.amazonaws.com/sagemaker-scikit-learn

  For example, `746614075791.dkr.ecr.us-west-1.amazonaws.com/sagemaker-scikit-learn`

• `<ACCOUNT_ID>`.dkr.ecr.`<REGION_NAME>`.amazonaws.com/sagemaker-sparkml-serving

For example, `341280168497.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-sparkml-serving`

The following table lists the supported values for account IDs and corresponding AWS Region names.

ACCOUNT_ID	REGION_NAME
746614075791	us-west-1
246618743249	us-west-2
683313688378	us-east-1
257758044811	us-east-2
354813040037	ap-northeast-1
366743142698	ap-northeast-2
121021644041	ap-southeast-1
783357654285	ap-southeast-2
720646828776	ap-south-1
141502667606	eu-west-1
764974769150	eu-west-2
492215442770	eu-central-1
341280168497	ca-central-1
414596584902	us-gov-west-1

The supported values listed in the table are also available on the fw_registry.py page of the Amazon SageMaker Python SDK GitHub repository.

Amazon SageMaker also provides prebuilt Docker images for popular deep learning frameworks. For information about Docker images that enable using deep learning frameworks in Amazon SageMaker, see Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 398).

For information on Docker images for developing reinforcement learning (RL) solutions in Amazon SageMaker, see Amazon SageMaker RL Containers.

Example Notebooks: Use Your Own Algorithm or Model

The following sample notebooks show how to use your own algorithms or pretrained models from an Amazon SageMaker notebook instance. . After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab for a list of all Amazon SageMaker example notebooks. You can open the sample notebooks from the **Advanced Functionality** section in your notebook instance or in GitHub at the provided links. To open a notebook, choose its **Use** tab, then choose **Create copy**.

For instructions on how to create and access Jupyter notebook instances, see Use Notebook Instances (p. 36)

To learn how to **host models trained in scikit-learn** for making predictions in Amazon SageMaker by injecting them first-party k-means and XGBoost containers, see the following sample notebooks.

- kmeans_bring_your_own_model - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/kmeans_bring_your_own_model
- xgboost_bring_your_own_model - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/xgboost_bring_your_own_model

To learn how to **package algorithms that you have developed in TensorFlow and scikit-learn frameworks for training and deployment in the Amazon SageMaker** environment, see the following notebooks. They show you how to build, register, and deploy you own Docker containers using Dockerfiles.

- tensorflow_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/tensorflow_bring_your_own
- scikit_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/scikit_bring_your_own

To learn how to **train a neural network locally using MXNet or TensorFlow**, and then create an endpoint from the trained model and **deploy it on Amazon SageMaker**, see the following notebooks. The MXNet model is trained to recognize handwritten numbers from the MNIST dataset. The TensorFlow model is trained to classify irises.

- mxnet_mnist_byom - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/mxnet_mnist_byom
- tensorflow_iris_byom - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/tensorflow_iris_byom

To learn how to **use a Dockerfile to build a container that calls the `train.py script`** and uses pipe mode to custom train an algorithm, see the following notebook. In pipe mode, the input data is transferred to the algorithm while it is training. This can decrease training time compared to using file-mode.

- pipe_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/pipe_bring_your_own

To learn how to **use an R container to train and host a model** with the R kernel installed in a notebook , see the following notebook. To take advantage of the AWS SDK for Python (Boto 3), we use Python within the notebook. You can achieve the same results completely in R by invoking command line arguments.

- r_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/r_bring_your_own

To learn how to **extend a prebuilt Amazon SageMaker PyTorch container image** when you have additional functional requirements for your algorithm or model that the pre-built Docker image doesn't support, see the following notebook.

- pytorch_extending_our_containers - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/pytorch_extending_our_containers

For links to the GitHub repositories with the prebuilt Dockerfiles for the TensorFlow, MXNet, Chainer, and PyTorch frameworks and instructions on use the AWS SDK for Python (Boto 3) estimators to run your own training algorithms on Amazon SageMaker Learner and your own models on Amazon SageMaker hosting, see Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 398)

Use Your Own Training Algorithms

This section explains how Amazon SageMaker interacts with a Docker container that runs your custom training algorithm. Use this information to write training code and create a Docker image for your training algorithms.

Topics

- How Amazon SageMaker Runs Your Training Image (p. 404)
- How Amazon SageMaker Provides Training Information (p. 405)
- How Amazon SageMaker Signals Algorithm Success and Failure (p. 407)
- How Amazon SageMaker Processes Training Output (p. 408)

How Amazon SageMaker Runs Your Training Image

To configure a Docker container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For model training, Amazon SageMaker runs the container as follows:

```
docker run image train
```

  Amazon SageMaker overrides any default `CMD` statement in a container by specifying the `train` argument after the image name. The `train` argument also overrides arguments that you provide using `CMD` in the Dockerfile.

- Use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT ["executable", "param1", "param2", ...]
```

  For example:

```
ENTRYPOINT ["python", "k-means-algorithm.py"]
```

  The `exec` form of the `ENTRYPOINT` instruction starts the executable directly, not as a child of `/bin/sh`. This enables it to receive signals like `SIGTERM` and `SIGKILL` from Amazon SageMaker APIs. Note the following:

  - The CreateTrainingJob (p. 667) API has a stopping condition that directs Amazon SageMaker to stop model training after a specific time.

  - The StopTrainingJob (p. 834) API issues the equivalent of the `docker stop`, with a 2 minute timeout, command to gracefully stop the specified container:

```
docker stop –t120
```

The command attempts to stop the running container by sending a `SIGTERM` signal. After the 2 minute timeout, SIGKILL is sent and the containers are forcibly stopped. If the container handles the SIGTERM gracefully and exits within 120 seconds from receiving it, no SIGKILL is sent.

> **Note**
> If you want access to the intermediate model artifacts after Amazon SageMaker stops the training, add code to handle saving artifacts in your `SIGTERM` handler.

- If you plan to use GPU devices for model training, make sure that your containers are `nvidia-docker` compatible. Only the CUDA toolkit should be included on containers; don't bundle NVIDIA drivers with the image. For more information about `nvidia-docker`, see [NVIDIA/nvidia-docker](#).

- You can't use the `tini` initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

- `/opt/ml` and all sub-directories are reserved by Amazon SageMaker training. When building your algorithm's docker image, please ensure you don't place any data required by your algorithm under them as the data may no longer be visible during training.

How Amazon SageMaker Provides Training Information

This section explains how Amazon SageMaker makes training information, such as training data, hyperparameters, and other configuration information, available to your Docker container.

When you send a CreateTrainingJob (p. 667) request to Amazon SageMaker to start model training, you specify the Amazon Elastic Container Registry path of the Docker image that contains the training algorithm. You also specify the Amazon Simple Storage Service (Amazon S3) location where training data is stored and algorithm-specific parameters. Amazon SageMaker makes this information available to the Docker container so that your training algorithm can use it. This section explains how we make this information available to your Docker container. For information about creating a training job, see `CreateTrainingJob`.

Topics

Hyperparameters

Amazon SageMaker makes the hyperparameters in a `CreateTrainingJob` request available in the Docker container in the `/opt/ml/input/config/hyperparameters.json` file.

Environment Variables

- TRAINING_JOB_NAME—The training job name stored in the `TrainingJobName` parameter in a CreateTrainingJob (p. 667) request.
- TRAINING_JOB_ARN—The Amazon Resource Name (ARN) of the training job returned as the `TrainingJobArn` response element for CreateTrainingJob (p. 667).

Input Data Configuration

You specify data channel information in the `InputDataConfig` parameter in a `CreateTrainingJob` request. Amazon SageMaker makes this information available in the `/opt/ml/input/config/inputdataconfig.json` file in the Docker container.

For example, suppose that you specify three data channels (`train`, `evaluation`, and `validation`) in your request. Amazon SageMaker provides the following JSON:

```
{
"train" : {"ContentType":  "trainingContentType",
           "TrainingInputMode": "File",
           "S3DistributionType": "FullyReplicated",
           "RecordWrapperType": "None"},
"evaluation" : {"ContentType":  "evalContentType",
                "TrainingInputMode": "File",
                "S3DistributionType": "FullyReplicated",
                "RecordWrapperType": "None"},
"validation" : {"TrainingInputMode": "File",
                "S3DistributionType": "FullyReplicated",
                "RecordWrapperType": "None"}
}
```

> **Note**
> Amazon SageMaker provides only relevant information about each data channel (for example, the channel name and the content type) to the container, as shown. `S3DistributionType` will be set as `FullyReplicated` if specify EFS or FSxLustre as input data sources.

Training Data

The `TrainingInputMode` parameter in a `CreateTrainingJob` request specifies how to make data available for model training: in `FILE` mode or `PIPE` mode. Depending on the specified input mode, Amazon SageMaker does the following:

- `FILE` mode—Amazon SageMaker makes the data for the channel available in the `/opt/ml/input/data/channel_name` directory in the Docker container. For example, if you have three channels named `training`, `validation`, and `testing`, Amazon SageMaker makes three directories in the Docker container:
  - `/opt/ml/input/data/training`
  - `/opt/ml/input/data/validation`
  - `/opt/ml/input/data/testing`

- `PIPE` mode—Amazon SageMaker makes data for the channel available from the named pipe: `/opt/ml/input/data/channel_name_epoch_number`. For example, if you have three channels named `training`, `validation`, and `testing`, you will need to read from the following pipes:
  - `/opt/ml/input/data/training_0,/opt/ml/input/data/training_1, ...`
  - `/opt/ml/input/data/validation_0, /opt/ml/input/data/validation_1, ...`
  - `/opt/ml/input/data/testing_0, /opt/ml/input/data/testing_1, ...`

  Read the pipes sequentially. For example, if you have a channel called **training**, read the pipes in this sequence:
  1. Open `/opt/ml/input/data/training_0` in read mode and read it to EOF (or if you are done with the first epoch, close the file early).
  2. After closing the first pipe file, look for `/opt/ml/input/data/training_1` and read it to go through the second epoch, and so on.

If the file for a given epoch doesn't exist yet, your code may need to retry until the pipe is created. There is no sequencing restriction across channel types. That is, you can read multiple epochs for the **training** channel, for example, and only start reading the **validation** channel when you are ready. Or, you can read them simultaneously if your algorithm requires that.

Distributed Training Configuration

If you're performing distributed training with multiple containers, Amazon SageMaker makes information about all containers available in the `/opt/ml/input/config/resourceconfig.json` file.

To enable inter-container communication, this JSON file contains information for all containers. Amazon SageMaker makes this file available for both FILE and PIPE mode algorithms. The file provides the following information:

- `current_host`—The name of the current container on the container network. For example, `algo-1`. Host values can change at any time. Don't write code with specific values for this variable.
- `hosts`—The list of names of all containers on the container network, sorted lexicographically. For example, `["algo-1", "algo-2", "algo-3"]` for a three-node cluster. Containers can use these names to address other containers on the container network. Host values can change at any time. Don't write code with specific values for these variables.
- `network_interface_name`—The name of the network interface that is exposed to your container. For example, containers running the Message Passing Interface (MPI) can use this information to set the network interface name.
- Do not use the information in `/etc/hostname` or `/etc/hosts` because it might be inaccurate.
- Hostname information may not be immediately available to the algorithm container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.

The following is an example file on node 1 in a three-node cluster:

```
{
"current_host": "algo-1",
"hosts": ["algo-1","algo-2","algo-3"],
"network_interface_name":"eth1"
}
```

How Amazon SageMaker Signals Algorithm Success and Failure

A training algorithm indicates whether it succeeded or failed using the exit code of its process.

A successful training execution should exit with an exit code of 0 and an unsuccessful training execution should exit with a non-zero exit code. These will be converted to "Completed" and "Failed" in the `TrainingJobStatus` returned by `DescribeTrainingJob`. This exit code convention is standard and is easily implemented in all languages. For example, in Python, you can use `sys.exit(1)` to signal a failure exit and simply running to the end of the main routine will cause Python to exit with code 0.

In the case of failure, the algorithm can write a description of the failure to the failure file. See next section for details.

How Amazon SageMaker Processes Training Output

As your algorithm runs in a container, it generates output including the status of the training job and model and output artifacts. Your algorithm should write this information to the following files, which are located in the container's `/output` directory. Amazon SageMaker processes the information contained in this directory as follows:

- `/opt/ml/output/failure`—If training fails, after all algorithm output (for example, logging) completes, your algorithm should write the failure description to this file. In a `DescribeTrainingJob` response, Amazon SageMaker returns the first 1024 characters from this file as `FailureReason`.

- `/opt/ml/model`—Your algorithm should write all final model artifacts to this directory. Amazon SageMaker copies this data as a single object in compressed tar format to the S3 location that you specified in the `CreateTrainingJob` request. If multiple containers in a single training job write to this directory they should ensure no `file/directory` names clash. Amazon SageMaker aggregates the result in a tar file and uploads to s3.

Use Your Own Inference Code

You can use Amazon SageMaker to interact with Docker containers and run your own inference code in one of two ways:

- To use your own inference code with a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
- To use your own inference code to get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics

Use Your Own Inference Code with Hosting Services

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for hosting services. Use this information to write inference code and create a Docker image.

Topics

How Amazon SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For model inference, Amazon SageMaker runs the container as:

```
docker run image serve
```

  Amazon SageMaker overrides default `CMD` statements in a container by specifying the `serve` argument after the image name. The `serve` argument overrides arguments that you provide with the `CMD` command in the Dockerfile.

- We recommend that you use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

  For example:

```
ENTRYPOINT ["python", "k_means_inference.py"]
```

  The `exec` form of the `ENTRYPOINT` instruction starts the executable directly, not as a child of `/bin/sh`. This enables it to receive signals like `SIGTERM` and `SIGKILL` from the Amazon SageMaker APIs, which is a requirement.

  For example, when you use the CreateEndpoint (p. 632) API to create an endpoint, Amazon SageMaker provisions the number of ML compute instances required by the endpoint configuration, which you specify in the request. Amazon SageMaker runs the Docker container on those instances.

  If you reduce the number of instances backing the endpoint (by calling the UpdateEndpointWeightsAndCapacities (p. 842) APIs), Amazon SageMaker runs a command to stop the Docker container on the instances being terminated. The command sends the `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

  If you update the endpoint (by calling the UpdateEndpoint (p. 840) API), Amazon SageMaker launches another set of ML compute instances and runs the Docker containers that contain your inference code on them. Then it runs a command to stop the previous Docker containers. To stop a Docker container, command sends the `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

- Amazon SageMaker uses the container definition that you provided in your CreateModel (p. 648) request to set environment variables and the DNS hostname for the container as follows:

  - It sets environment variables using the `ContainerDefinition.Environment` string-to-string map.
  - It sets the DNS hostname using the `ContainerDefinition.ContainerHostname`.

- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your `CreateEndpointConfig` request), make sure that your containers are `nvidia-docker`

compatible. Don't bundle NVIDIA drivers with the image. For more information about `nvidia-docker`, see NVIDIA/nvidia-docker.

- You can't use the `tini` initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

How Amazon SageMaker Loads Your Model Artifacts

In your CreateModel (p. 648) request, the container definition includes the `ModelDataUrl` parameter, which identifies the S3 location where model artifacts are stored. Amazon SageMaker uses this information to determine where to copy the model artifacts from. It copies the artifacts to the `/opt/ml/model` directory for use by your inference code.

The `ModelDataUrl` must point to a tar.gz file. Otherwise, Amazon SageMaker won't download the file.

If you trained your model in Amazon SageMaker, the model artifacts are saved as a single compressed tar file in Amazon S3. If you trained your model outside Amazon SageMaker, you need to create this single compressed tar file and save it in a S3 location. Amazon SageMaker decompresses this tar file into /opt/ml/model directory before your container starts.

How Containers Serve Requests

Containers need to implement a web server that responds to `/invocations` and `/ping` on port 8080.

How Your Container Should Respond to Inference Requests

To obtain inferences, the client application sends a POST request to the Amazon SageMaker endpoint. For more information, see the InvokeEndpoint (p. 853) API. Amazon SageMaker passes the request to the container, and returns the inference result from the container to the client. Note the following:

- Amazon SageMaker strips all `POST` headers except those supported by `InvokeEndpoint`. Amazon SageMaker might add additional headers. Inference containers must be able to safely ignore these additional headers.
- To receive inference requests, the container must have a web server listening on port 8080 and must accept `POST` requests to the `/invocations` endpoint.
- A customer's model containers must accept socket connection requests within 250 ms.
- A customer's model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the /invocations. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

How Your Container Should Respond to Health Check (Ping) Requests

The `CreateEndpoint` and `UpdateEndpoint` API calls result in Amazon SageMaker starting new inference containers. Soon after container startup, Amazon SageMaker starts sending periodic GET requests to the /ping endpoint.

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the /invocations endpoint.

If the container does not begin to pass health checks, by consistently responding with 200s, during the 4 minutes after startup, `CreateEndPoint` will fail, leaving Endpoint in a failed state, and the update requested by `UpdateEndpoint` will not be completed.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on /ping attempts is 2 seconds.

Use Your Own Inference Code with Batch Transform

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for batch transform. Use this information to write inference code and create a Docker image.

Topics

How Amazon SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For batch transforms, Amazon SageMaker runs the container as:

```
docker run image serve
```

  Amazon SageMaker overrides default `CMD` statements in a container by specifying the `serve` argument after the image name. The `serve` argument overrides arguments that you provide with the `CMD` command in the Dockerfile.

- We recommend that you use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

  For example:

```
ENTRYPOINT ["python", "k_means_inference.py"]
```

- Amazon SageMaker sets environment variables specified in CreateModel (p. 648) and CreateTransformJob (p. 673) on your container. Additionally, the following environment variables will be populated:
  - `SAGEMAKER_BATCH` is always set to `true` when the container runs in Batch Transform.
  - `SAGEMAKER_MAX_PAYLOAD_IN_MB` is set to the largest size payload that will be sent to the container via HTTP.
  - `SAGEMAKER_BATCH_STRATEGY` will be set to `SINGLE_RECORD` when the container will be sent a single record per call to invocations and `MULTI_RECORD` when the container will get as many records as will fit in the payload.

- `SAGEMAKER_MAX_CONCURRENT_TRANSFORMS` is set to the maximum number of `/invocations` requests that can be opened simultaneously.

  Note
  The last three environment variables come from the API call made by the user. If the user doesn't set values for them, they aren't passed. In that case, either the default values or the values requested by the algorithm (in response to the `/execution-parameters`) are used.

- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your `CreateTransformJob` request), make sure that your containers are nvidia-docker compatible. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see NVIDIA/nvidia-docker.

- You can't use the `init` initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

How Amazon SageMaker Loads Your Model Artifacts

In a CreateModel (p. 648) request, container definitions includes the `ModelDataUrl` parameter, which identifies the location in Amazon S3 where model artifacts are stored. When you use Amazon SageMaker to run inferences, it uses this information to determine where to copy the model artifacts from. It copies the artifacts to the `/opt/ml/model` directory in the Docker container for use by your inference code.

The `ModelDataUrl` parameter must point to a tar.gz file. Otherwise, Amazon SageMaker can't download the file. If you train a model in Amazon SageMaker, it saves the artifacts as a single compressed tar file in Amazon S3. If you train a model in another framework, you need to store the model artifacts in Amazon S3 as a compressed tar file. Amazon SageMaker decompresses this tar file and saves it in the `/opt/ml/model` directory in the container before the batch transform job starts.

How Containers Serve Requests

Containers must implement a web server that responds to invocations and ping requests on port 8080. For batch transforms,you have the option to set algorithms to implement execution-parameters requests to provide a dynamic runtime configuration to Amazon SageMaker. Amazon SageMaker uses the following endpoints:

- `ping`—Used to periodically check the health of the container. Amazon SageMaker waits for an HTTP `200` status code and an empty body for a successful ping request before sending an invocations request. You might use a ping request to load a model into memory to generate inference when invocations requests are sent.
- (Optional) `execution-parameters`—Allows the algorithm to provide the optimal tuning parameters for a job during runtime. Based on the memory and CPUs available for a container, the algorithm chooses the appropriate `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` values for the job.

Before calling the invocations request, Amazon SageMaker attempts to invoke the execution-parameters request. When you create a batch transform job, you can provide values for the `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` parameters. Amazon SageMaker determines the values for these parameters using this order of precedence:

1. The parameter values that you provide when you create the `CreateTransformJob` request,
2. The values that the model container returns when Amazon SageMaker invokes the execution-parameters endpoint
3. The parameters default values, listed in the following table.

Parameter	Default Values
MaxConcurrentTransforms	1
BatchStrategy	MULTI_RECORD
MaxPayloadInMB	6

The response for a `GET` execution-parameters request is a JSON object with keys for `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` parameters. This is an example of a valid response:

```
{
  "MaxConcurrentTransforms": 8,
  "BatchStrategy": "MULTI_RECORD",
  "MaxPayloadInMB": 6
}
```

How Your Container Should Respond to Health Check (Ping) Requests

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the `/invocations` endpoint.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on `/ping` attempts is 2 seconds.

Create Algorithm and Model Package Resources

After your training and/or inference code is packaged in Docker containers, create algorithm and model package resources that you can use in your Amazon SageMaker account and, optionally, publish on AWS Marketplace.

Topics
- Create an Algorithm Resource (p. 413)
- Create a Model Package Resource (p. 417)

Create an Algorithm Resource

To create an algorithm resource that you can use to run training jobs in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker containers that contains the training and, optionally, inference code.
- The configuration of the input data that your algorithm expects for training.
- The hyperparameters that your algorithm supports.
- Metrics that your algorithm sends to Amazon CloudWatch during training jobs.
- The instance types that your algorithm supports for training and inference, and whether it supports distributed training across multiple instances.

- Validation profiles, which are training jobs that Amazon SageMaker uses to test your algorithm's training code and batch transform jobs that Amazon SageMaker runs to test your algorithm's inference code.

  To ensure that buyers and sellers can be confident that products work in Amazon SageMaker, we require that you validate your algorithms before listing them on AWS Marketplace. You can list products in the AWS Marketplace only if validation succeeds. To validate your algorithms, Amazon SageMaker uses your validation profile and sample data to run the following validations tasks:

  1. Create a training job in your account to verify that your training image works with Amazon SageMaker.
  2. If you included inference code in your algorithm, create a model in your account using the algorithm's inference image and the model artifacts produced by the training job.
  3. If you included inference code in your algorithm, create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.

  When you list your product on AWS Marketplace, the inputs and outputs of this validation process persist as part of your product and are made available to your buyers. This helps buyers understand and evaluate the product before they buy it. For example, buyers can inspect the input data that you used, the outputs generated, and the logs and metrics emitted by your code. The more comprehensive your validation specification, the easier it is for customers to evaluate your product.

  > **Note**
  > In your validation profile, provide only data that you want to expose publicly.

  Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the **Training jobs** and **Transform jobs** pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. If any issues are found, you will have to create the algorithm again.

  > **Note**
  > To publish your algorithm on AWS Marketplace, at least one validation profile is required.

You can create an algorithm by using either the Amazon SageMaker console or the Amazon SageMaker API.

Topics

Create an Algorithm Resource (Console)

To create an algorithm resource (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Algorithms**, then choose **Create algorithm**.
3. On the **Training specifications** page, provide the following information:

   a. For **Algorithm name**, type a name for your algorithm. The algorithm name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

   b. Type a description for your algorithm. This description appears in the Amazon SageMaker console and in the AWS Marketplace.

   c. For **Training image, type the path in Amazon ECR where your training container is stored.**

   d. For **Support distributed training**, Choose **Yes** if your algorithm supports training on multiple instances. Otherwise, choose **No**.

e. For **Support instance types for training**, choose the instance types that your algorithm supports.

f. For **Channel specification**, specify up to 8 channels of input data for your algorithm. For example, you might specify 3 input channels named `train`, `validation`, and `test`. For each channel, specify the following information:

   i. For **Channel name**, type a name for the channel. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

   ii. To require the channel for your algorithm, choose **Channel required**.

   iii. Type a description for the channel.

   iv. For **Supported input modes**, choose **Pipe mode** if your algorithm supports streaming the input data, and **File mode** if your algorithm supports downloading the input data as a file. You can choose both.

   v. For **Supported content types**, type the MIME type that your algorithm expects for input data.

   vi. For **Supported compression type**, choose **Gzip** if your algorithm supports Gzip compression. Otherwise, choose **None**.

   vii. Choose **Add channel** to add another data input channel, or choose **Next** if you are done adding channels.

4. On the **Tuning specifications** page, provide the following information:

   a. For **Hyperparameter specification**, specify the hyperparameters that your algorithm supports by editing the JSON object. For each hyperparameter that your algorithm supports, construct a JSON block similar to the following:

```
{
  "DefaultValue": "5",
  "Description": "The first hyperparameter",
  "IsRequired": true,
  "IsTunable": false,
  "Name": "intRange",
  "Range": {
   "IntegerParameterRangeSpecification": {
    "MaxValue": "10",
    "MinValue": "1"
   }
  },
  "Type": "Integer"
```

   In the JSON, supply the following:

   i. For `DefaultValue`, specify a default value for the hyperparameter, if there is one.

   ii. For `Description`, specify a description for the hyperparameter.

   iii. For `IsRequired`, specify whether the hyperparameter is required.

   iv. For `IsTunable`, specify `true` if this hyperparameter can be tuned when a user runs a hyperparameter tuning job that uses this algorithm. For information, see Automatic Model Tuning (p. 288).

   v. For `Name`, specify a name for the hyperparameter.

   vi. For `Range`, specify one of the following:

   - `IntegerParameterRangeSpecification` - the values of the hyperparameter are integers. Specify minimum and maximum values for the hyperparameter.

   -

   - `ContinuousParameterRangeSpecification` - the values of the hyperparameter are floating-point values. Specify minimum and maximum values for the hyperparameter.

- `CategoricalParameterRangeSpecification` - the values of the hyperparameter are categorical values. Specify a list of all of the possible values.

vii. For `Type`, specify `Integer`, `Continuous`, or `Categorical`. The value must correspond to the type of `Range` that you specified.

b. For **Metric definitions**, specify any training metrics that you want your algorithm to emit. Amazon SageMaker uses the regular expression that you specify to find the metrics by parsing the logs from your training container during training. Users can view these metrics when they run training jobs with your algorithm, and they can monitor and plot the metrics in Amazon CloudWatch. For information, see Monitor and Analyze Training Jobs Using Metrics (p. 276). For each metric, provide the following information:

   i. For **Metric name**, type a name for the metric.

   ii. For `Regex`, type the regular expression that Amazon SageMaker uses to parse training logs so that it can find the metric value.

   iii. For **Objective metric support** choose **Yes** if this metric can be used as the objective metric for a hyperparameter tuning job. For information, see Automatic Model Tuning (p. 288).

   iv. Choose **Add metric** to add another metric, or choose **Next** if you are done adding metrics.

5. On the **Inference specifications** page, provide the following information if your algorithm supports inference:

   a. For **Container definition**, type path in Amazon ECR where your inference container is stored.

   b. For **Container DNS host name**, type the name of a DNS host for your image.

   c. For **Supported instance types for real-time inference**, choose the instance types that your algorithm supports for models deployed as hosted endpoints in Amazon SageMaker. For information, see Deploy a Model on Amazon SageMaker Hosting Services (p. 7).

   d. For **Supported instance types for batch transform jobs**, choose the instance types that your algorithm supports for batch transform jobs. For information, see Get Inferences for an Entire Dataset with Batch Transform (p. 10).

   e. For **Supported content types**, type the type of input data that your algorithm expects for inference requests.

   f. For **Supported response MIME types**, type the MIME types that your algorithm supports for inference responses.

   g. Choose **Next**.

6. On the **Validation specifications** page, provide the following information:

   a. For **Publish this algorithm on AWS Marketplace**, choose **Yes** to publish the algorithm on AWS Marketplace.

   b. For **Validate this algorithm**, choose **Yes** if you want Amazon SageMaker to run training jobs and/or batch transform jobs that you specify to test the training and/or inference code of your algorithm.

      Note
      To publish your algorithm on AWS Marketplace, your algorithm must be validated.

   c. For **IAM role**, choose an IAM role that has the required permissions to run training jobs and batch transform jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see Amazon SageMaker Roles (p. 496).

   d. For **Validation profile**, specify the following:

      - A name for the validation profile.

      - A **Training job definition**. This is a JSON block that describes a training job. This is in the same format as the TrainingJobDefinition (p. 1015) input parameter of the CreateAlgorithm (p. 622) API.

- A **Transform job definition**. This is a JSON block that describes a batch transform job. This is in the same format as the TransformJobDefinition (p. 1026)input parameter of the CreateAlgorithm (p. 622) API.

e. Choose **Create algorithm**.

Create an Algorithm Resource (API)

To create an algorithm resource by using the Amazon SageMaker API, call the CreateAlgorithm (p. 622) API.

Create a Model Package Resource

To create a model package resource that you can use to create deployable models in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker container that contains the inference code, or the algorithm resource that was used to train the model.

- The location of the model artifacts. Model artifacts can either be packaged in the same Docker container as the inference code or stored in Amazon S3.

- The instance types that your model package supports for both real-time inference and batch transform jobs.

- Validation profiles, which are batch transform jobs that Amazon SageMaker runs to test your model package's inference code.

  Before listing model packages on AWS Marketplace, you must validate them. This ensures that buyers and sellers can be confident that products work in Amazon SageMaker. You can list products on AWS Marketplace only if validation succeeds.

  The validation procedure uses your validation profile and sample data to run the following validations tasks:

  1. Create a model in your account using the model package's inference image and the optional model artifacts that are stored in Amazon S3.

     Note
     A model package is specific to the region in which you create it. The S3 bucket where the model artifacts are stored must be in the same region where your created the model package.

  2. Create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.

  3. Create a validation profile.

     Note
     In your validation profile, provide only data that you want to expose publicly.

  Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the **Transform jobs** pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. After fixing issues, recreate the algorithm. When the status of the algorithm is COMPLETED, find it in the Amazon SageMaker console and start the listing process

     Note
     To publish your model package on AWS Marketplace, at least one validation profile is required.

You can create an model package either by using the Amazon SageMaker console or by using the Amazon SageMaker API.

Topics

- Create a Model Package Resource (Console) (p. 418)
- Create a Model Package Resource (API) (p. 419)

Create a Model Package Resource (Console)

To create a model package in the Amazon SageMaker console:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

2. Choose **Model packages**, then choose **Create model package**.

3. On the **Inference specifications** page, provide the following information:

    a. For **Model package name**, type a name for your model package. The model package name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

    b. Type a description for your model package. This description appears in the Amazon SageMaker console and in the AWS Marketplace.

    c. For **Inference specification options**, choose **Provide the location of the inference image and model artifacts** to create a model package by using an inference container and model artifacts. Choose **Provide the algorithm used for training and its model artifacts** to create a model package from an algorithm resource that you created or subscribe to from AWS Marketplace.

    d. If you chose **Provide the location of the inference image and model artifacts** for **Inference specification options**, provide the following information for **Container definition** and **Supported resources**:

        i. For **Location of inference image**, type the path to the image that contains your inference code. The image must be stored as a Docker container in Amazon ECR.

        ii. For **Location of model data artifacts**, type the location in S3 where your model artifacts are stored.

        iii. For **Container DNS host name** , type the name of the DNS host to use for your container.

        iv. For **Supported instance types for real-time inference**, choose the instance types that your model package supports for real-time inference from Amazon SageMaker hosted endpoints.

        v. For **Supported instance types for batch transform jobs**, choose the instance types that your model package supports for batch transform jobs.

        vi. **Supported content types**, type the content types that your model package expects for inference requests.

        vii. For **Supported response MIME types**, type the MIME types that your model package uses to provide inferences.

    e. If you chose **Provide the algorithm used for training and its model artifacts** for **Inference specification options**, provide the following information:

        i. For **Algorithm ARN**, type the Amazon Resource Name (ARN) of the algorithm resource to use to create the model package.

        ii. For **Location of model data artifacts**, type the location in S3 where your model artifacts are stored.

    f. Choose **Next**.

4. On the **Validation and scanning** page, provide the following information:

a. For **Publish this model package on AWS Marketplace**, choose **Yes** to publish the model package on AWS Marketplace.

b. For **Validate this model package**, choose **Yes** if you want Amazon SageMaker to run batch transform jobs that you specify to test the inference code of your model package.

> **Note**
> To publish your model package on AWS Marketplace, your model package must be validated.

c. For **IAM role**, choose an IAM role that has the required permissions to run batch transform jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see Amazon SageMaker Roles (p. 496).

d. For **Validation profile**, specify the following:

- A name for the validation profile.
- A **Transform job definition**. This is a JSON block that describes a batch transform job. This is in the same format as the TransformJobDefinition (p. 1026)input parameter of the CreateAlgorithm (p. 622) API.

5. Choose **Create model package**.

Create a Model Package Resource (API)

To create a model package by using the Amazon SageMaker API, call the CreateModelPackage (p. 652) API.

Use Algorithm and Model Package Resources

You can create algorithms and model packages as resources in your Amazon SageMaker account, and you can find and subscribe to algorithms and model packages on AWS Marketplace.

Use algorithms to:

- Run training jobs. For information, see Use an Algorithm to Run a Training Job (p. 420).
- Run hyperparameter tuning jobs. For information, see Use an Algorithm to Run a Hyperparameter Tuning Job (p. 423).
- Create model packages. After you use an algorithm resource to run a training job or a hyperparameter tuning job, you can use the model artifacts that these jobs output along with the algorithm to create a model package. For information, see Create a Model Package Resource (p. 417).

> **Note**
> If you subscribe to an algorithm on AWS Marketplace, you must create a model package before you can use it to get inferences by creating hosted endpoint or running a batch transform job.
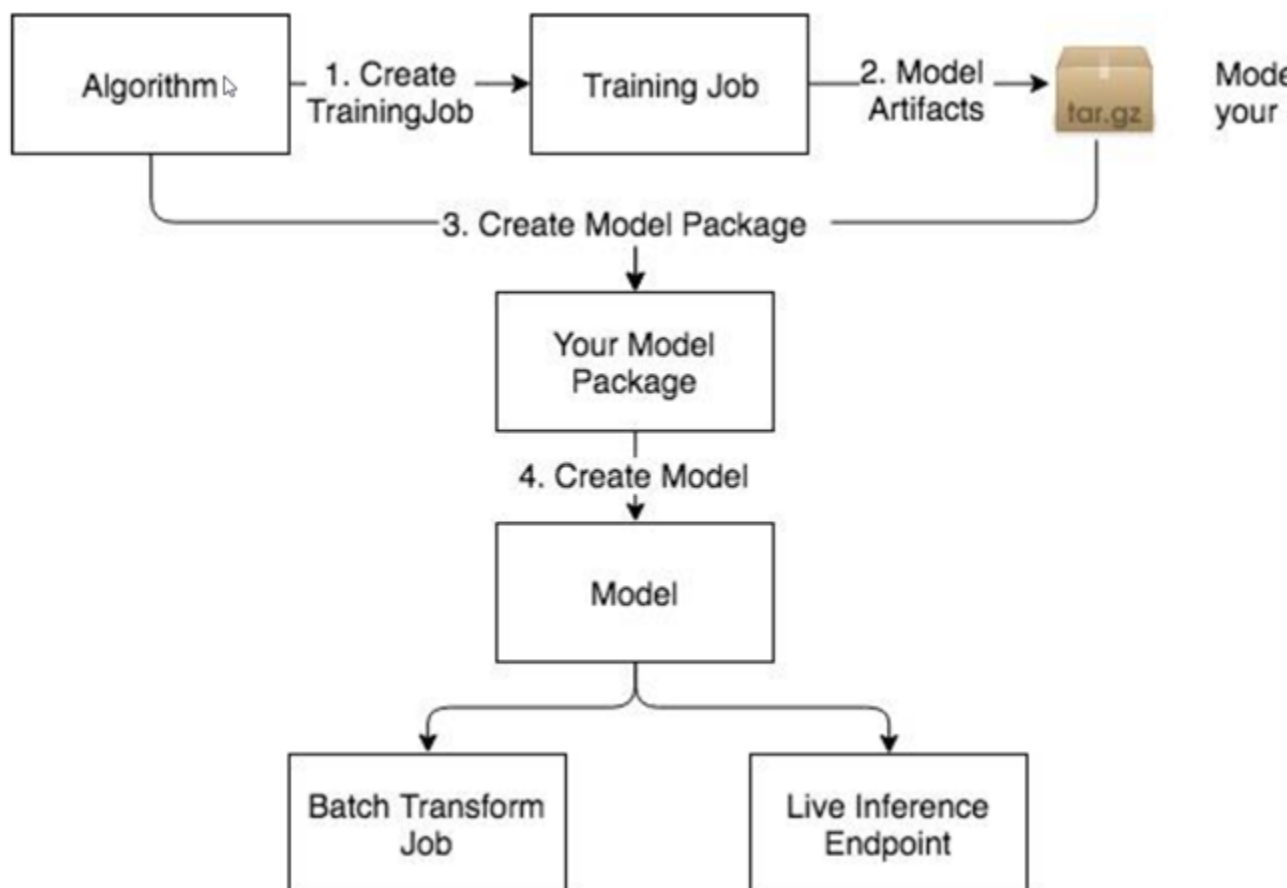
Use model packages to:

- Create models that you can use to get real-time inference or run batch transform jobs. For information, see Use a Model Package to Create a Model (p. 425).
- Create hosted endpoints to get real-time inference. For information, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26).
- Create batch transform jobs. For information, see Step 6.2: Deploy the Model with Batch Transform (p. 28).

Topics

Use an Algorithm to Run a Training Job

You can create use an algorithm resource to create a training job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

Topics

Use an Algorithm to Run a Training Job (Console)

To use an algorithm to run a training job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Algorithms**.
3. Choose an algorithm that you created from the list on the **My algorithms** tab or choose an algorithm that you subscribed to on the **AWS Marketplace subscriptions** tab.
4. Choose **Create training job**.

   The algorithm you chose will automatically be selected.
5. On the **Create training job** page, provide the following information:

   a. For **Job name**, type a name for the training job.

   b. For **IAM role**, choose an IAM role that has the required permissions to run training jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see Amazon SageMaker Roles (p. 496).

   c. For **Resource configuration**, provide the following information:

      i. For **Instance type**, choose the instance type to use for training.

      ii. For **Instance count**, type the number of ML instances to use for the training job.

      iii. For **Additional volume per instance (GB)**, type the size of the ML storage volume that you want to provision. ML storage volumes store model artifacts and incremental states.

      iv. For **Encryption key**, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instance, specify the key.

      v. For **Stopping condition**, specify the maximum amount of time in seconds, minutes, hours, or days, that you want the training job to run.

   d. For **VPC**, choose a Amazon VPC that you want to allow your training container to access. For more information, see Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC (p. 522).

   e. For **Hyperparameters**, specify the values of the hyperparameters to use for the training job.

   f. For **Input data configuration**, specify the following values for each channel of input data to use for the training job. You can see what channels the algorithm you're using for training sports, and the content type, supported compression type, and supported input modes for each channel, under **Channel specification** section of the **Algorithm summary** page for the algorithm.

      i. For **Channel name**, type the name of the input channel.

      ii. For **Content type**, type the content type of the data that the algorithm expects for the channel.

      iii. For **Compression type**, choose the data compression type to use, if any.

      iv. For **Record wrapper**, choose `RecordIO` if the algorithm expects data in the `RecordIO` format.

      v. For **S3 data type**, **S3 data distribution type**, and **S3 location**, specify the appropriate values. For information about what these values mean, see S3DataSource (p. 994).

vi. For **Input mode**, choose **File** to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose **Pipe**To stream data directly from Amazon S3 to the container.

vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.

g. For **Output** location, specify the following values:

i. For **S3 output path**, choose the S3 location where the training job stores output, such as model artifacts.

> **Note**
> You use the model artifacts stored at this location to create a model or model package from this training job.

ii. For **Encryption key**, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.

h. For **Tags**, specify one or more tags to manage the training job. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.

i. Choose **Create training job** to run the training job.

Use an Algorithm to Run a Training Job (API)

To use an algorithm to run a training job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) as the `AlgorithmName` field of the AlgorithmSpecification (p. 863) object that you pass to CreateTrainingJob (p. 667). For information about training models in Amazon SageMaker, see Train a Model with Amazon SageMaker (p. 4).

Use an Algorithm to Run a Training Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on AWS Marketplace to create a training job, create an `AlgorithmEstimator` object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the `algorithm_arn` argument. Then call the `fit` method of the estimator. For example:

```
from sagemaker import AlgorithmEstimator
data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
            algorithm_arn='arn:aws:sagemaker:us-east-2:012345678901:algorithm/my-
algorithm',
            role='SageMakerRole',
            train_instance_count=1,
            train_instance_type='ml.c4.xlarge',
            sagemaker_session=sagemaker_session,
            base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.fit({'training': train_input})
```

Use an Algorithm to Run a Hyperparameter Tuning Job

A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. For more information, see Automatic Model Tuning (p. 288).

You can create use an algorithm resource to create a hyperparameter tuning job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

Topics

Use an Algorithm to Run a Hyperparameter Tuning Job (Console)

To use an algorithm to run a hyperparameter tuning job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Algorithms**.
3. Choose an algorithm that you created from the list on the **My algorithms** tab or choose an algorithm that you subscribed to on the **AWS Marketplace subscriptions** tab.
4. Choose **Create hyperparameter tuning job**.

    The algorithm you chose will automatically be selected.
5. On the **Create hyperparameter tuning job** page, provide the following information:

    a. For **Warm start**, choose **Enable warm start** to use the information from previous hyperparameter tuning jobs as a starting point for this hyperparameter tuning job. For more information, see Run a Warm Start Hyperparameter Tuning Job (p. 303).

        i. Choose **Identical data and algorithm** if your input data is the same as the input data for the parent jobs of this hyperparameter tuning job, or choose **Transfer learning** to use additional or different input data for this hyperparameter tuning job.

        ii. For **Parent hyperparameter tuning job(s)**, choose up to 5 hyperparameter tuning jobs to use as parents to this hyperparameter tuning job.

    b. For **Hyperparameter tuning job name**, type a name for the tuning job.

    c. For **IAM role**, choose an IAM role that has the required permissions to run hyperparameter tuning jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see Amazon SageMaker Roles (p. 496).

    d. For **VPC**, choose a Amazon VPC that you want to allow the training jobs that the tuning job launches to access. For more information, see Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC (p. 522).

    e. Choose **Next**.

    f. For **Objective metric**, choose the metric that the hyperparameter tuning job uses to determine the best combination of hyperparameters, and choose whether to minimize or maximize this metric. For more information, see View the Best Training Job (p. 301).

g. For **Hyperparameter configuration**, choose ranges for the tunable hyperparameters that you want the tuning job to search, and set static values for hyperparameters that you want to remain constant in all training jobs that the hyperparameter tuning job launches. For more information, see Define Hyperparameter Ranges (p. 292).

h. Choose **Next**.

i. For **Input data configuration**, specify the following values for each channel of input data to use for the hyperparameter tuning job. You can see what channels the algorithm you're using for hyperparameter tuning supports, and the content type, supported compression type, and supported input modes for each channel, under **Channel specification** section of the **Algorithm summary** page for the algorithm.

    i. For **Channel name**, type the name of the input channel.

    ii. For **Content type**, type the content type of the data that the algorithm expects for the channel.

    iii. For **Compression type**, choose the data compression type to use, if any.

    iv. For **Record wrapper**, choose `RecordIO` if the algorithm expects data in the `RecordIO` format.

    v. For **S3 data type**, **S3 data distribution type**, and **S3 location**, specify the appropriate values. For information about what these values mean, see S3DataSource (p. 994).

    vi. For **Input mode**, choose **File** to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose **Pipe**To stream data directly from Amazon S3 to the container.

    vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.

j. For **Output** location, specify the following values:

    i. For **S3 output path**, choose the S3 location where the training jobs that this hyperparameter tuning job launches store output, such as model artifacts.

        Note
        You use the model artifacts stored at this location to create a model or model package from this hyperparameter tuning job.

    ii. For **Encryption key**, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.

k. For **Resource configuration**, provide the following information:

    i. For **Instance type**, choose the instance type to use for each training job that the hyperparameter tuning job launches.

    ii. For **Instance count**, type the number of ML instances to use for each training job that the hyperparameter tuning job launches.

    iii. For **Additional volume per instance (GB)**, type the size of the ML storage volume that you want to provision each training job that the hyperparameter tuning job launches. ML storage volumes store model artifacts and incremental states.

    iv. For **Encryption key**, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instances, specify the key.

l. For **Resource limits**, provide the following information:

    i. For **Maximum training jobs**, specify the maximum number of training jobs that you want the hyperparameter tuning job to launch. A hyperparameter tuning job can launch a maximum of 500 training jobs.

    ii. For **Maximum parallel training jobs**, specify the maximum number of concurrent training jobs that the hyperparameter tuning job can launch. A hyperparameter tuning job can launch a maximum of 10 concurrent training jobs.

iii. For **Stopping condition**, specify the maximum amount of time in seconds, minutes, hours, or days, that you want each training job that the hyperparameter tuning job launches to run.

m. For **Tags**, specify one or more tags to manage the hyperparameter tuning job. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.

n. Choose **Create jobs** to run the hyperparameter tuning job.

Use an Algorithm to Run a Hyperparameter Tuning Job (API)

To use an algorithm to run a hyperparameter tuning job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) of the algorithm as the `AlgorithmName` field of the AlgorithmSpecification (p. 863) object that you pass to CreateHyperParameterTuningJob (p. 638). For information about hyperparameter tuning in Amazon SageMaker, see Automatic Model Tuning (p. 288).

Use an Algorithm to Run a Hyperparameter Tuning Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on AWS Marketplace to create a hyperparameter tuning job, create an `AlgorithmEstimator` object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the `algorithm_arn` argument. Then initialize a `HyperparameterTuner` object with the `AlgorithmEstimator` you created as the value of the `estimator` argument. Finally, call the `fit` method of the `AlgorithmEstimator`. For example:

```
from sagemaker import AlgorithmEstimator
from sagemaker.tuner import HyperparameterTuner

data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
            algorithm_arn='arn:aws:sagemaker:us-east-2:764419575721:algorithm/scikit-
decision-trees-1542410022',
            role='SageMakerRole',
            train_instance_count=1,
            train_instance_type='ml.c4.xlarge',
            sagemaker_session=sagemaker_session,
            base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.set_hyperparameters(max_leaf_nodes=10)
tuner = HyperparameterTuner(estimator=algo, base_tuning_job_name='some-name',
                                objective_metric_name='validation:accuracy',
                                hyperparameter_ranges=hyperparameter_ranges,
                                max_jobs=2, max_parallel_jobs=2)

tuner.fit({'training': train_input}, include_cls_metadata=False)
tuner.wait()
```

Use a Model Package to Create a Model

Use a model package to create a deployable model that you can use to get real-time inferences by creating a hosted endpoint or to run batch transform jobs. You can create a deployable model from a

model package by using the Amazon SageMaker console, the low-level Amazon SageMaker API), or the Amazon SageMaker Python SDK.

Topics

- Use a Model Package to Create a Model (Console) (p. 426)
- Use a Model Package to Create a Model (API) (p. 426)
- Use a Model Package to Create a Model (Amazon SageMaker Python SDK) (p. 427)

Use a Model Package to Create a Model (Console)

To create a deployable model from a model package (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Model packages**.
3. Choose a model package that you created from the list on the **My model packages** tab or choose a model package that you subscribed to on the **AWS Marketplace subscriptions** tab.
4. Choose **Create model**.
5. For **Model name**, type a name for the model.
6. For **IAM role**, choose an IAM role that has the required permissions to call other services on your behalf, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see Amazon SageMaker Roles (p. 496).
7. For **VPC**, choose a Amazon VPC that you want to allow the model to access. For more information, see Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC (p. 525).
8. Leave the default values for **Container input options** and **Choose model package**.
9. For environment variables, provide the names and values of environment variables you want to pass to the model container.
10. For **Tags**, specify one or more tags to manage the model. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.
11. Choose **Create model**.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26). For information about batch transform jobs, see Step 6.2: Deploy the Model with Batch Transform (p. 28).

Use a Model Package to Create a Model (API)

To use a model package to create a deployable model by using the Amazon SageMaker API, specify the name or the Amazon Resource Name (ARN) of the model package as the `ModelPackageName` field of the ContainerDefinition (p. 886) object that you pass to the CreateModel (p. 648) API.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26). For information about batch transform jobs, see Step 6.2: Deploy the Model with Batch Transform (p. 28).

Use a Model Package to Create a Model (Amazon SageMaker Python SDK)

To use a model package to create a deployable model by using the Amazon SageMaker Python SDK, initialize a `ModelPackage` object, and pass the Amazon Resource Name (ARN) of the model package as the `model_package_arn` argument. For example:

```
from sagemaker import ModelPackage
model = ModelPackage(role='SageMakerRole',
                     model_package_arn='training-job-scikit-decision-
trees-1542660466-6f92',
                     sagemaker_session=sagemaker_session)
```

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 26). For information about batch transform jobs, see Step 6.2: Deploy the Model with Batch Transform (p. 28).

Amazon SageMaker Resources in AWS Marketplace

Amazon SageMaker integrates with AWS Marketplace, enabling developers to charge other Amazon SageMaker users for the use of their algorithms and model packages. AWS Marketplace is a curated digital catalog that makes it easy for customers to find, buy, deploy, and manage third-party software and services that customers need to build solutions and run their businesses. AWS Marketplace includes thousands of software listings in popular categories, such as security, networking, storage, machine learning, business intelligence, database, and DevOps. It simplifies software licensing and procurement with flexible pricing options and multiple deployment methods. For information, see AWS Marketplace Documentation.

Topics

Amazon SageMaker Algorithms

An algorithm enables you to perform end-to-end machine learning. It has two logical components: training and inference. Buyers can use the training component to create training jobs in Amazon SageMaker and build a machine learning model. Amazon SageMaker saves the model artifacts generated by the algorithm during training to an Amazon S3 bucket. For more information, see Train a Model with Amazon SageMaker (p. 4).

Buyers use the inference component with the model artifacts generated during a training job to create a deployable model in their Amazon SageMaker account. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see Deploy a Model in Amazon SageMaker (p. 7).

Amazon SageMaker Model Packages

Buyers use a model package to build a deployable model in Amazon SageMaker. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see Deploy a Model in Amazon SageMaker (p. 7). As a seller, you can build your model artifacts by training in Amazon SageMaker, or you can use your own model artifacts from a model that you trained outside of Amazon SageMaker. You can charge buyers for inference.

Sell Amazon SageMaker Algorithms and Model Packages

Selling Amazon SageMaker algorithms and model packages is a three-step process:

1. Develop your algorithm or model, and package it in a Docker container. For information, see Develop Algorithms and Models in Amazon SageMaker (p. 429).

2. Create an algorithm or model package resource in Amazon SageMaker. For information, see Create Algorithm and Model Package Resources (p. 413).

3. Register as a seller on AWS Marketplace and list your algorithm or model package on AWS Marketplace. For information about registering as a seller, see Getting Started as a Seller in the *User Guide for AWS Marketplace Providers*. For information about listing and monetizing your algorithms and model packages, see Listing Algorithms and Model Packages in AWS Marketplace for Machine Learning in the *User Guide for AWS Marketplace Providers*.
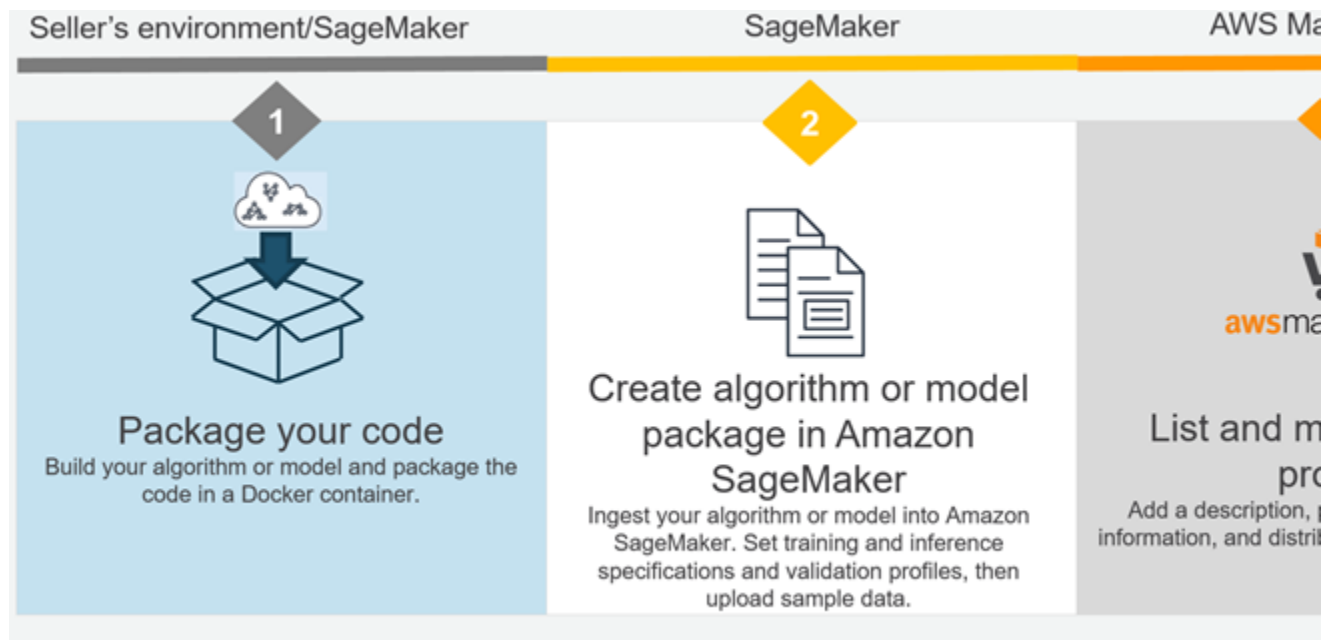
Topics

- Develop Algorithms and Models in Amazon SageMaker (p. 429)
- Create Algorithm and Model Package Resources (p. 413)
- List Your Algorithm or Model Package on AWS Marketplace (p. 431)

Develop Algorithms and Models in Amazon SageMaker

Before you can create algorithm and model package resources to use in Amazon SageMaker or list on AWS Marketplace, you have to develop them and package them in Docker containers.

Note
When algorithms and model packages are created for listing on AWS Marketplace, Amazon SageMaker scans the containers for security vulnerabilities on supported operating systems. Only the following operating system versions are supported:

- Debian: 6.0, 7, 8, 9, 10
- Ubuntu: 12.04, 12.10, 13.04, 14.04, 14.10, 15.04, 15.10, 16.04, 16.10, 17.04, 17.10, 18.04, 18.10
- CentOS: 5, 6, 7
- Oracle Linux: 5, 6, 7
- Alpine: 3.3, 3.4, 3.5
- Amazon Linux

Topics

Develop Algorithms in Amazon SageMaker

An algorithm should be packaged as a docker container and stored in Amazon ECR to use it in Amazon SageMaker. The Docker container contains the training code used to run training jobs and, optionally, the inference code used to get inferences from models trained by using the algorithm.

For information about developing algorithms in Amazon SageMaker and packaging them as containers, see Use Your Own Algorithms or Models with Amazon SageMaker (p. 384). For a complete example of how to create an algorithm container, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb. You can also find the sample notebook in an Amazon SageMaker notebook instance. The notebook is in the **Advanced Functionality** section, and is named `scikit_bring_your_own.ipynb`. For information about using the sample notebooks in a notebook instance, see Use Example Notebooks (p. 42).

Always thoroughly test your algorithms before you create algorithm resources to publish on AWS Marketplace.

Note
When a buyer subscribes to your containerized product, the Docker containers run in an isolated (internet-free) environment. When you create your containers, do not rely on making outgoing calls over the internet. Calls to AWS services are also not allowed.

Develop Models in Amazon SageMaker

A deployable model in Amazon SageMaker consists of inference code, model artifacts, an IAM role that is used to access resources, and other information required to deploy the model in Amazon SageMaker. Model artifacts are the results of training a model by using a machine learning algorithm. The inference code must be packaged in a Docker container and stored in Amazon ECR. You can either package the model artifacts in the same container as the inference code, or store them in Amazon S3.

You create a model by running a training job in Amazon SageMaker, or by training a machine learning algorithm outside of Amazon SageMaker. If you run a training job in Amazon SageMaker, the resulting model artifacts are available in the `ModelArtifacts` field in the response to a call to the DescribeTrainingJob (p. 744) operation. For information about how to develop an Amazon SageMaker model container, see Use Your Own Inference Code (p. 408). For a complete example of how to create

a model container from a model trained outside of Amazon SageMaker, see the sample notebook at
https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/
xgboost_bring_your_own_model/xgboost_bring_your_own_model.ipynb. You can also find the sample
notebook in an Amazon SageMaker notebook instance. The notebook is in the **Advanced Functionality**
section, and is named `xgboost_bring_your_own_model.ipynb`. For information about using the
sample notebooks in a notebook instance, see .

Always thoroughly test your models before you create model packages to publish on AWS Marketplace.

> **Note**
> When a buyer subscribes to your containerized product, the Docker containers run in an isolated
> (internet-free) environment. When you create your containers, do not rely on making outgoing
> calls over the internet. Calls to AWS services are also not allowed.

List Your Algorithm or Model Package on AWS Marketplace

After creating and validating your algorithm or model in Amazon SageMaker, list your product on AWS
Marketplace. The listing process makes your products available in the AWS Marketplace and the Amazon
SageMaker console.

To list products on AWS Marketplace, you must be a registered seller. To register, use the self-registration
process from the AWS Marketplace Management Portal (AMMP). For information, see Getting Started as
a Seller in the *User Guide for AWS Marketplace Providers*. When you start the product listing process from
the Amazon SageMaker console, we check your seller registration status. If you have not registered, we
direct you to do so.

To start the listing process, do one of the following:

- From the Amazon SageMaker console, choose the product, choose **Actions**, and choose **Publish new
  ML Marketplace listing**. This carries over your product reference, the Amazon Resource Name (ARN),
  and directs you to the AMMP to create the listing.

- Go to ML listing process, manually enter the Amazon Resource Name (ARN), and start your product
  listing. This process carries over the product metadata that you entered when creating the product in
  Amazon SageMaker. For an algorithm listing, the information includes the supported instance types
  and hyperparameters. In addition, you can enter a product description, promotional information, and
  support information as you would with other AWS Marketplace products.

Find and Subscribe to Algorithms and Model Packages on AWS Marketplace

With AWS Marketplace, you can browse and search for hundreds of machine learning algorithms and
models in a broad range of categories, such as computer vision, natural language processing, speech
recognition, text, data, voice, image, video analysis, fraud detection, predictive analysis, and more.

Discover algorithms/model packages
AWS Marketplace

Subscribe and configure your
product in AWS Marketplace/
Amazon SageMaker

Deploy on Amazon SageMaker
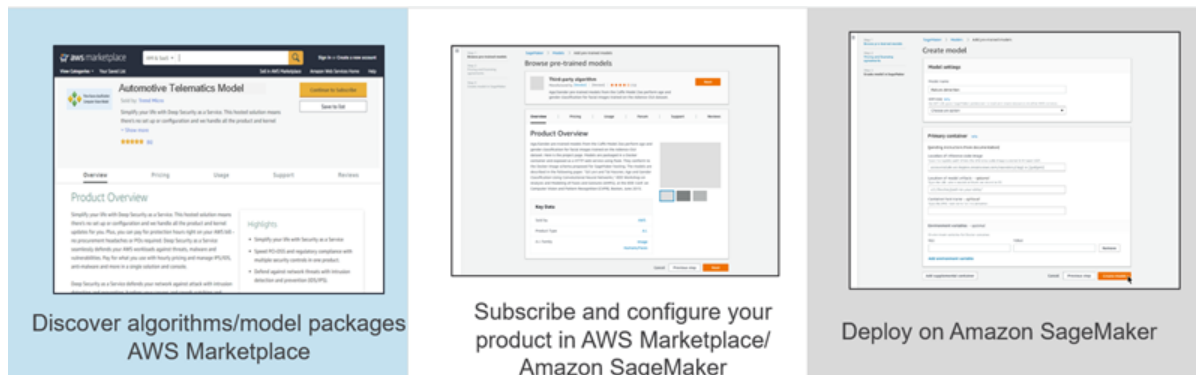
To find algorithms on AWS Marketplace

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Algorithms**, then choose **Find algorithms**.

   This takes you to the AWS Marketplace algorithms page. For information about finding and subscribing to algorithms on AWS Marketplace, see Machine Learning Products in the *AWS Marketplace User Guide for AWS Consumers*.

To find model packages on AWS Marketplace

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Model packages**, then choose **Find model packages**.

   This takes you to the AWS Marketplace model packages page. For information about finding and subscribing to model packages on AWS Marketplace, see Machine Learning Products in the *AWS Marketplace User Guide for AWS Consumers*.

Use Algorithms and Model Packages

For information about using algorithms and model packages that you subscribe to in Amazon SageMaker, see Use Algorithm and Model Package Resources (p. 419).

> **Note**
> When you create a training job, inference endpoint, and batch transform job from an algorithm or model package that you subscribe to on AWS Marketplace, the training and inference containers do not have access to the internet. Because the containers do not have access to the internet, the seller of the algorithm or model package does not have access to your data.

Manage Machine Learning Experiments with Amazon SageMaker Model Tracking Capability

To organize, find, and evaluate machine leaning model experiments, use Amazon SageMaker model tracking capabilities. Developing models typically requires extensive experimenting with different datasets, algorithms, and parameter values. Using the model tracking capability, you can search, filter and sort through hundreds and possibly thousands of experiments using model attributes such as parameters, metrics and tags. This helps you find the best model for your use case quickly.

Amazon SageMaker model tracking capability can be used to:

- Find, organize, or evaluate training jobs using properties, hyperparameters, performance metrics, or any other metadata.
- Find the best performing model by ranking the results of training jobs and models based on metrics, such as training loss or validation accuracy.
- Trace the lineage of a model back to the training job and its related resources, such as the training datasets.

Sample Notebooks that Manage ML Experiments with Amazon SageMaker Model Tracking Capability

For a sample notebook that uses Amazon SageMaker model tracking capability to manage ML experiments, see Managing ML Experimentation using Amazon SageMaker Model Tracking Capability. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 36). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all of the Amazon SageMaker samples. The notebook managing ML experiments is located in the **Advanced Functionality** section. To open a notebook, choose its **Use** tab and choose **Create copy**. If you have questions, post them on our developer forum.

Topics

Use Model Tracking to Find, Organize, and Evaluate Training Jobs (Console)

To create and test a model, you need to conduct experiments. You can perform an experiment to test different algorithms, tune hyperparameters, or use different datasets. Typically, you study the effect of the changes made in these experiments on the performance of a model. You can organize these experiments by tagging them with key/ value pairs that are seaerchable.

To find a specific training job, model, or resource, use Amazon SageMaker model tracking to search on keywords in any items that are searchable. Searchable items include training jobs, models, hyperparameters, metadata, tags, and URLs. You can use model tracking with tags to help organize your training jobs. To refine your tracking results, you can search using multiple criteria.

To choose the optimal model for deployment, you need to evaluate how they performed against one or more metrics. You can use model tracking results to list, sort, and evaluate the performance of the models in your experiments.

Topics
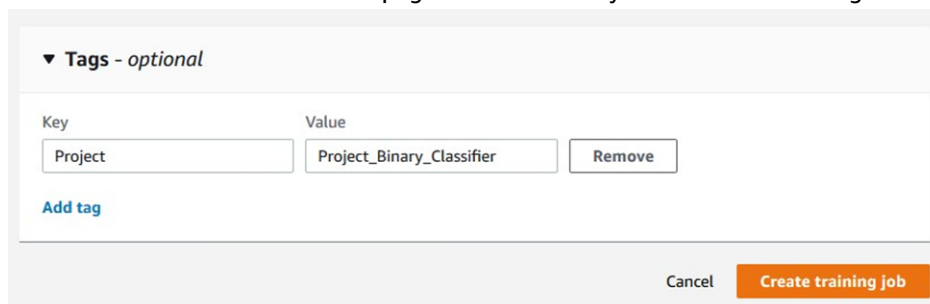
Use Tags to Track Training Jobs (Console)

You can use tags as search criteria. To group training jobs, create tags with descriptive keys and value. For example, create tag keys for: project, owner, customer, and industry.

Add tags to training job and search for taged jobs (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker.

2. In the left navigation pane, choose **Training jobs** and select **Create training job**.

3. Scroll down to the bottom of the page to enter the Key and Value for the tag.

4. To add more tags to the search, choose **Add tag** and add a another tag key-value pair for each new tag that you want to add.

5. After you have trained models that have been tagged, you can search for the models that had them added. In the left navigation pane, choose **Search**.

6. For **Property**, enter a tag key and a tag value.

When you use tags in a search, in the results, the key is a column name and the values are entries in rows.

Find Training Jobs (Console)

To use Amazon SageMaker model tracking capability (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker.
2. In the left navigation pane, choose **Search**.
3. For **Resource type**, choose **Training jobs**.
4. To add a parameter, for **Parameters**, provide the following:

   a. Enter a parameter in the search box and choose a parameter type, for example **TrainingJobName**.

   b. Choose the conditional operation to use. This sets the condition that values must satisfy to be included in a search result. For numeral values, use operators such as **is equals to**, **lesser than**, or **or greater than**. For text-based values use operators, such as **equals to** or **contains**.

   c. Enter the value for the parameter.
5. (optional) To refine your search, add additional search criteria, choose **Add parameter** and enter the parameter values. When you provide multiple parameters, Amazon SageMaker includes all parameters in the search.
6. Choose **Search**.

Evaluate Models Returned by a Search (Console)

To evaluate different models, find their metrics with a search. To highlight metrics, adjust the view to show only metrics and important hyperparameters.

To change the viewable metadata, hyperparameters, and metrics:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker.
2. In the left navigation pane, choose **Search** and run a search on training jobs for relevant parameters. The results are displayed in a table.

Results: Training jobs

▼	HyperParameter mini_batch_size ▼	HyperParameter predictor_type ▼	Metric train:binary_f_beta ▼	Metric train:progress ▼	Metric train:objective_los
	300	binary_classifier	0.966639518737793	100	0.0238142367452⋮
	100	binary_classifier	0.9652714133262634	100	0.0235049128532⋮
	200	binary_classifier	0.9647442698478699	100	0.0232598073780⋮

3. After performing a search, choose the cog icon at the search results table to show the preferences window.

4. To show or hide a **Hyperparameter** or **Metric**, use its toggle switch.

5. To update the view after making changes, choose **Update view**.

After viewing metrics and important hyperparameters, you can compare and contrast the result. From there, you can choose the best model to host or investigate the models that are performing poorly.

Use Model Tracking to Find and Evaluate Training Jobs (API)

You can also use Search (p. 819) to the find and evaluate training jobs or to get suggestions for items used in experiments that are searchable.

Topics

Use Search to Find Training Jobs Tagged with Specific Values (API)

To use Amazon SageMaker model tracking capability, create a search parameter, `search_params`, then use the search function found in the `smclient` of the AWS SDK for Python (Boto 3).

The following example shows how to use search using the API:

```
import boto3

search_params={
    "MaxResults": 10,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [{
              "Name": "Tags.Project",
              "Operator": "Equals",
              "Value": "Project_Binary_Classifier"
          }]},
  "SortBy": "Metrics.train:binary_classification_accuracy",
  "SortOrder": "Descending"
}

smclient = boto3.client(service_name='sagemaker')
results = smclient.search(**search_params)
```

Evaluate Models (API)

To evaluate different models, see the metrics in a search result. To evaluate models using the AWS SDK for Python (Boto 3), create a table and plot it.

The following example shows how to use model tracking capability to evaluate models and to display the results in a table:

```
import pandas

headers=["Training Job Name", "Training Job Status", "Batch Size", "Binary Classification
 Accuracy"]
rows=[]
for result in results['Results']:
    trainingJob = result['TrainingJob']
    metrics = trainingJob['FinalMetricDataList']
    rows.append([trainingJob['TrainingJobName'],
     trainingJob['TrainingJobStatus'],
     trainingJob['HyperParameters']['mini_batch_size'],
     metrics[[x['MetricName'] for x in
     metrics].index('train:binary_classification_accuracy')]['Value']
    ])

df = pandas.DataFrame(data=rows,columns=headers)

from IPython.display import display, HTMLdisplay(HTML(df.to_html()))
```

Get Suggestions for a Search (API)

To get suggestions for a search, use GetSearchSuggestions (p. 759) in the API.

The following code example for AWS SDK for Python (Boto 3) shows a `get_search_suggestions` request for items containing "linear":

```
search_suggestion_params={
    "Resource": "TrainingJob",
    "SuggestionQuery": {
        "PropertyNameQuery": {
           "PropertyNameHint": "linear"
        }
    }
}
```

```
}
```

The following code shows an example of an expected response for `get_search_suggestions`:

```
{
        'PropertyNameSuggestions': [{'PropertyName': 'hyperparameters.linear_init_method'},
                    {'PropertyName': 'hyperparameters.linear_init_value'},
                    {'PropertyName': 'hyperparameters.linear_init_sigma'},
                    {'PropertyName': 'hyperparameters.linear_lr'},
                    {'PropertyName': 'hyperparameters.linear_wd'}]
}
```

After you get the search suggestions, you can use one of the property names in a search.

Verify the Contents of Your Training Jobs

You can use Amazon SageMaker model tracking capability to verify which datasets were used in training, where the holdout datasets were used, and other details about training jobs. Use it, for example, if you need to verify that a dataset was used in a training job for an audit or to verify compliance.

To check if a holdout dataset or any other dataset was used in a training job, search for its Amazon S3 URL. Amazon SageMaker model tracking uses the dataset as a search parameter and lists the training jobs that used the dataset. If your search result is empty, it means that the dataset was not used in a training job.

Trace the Lineage of your Models

You can use Amazon SageMaker model tracking to trace information about the lineage of training jobs and model resources related to it, including the dataset, algorithm, hyperparameters, and metrics used. For example, if you find that the performance of a hosted model has declined, you can review its training job and the resources it used to determine what is causing the problem. This investigation can be done from the console or by using the API.

Use Single-click on the Amazon SageMaker Console to Trace the Lineage of Your Models (Console)

In the left navigation pane of the Amazon SageMaker, choose **Endpoints**, and select the relevant endpoint from the list of your deployed endpoints. Scroll to **Endpoint Configuration Settings**, which lists all the model versions deployed at the endpoint. Here you have access to a hyperlink to the Model Training Job that created that model in the first place. For an example that used a linear-learner model, you would see:

Production variants

Model name	Training job	Variant name	Instance type
linear-learner-2018-█████	linear-learner-2018-█████	AllTraffic	ml.m4.xlarge

Use Code to Trace the Lineage of Your Models (API)

To trace a model's lineage, you need to obtain the model's name, then use it to search for training jobs.

The following example shows how to use model tracking capability to trace a model's lineage using the API:

```
# Get the name of model deployed at endpoint
endpoint_config = smclient.describe_endpoint_config(EndpointConfigName=endpointName)
model_name = endpoint_config['ProductionVariants'][0]['ModelName']

# Get the model's name
model = smclient.describe_model(ModelName=model_name)

# Search the training job by Amazon S3 location of model artifacts
search_params={
    "MaxResults": 1,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "ModelArtifacts.S3ModelArtifacts",
                "Operator": "Equals",
                "Value": model['PrimaryContainer']['ModelDataUrl']
            }]},
}
results = smclient.search(**search_params)
```

After you find your training job, you can investigate the resources used to train the model.

Use Machine Learning Frameworks with Amazon SageMaker

The Amazon SageMaker Python SDK provides open source APIs and containers that make it easy to train and deploy models in Amazon SageMaker with several different machine learning and deep learning frameworks. For general information about the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk. For information about using specific frameworks in Amazon SageMaker, see the following topics:

Topics

Use Apache Spark with Amazon SageMaker

This section provides information for developers who want to use Apache Spark for preprocessing data and Amazon SageMaker for model training and hosting. For information about supported versions of Apache Spark, see https://github.com/aws/sagemaker-spark#getting-sagemaker-spark.

Amazon SageMaker provides an Apache Spark library, in both Python and Scala, that you can use to easily train models in Amazon SageMaker using `org.apache.spark.sql.DataFrame` data frames in your Spark clusters. After model training, you can also host the model using Amazon SageMaker hosting services.

The Amazon SageMaker Spark library, `com.amazonaws.services.sagemaker.sparksdk`, provides the following classes, among others:

- `SageMakerEstimator`—Extends the `org.apache.spark.ml.Estimator` interface. You can use this estimator for model training in Amazon SageMaker.
- `KMeansSageMakerEstimator`, `PCASageMakerEstimator`, and `XGBoostSageMakerEstimator`—Extend the `SageMakerEstimator` class.
- `SageMakerModel`—Extends the `org.apache.spark.ml.Model` class. You can use this `SageMakerModel` for model hosting and obtaining inferences in Amazon SageMaker.

Download the Amazon SageMaker Spark Library

You have the following options for downloading the Spark library provided by Amazon SageMaker:

- You can download the source code for both PySpark and Scala libraries from GitHub at https://github.com/aws/sagemaker-spark.

- For the Python Spark library, you have the following additional options:
  - Use pip install:

    ```
    $ pip install sagemaker_pyspark
    ```

  - In a notebook instance, create a new notebook that uses either the `Sparkmagic (PySpark)` or the `Sparkmagic (PySpark3)` kernel and connect to a remote Amazon EMR cluster. For more information, see Build Amazon SageMaker Notebooks Backed by Spark in Amazon EMR.

    Note
    The EMR cluster must be configured with an IAM role that has the `AmazonSageMakerFullAccess` policy attached. For information about configuring roles for an EMR cluster, see Configure IAM Roles for Amazon EMR Permissions to AWS Services in the *Amazon EMR Management Guide*.

- You can get the Scala library from Maven. Add the Spark library to your project by adding the following dependency to your `pom.xml` file:

  ```
  <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>sagemaker-spark_2.11</artifactId>
      <version>spark_2.2.0-1.0</version>
  </dependency>
  ```

Integrate Your Apache Spark Application with Amazon SageMaker

The following is high-level summary of the steps for integrating your Apache Spark application with Amazon SageMaker.

1. Continue data preprocessing using the Apache Spark library that you are familiar with. Your dataset remains a `DataFrame` in your Spark cluster.

   Note
   Load your data into a `DataFrame` and preprocess it so that you have a `features` column with `org.apache.spark.ml.linalg.Vector` of `Doubles`, and an optional `label` column with values of `Double` type.

2. Use the estimator in the Amazon SageMaker Spark library to train your model. For example, if you choose the k-means algorithm provided by Amazon SageMaker for model training, you call the `KMeansSageMakerEstimator.fit` method.

   Provide your `DataFrame` as input. The estimator returns a `SageMakerModel` object.

   Note
   SageMakerModel extends the `org.apache.spark.ml.Model`.

   The `fit` method does the following:

   a. Converts the input `DataFrame` to the protobuf format by selecting the `features` and `label` columns from the input `DataFrame` and uploading the protobuf data to an Amazon S3 bucket. The protobuf format is efficient for model training in Amazon SageMaker.

    b.   Starts model training in Amazon SageMaker by sending an Amazon SageMaker
CreateTrainingJob (p. 667) request. After model training has completed, Amazon SageMaker
saves the model artifacts to an S3 bucket.

           Amazon SageMaker assumes the IAM role that you specified for model training to perform tasks
on your behalf. For example, it uses the role to read training data from an S3 bucket and to
write model artifacts to a bucket.

    c.   Creates and returns a `SageMakerModel` object. The constructor does the following tasks, which
are related to deploying your model to Amazon SageMaker.

        i.   Sends a CreateModel (p. 648) request to Amazon SageMaker.

        ii.   Sends a CreateEndpointConfig (p. 635) request to Amazon SageMaker.

        iii.   Sends a CreateEndpoint (p. 632) request to Amazon SageMaker, which then launches the
specified resources, and hosts the model on them.

3.   You can get inferences from your model hosted in Amazon SageMaker with the
`SageMakerModel.transform`.

    Provide an input `DataFrame` with features as input. The `transform` method transforms it to
a `DataFrame` containing inferences. Internally, the `transform` method sends a request to the
InvokeEndpoint (p. 853) Amazon SageMaker API to get inferences. The `transform` method
appends the inferences to the input `DataFrame`.

Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark

Topics

- Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache
Spark (p. 446)
- Use the SageMakerEstimator in a Spark Pipeline (p. 447)

Amazon SageMaker provides an Apache Spark library (in both Python and Scala) that you can use
to integrate your Apache Spark applications with Amazon SageMaker. For example, you might use
Apache Spark for data preprocessing and Amazon SageMaker for model training and hosting. For more
information, see Use Apache Spark with Amazon SageMaker (p. 440). This section provides example
code that uses the Apache Spark Scala library provided by Amazon SageMaker to train a model in
Amazon SageMaker using `DataFrames` in your Spark cluster. The example also hosts the resulting model
artifacts using Amazon SageMaker hosting services. Specifically, this example does the following:

- Uses the `KMeansSageMakerEstimator` to fit (or train) a model on data

    Because the example uses the k-means algorithm provided by Amazon SageMaker to train a model,
you use the `KMeansSageMakerEstimator`. You train the model using images of handwritten single-
digit numbers (from the MNIST dataset). You provide the images as an input `DataFrame`. For your
convenience, Amazon SageMaker provides this dataset in an S3 bucket.

    In response, the estimator returns a `SageMakerModel` object.

- Obtains inferences using the trained `SageMakerModel`

To get inferences from a model hosted in Amazon SageMaker, you call the
`SageMakerModel.transform` method. You pass a `DataFrame` as input. The method transforms the
input `DataFrame` to another `DataFrame` containing inferences obtained from the model.

For a given input image of a handwritten single-digit number, the inference identifies a cluster that the
image belongs to. For more information, see K-Means Algorithm (p. 141).

This is the example code:

```
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

val roleArn = "arn:aws:iam::account-id:role/rolename"

val estimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(784)

// train
val model = estimator.fit(trainingData)

val transformedData = model.transform(testData)
transformedData.show
```

The code does the following:

- Loads the MNIST dataset from an S3 bucket provided by Amazon SageMaker (`awsai-sparksdk-dataset`) into a Spark `DataFrame` (`mnistTrainingDataFrame`):

```
// Get a Spark session.

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
```

```
    .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

val roleArn = "arn:aws:iam::account-id:role/rolename"
trainingData.show()
```

The `show` method displays the first 20 rows in the data frame:

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  5.0|(784,[152,153,154...|
|  0.0|(784,[127,128,129...|
|  4.0|(784,[160,161,162...|
|  1.0|(784,[158,159,160...|
|  9.0|(784,[208,209,210...|
|  2.0|(784,[155,156,157...|
|  1.0|(784,[124,125,126...|
|  3.0|(784,[151,152,153...|
|  1.0|(784,[152,153,154...|
|  4.0|(784,[134,135,161...|
|  3.0|(784,[123,124,125...|
|  5.0|(784,[216,217,218...|
|  3.0|(784,[143,144,145...|
|  6.0|(784,[72,73,74,99...|
|  1.0|(784,[151,152,153...|
|  7.0|(784,[211,212,213...|
|  2.0|(784,[151,152,153...|
|  8.0|(784,[159,160,161...|
|  6.0|(784,[100,101,102...|
|  9.0|(784,[209,210,211...|
+-----+--------------------+
only showing top 20 rows
```

In each row:

- The `label` column identifies the image's label. For example, if the image of the handwritten number is the digit 5, the label value is 5.

- The `features` column stores a vector (`org.apache.spark.ml.linalg.Vector`) of `Double` values. These are the 784 features of the handwritten number. (Each handwritten number is a 28 x 28-pixel image, making 784 features.)

- Creates an Amazon SageMaker estimator (`KMeansSageMakerEstimator`)

  The `fit` method of this estimator uses the k-means algorithm provided by Amazon SageMaker to train models using an input `DataFrame`. In response, it returns a `SageMakerModel` object that you can use to get inferences.

  > **Note**
  > The `KMeansSageMakerEstimator` extends the Amazon SageMaker `SageMakerEstimator`, which extends the Apache Spark `Estimator`.

```
val estimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(784)
```

The constructor parameters provide information that is used for training a model and deploying it on Amazon SageMaker:

- `trainingInstanceType` and `trainingInstanceCount`—Identify the type and number of ML compute instances to use for model training.

- `endpointInstanceType`—Identifies the ML compute instance type to use when hosting the model in Amazon SageMaker. By default, one ML compute instance is assumed.

- `endpointInitialInstanceCount`—Identifies the number of ML compute instances initially backing the endpoint hosting the model in Amazon SageMaker.

- `sagemakerRole`—Amazon SageMaker assumes this IAM role to perform tasks on your behalf. For example, for model training, it reads data from S3 and writes training results (model artifacts) to S3.

  Note
  This example implicitly creates an Amazon SageMaker client. To create this client, you must provide your credentials. The API uses these credentials to authenticate requests to Amazon SageMaker. For example, it uses the credentials to authenticate requests to create a training job and API calls for deploying the model using Amazon SageMaker hosting services.

- After the `KMeansSageMakerEstimator` object has been created, you set the following parameters, are used in model training:

  - The number of clusters that the k-means algorithm should create during model training. You specify 10 clusters, one for each digit, 0 through 9.

  - Identifies that each input image has 784 features (each handwritten number is a 28 x 28-pixel image, making 784 features).

- Calls the estimator `fit` method

```
// train
val model = estimator.fit(trainingData)
```

You pass the input `DataFrame` as a parameter. The model does all the work of training the model and deploying it to Amazon SageMaker. For more information see, . In response, you get a `SageMakerModel` object, which you can use to get inferences from your model deployed in Amazon SageMaker.

You provide only the input `DataFrame`. You don't need to specify the registry path to the k-means algorithm used for model training because the `KMeansSageMakerEstimator` knows it.

- Calls the `SageMakerModel.transform` method to get inferences from the model deployed in Amazon SageMaker.

The `transform` method takes a `DataFrame` as input, transforms it, and returns another `DataFrame` containing inferences obtained from the model.

```
val transformedData = model.transform(testData)
transformedData.show
```

For simplicity, we use the same `DataFrame` as input to the `transform` method that we used for model training in this example. The `transform` method does the following:

- Serializes the `features` column in the input `DataFrame` to protobuf and sends it to the Amazon SageMaker endpoint for inference.

- Deserializes the protobuf response into the two additional columns (`distance_to_cluster` and `closest_cluster`) in the transformed `DataFrame`.

The `show` method gets inferences to the first 20 rows in the input `DataFrame`:

```
+-----+--------------------+-------------------+---------------+
|label|            features|distance_to_cluster|closest_cluster|
+-----+--------------------+-------------------+---------------+
|  5.0|(784,[152,153,154...|  1767.897705078125|            4.0|
|  0.0|(784,[127,128,129...|  1392.157470703125|            5.0|
|  4.0|(784,[160,161,162...| 1671.5711669921875|            9.0|
|  1.0|(784,[158,159,160...| 1182.6082763671875|            6.0|
|  9.0|(784,[208,209,210...| 1390.4002685546875|            0.0|
|  2.0|(784,[155,156,157...|  1713.988037109375|            1.0|
|  1.0|(784,[124,125,126...| 1246.3016357421875|            2.0|
|  3.0|(784,[151,152,153...|  1753.229248046875|            4.0|
|  1.0|(784,[152,153,154...|  978.8394165039062|            2.0|
|  4.0|(784,[134,135,161...|  1623.176513671875|            3.0|
|  3.0|(784,[123,124,125...|  1533.863525390625|            4.0|
|  5.0|(784,[216,217,218...|  1469.357177734375|            6.0|
|  3.0|(784,[143,144,145...|  1736.765869140625|            4.0|
|  6.0|(784,[72,73,74,99...|   1473.69384765625|            8.0|
|  1.0|(784,[151,152,153...|   944.88720703125|            2.0|
|  7.0|(784,[211,212,213...| 1285.9071044921875|            3.0|
|  2.0|(784,[151,152,153...| 1635.0125732421875|            1.0|
|  8.0|(784,[159,160,161...| 1436.3162841796875|            6.0|
|  6.0|(784,[100,101,102...| 1499.7366943359375|            7.0|
|  9.0|(784,[209,210,211...| 1364.6319580078125|            6.0|
+-----+--------------------+-------------------+---------------+
```

You can interpret the data, as follows:

- A handwritten number with the `label` 5 belongs to cluster 5 (`closest_cluster`).

- A handwritten number with the `label` 0 belongs to cluster 2.

- A handwritten number with the `label` 4 belongs to cluster 4.

- A handwritten number with the `label` 1 belongs to cluster 1.

For more information on how to run these examples, see https://github.com/aws/sagemaker-spark/blob/master/README.md on GitHub.

Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark

In Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark (p. 442), you use the `kMeansSageMakerEstimator` because the example uses the k-means algorithm provided by Amazon SageMaker for model training. You might choose to use your own custom algorithm for model training instead. Assuming that you have already created a Docker image, you can create your own `SageMakerEstimator` and specify the Amazon Elastic Container Registry path for your custom image.

The following example shows how to create a `KMeansSageMakerEstimator` from the `SageMakerEstimator`. In the new estimator, you explicitly specify the Docker registry path to your training and inference code images.

```
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.SageMakerEstimator
import
 com.amazonaws.services.sagemaker.sparksdk.transformation.serializers.ProtobufRequestRowSerializer
import
 com.amazonaws.services.sagemaker.sparksdk.transformation.deserializers.KMeansProtobufResponseRowDeseri

val estimator = new SageMakerEstimator(
  trainingImage =
    "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  modelImage =
    "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  requestRowSerializer = new ProtobufRequestRowSerializer(),
  responseRowDeserializer = new KMeansProtobufResponseRowDeserializer(),
  hyperParameters = Map("k" -> "10", "feature_dim" -> "784"),
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1,
  trainingSparkDataFormat = "sagemaker")
```

In the code, the parameters in the `SageMakerEstimator` constructor include:

- `trainingImage` —Identifies the Docker registry path to the training image containing your custom code.
- `modelImage` —Identifies the Docker registry path to the image containing inference code.
- `requestRowSerializer` —Implements `com.amazonaws.services.sagemaker.sparksdk.transformation.RequestRowSerializer`.

  This parameter serializes rows in the input `DataFrame` to send them to the model hosted in Amazon SageMaker for inference.
- `responseRowDeserializer` —Implements

  `com.amazonaws.services.sagemaker.sparksdk.transformation.ResponseRowDeserializer`.

  This parameter deserializes responses from the model, hosted in Amazon SageMaker, back into a `DataFrame`.
- `trainingSparkDataFormat` —Specifies the data format that Spark uses when uploading training data from a `DataFrame` to S3. For example, `"sagemaker"` for protobuf format, `"csv"` for comma-separated values, and `"libsvm"` for LibSVM format.

You can implement your own `RequestRowSerializer` and `ResponseRowDeserializer` to serialize and deserialize rows from a data format that your inference code supports, such as .libsvm or ..csv.

Use the SageMakerEstimator in a Spark Pipeline

You can use `org.apache.spark.ml.Estimator` estimators and `org.apache.spark.ml.Model` models, and `SageMakerEstimator` estimators and `SageMakerModel` models in `org.apache.spark.ml.Pipeline` pipelines, as shown in the following example:

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.PCA
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator
```

```
val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

// substitute your SageMaker IAM role here
val roleArn = "arn:aws:iam::account-id:role/rolename"

val pcaEstimator = new PCA()
  .setInputCol("features")
  .setOutputCol("projectedFeatures")
  .setK(50)

val kMeansSageMakerEstimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(integTestingRole),
  requestRowSerializer =
    new ProtobufRequestRowSerializer(featuresColumnName = "projectedFeatures"),
  trainingSparkDataFormatOptions = Map("featuresColumnName" -> "projectedFeatures"),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(50)

val pipeline = new Pipeline().setStages(Array(pcaEstimator, kMeansSageMakerEstimator))

// train
val pipelineModel = pipeline.fit(trainingData)

val transformedData = pipelineModel.transform(testData)
transformedData.show()
```

The parameter `trainingSparkDataFormatOptions` configures Spark to serialize to protobuf the "projectedFeatures" column for model training. Additionally, Spark serializes to protobuf the "label" column by default.

Because we want to make inferences using the "projectedFeatures" column, we pass the column name into the `ProtobufRequestRowSerializer`.

The following example shows a transformed `DataFrame`:

```
+-----+--------------------+--------------------+------------------+---------------+
|label|            features|   projectedFeatures|distance_to_cluster|closest_cluster|
+-----+--------------------+--------------------+------------------+---------------+
|  5.0|(784,[152,153,154...|[880.731433034386...|       1500.470703125|            0.0|
|  0.0|(784,[127,128,129...|[1768.51722024166...|        1142.18359375|            4.0|
|  4.0|(784,[160,161,162...|[704.949236329314...|   1386.246826171875|            9.0|
|  1.0|(784,[158,159,160...|[-42.328192193771...|   1277.0736083984375|            5.0|
|  9.0|(784,[208,209,210...|[374.043902028333...|     1211.00927734375|            3.0|
|  2.0|(784,[155,156,157...|[941.267714528850...|   1496.157958984375|            8.0|
|  1.0|(784,[124,125,126...|[30.2848596410594...|   1327.6766357421875|            5.0|
|  3.0|(784,[151,152,153...|[1270.14374062052...|   1570.7674560546875|            0.0|
|  1.0|(784,[152,153,154...|[-112.10792566485...|       1037.568359375|            5.0|
|  4.0|(784,[134,135,161...|[452.068280676606...|   1165.1236572265625|            3.0|
|  3.0|(784,[123,124,125...|[610.596447285397...|    1325.953369140625|            7.0|
|  5.0|(784,[216,217,218...|[142.959601818422...|   1353.4930419921875|            5.0|
|  3.0|(784,[143,144,145...|[1036.71862533658...|   1460.4315185546875|            7.0|
|  6.0|(784,[72,73,74,99...|[996.740157435754...|   1159.8631591796875|            2.0|
```

```
|   1.0|(784,[151,152,153...|[-107.26076167417...|    960.963623046875|           5.0|
|   7.0|(784,[211,212,213...|[619.771820430940...|    1245.13623046875|           6.0|
|   2.0|(784,[151,152,153...|[850.152101817161...|   1304.437744140625|           8.0|
|   8.0|(784,[159,160,161...|[370.041887230547...|  1192.4781494140625|           0.0|
|   6.0|(784,[100,101,102...|[546.674328209335...|     1277.0908203125|           2.0|
|   9.0|(784,[209,210,211...|[-29.259112927426...|  1245.8182373046875|           6.0|
+-----+--------------------+--------------------+------------------+--------------+
```

Additional Examples: Use Amazon SageMaker with Apache Spark

Additional examples of using Amazon SageMaker with Apache Spark are available at https://github.com/aws/sagemaker-spark/tree/master/examples.

Use TensorFlow with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom TensorFlow code. The Amazon SageMaker Python SDK TensorFlow estimators and models and the Amazon SageMaker open-source TensorFlow containers make writing a TensorFlow script and running it in Amazon SageMaker easier.

Use TensorFlow Version 1.11 and Later

For TensorFlow versions 1.11 and later, the Amazon SageMaker Python SDK supports script mode training scripts.

What do you want to do?

I want to train a custom TensorFlow model in Amazon SageMaker.

> For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/tensorflow_distributed_mnist.

> For documentation, see Train a Model with TensorFlow.

I have a TensorFlow model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

> Deploy TensorFlow Serving models.

I have a TensorFlow model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

> Deploying directly from model artifacts.

I want to see the API documentation for Amazon SageMaker Python SDK TensorFlow classes.

> TensorFlow Estimator

I want to see information about Amazon SageMaker TensorFlow containers.

> https://github.com/aws/sagemaker-tensorflow-container.

For general information about writing TensorFlow script mode training scripts and using TensorFlow script mode estimators and models with Amazon SageMaker, see Using TensorFlow with the SageMaker Python SDK.

For information about TensorFlow versions supported by the Amazon SageMaker TensorFlow container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

Use TensorFlow Legacy Mode for Versions 1.11 and Earlier

The Amazon SageMaker Python SDK provides a legacy mode that supports TensorFlow versions 1.11 and earlier. Use legacy mode TensorFlow training scripts to run TensorFlow jobs in Amazon SageMaker if:

- You have existing legacy mode scripts that you do not want to convert to script mode.
- You want to use a TensorFlow version earlier than 1.11.

For information about writing legacy mode TensorFlow scipts to use with the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk/tree/v1.12.0/src/sagemaker/tensorflow#tensorflow-sagemaker-estimators-and-models.

Use Apache MXNet with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom MXNet code. The Amazon SageMaker Python SDK MXNet estimators and models and the Amazon SageMaker open-source MXNet container make writing a MXNet script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom MXNet model in Amazon SageMaker.

> For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/mxnet_mnist.

> For documentation, see Train a Model with MXNet.

I have an MXNet model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

> Deploy MXNet models.

I have an MXNet model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

> Deploy Endpoints from Model Data.

I want to see the API documentation for Amazon SageMaker Python SDK MXNet classes.

> MXNet Classes

I want to see information about Amazon SageMaker MXNet containers.

> https://github.com/aws/sagemaker-mxnet-container.

For general information about writing MXNet script mode training scripts and using MXNet script mode estimators and models with Amazon SageMaker, see Using MXNet with the SageMaker Python SDK.

For information about MXNet versions supported by the Amazon SageMaker MXNet container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst.

Use Scikit-learn with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Scikit-learn code. The Amazon SageMaker Python SDK Scikit-learn estimators and models and the Amazon SageMaker open-source Scikit-learn container make writing a Scikit-learn script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom Scikit-learn model in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/scikit_learn_iris.

For documentation, see Train a Model with Scikit-learn.

I have a Scikit-learn model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

Deploy Scikit-learn models.

I have a Scikit-learn model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

Deploy Endpoints from Model Data.

I want to see the API documentation for Amazon SageMaker Python SDK Scikit-learn classes.

Scikit-learn Classes

I want to see information about Amazon SageMaker Scikit-learn containers.

https://github.com/aws/sagemaker-scikit-learn-container.

For general information about writing Scikit-learn training scripts and using Scikit-learn estimators and models with Amazon SageMaker, see Using Scikit-learn with the SageMaker Python SDK.

For information about Scikit-learn versions supported by the Amazon SageMaker Scikit-learn container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/sklearn/README.rst.

Use PyTorch with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom PyTorch code. The Amazon SageMaker Python SDK PyTorch estimators and models and the Amazon SageMaker open-source PyTorch container make writing a PyTorch script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom PyTorch model in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/pytorch_mnist.

For documentation, see Train a Model with PyTorch.

I have a PyTorch model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

> Deploy PyTorch models.

I have a PyTorch model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

> Deploy Endpoints from Model Data.

I want to see the API documentation for Amazon SageMaker Python SDK PyTorch classes.

> PyTorch Classes

I want to see information about Amazon SageMaker PyTorch containers.

> https://github.com/aws/sagemaker-pytorch-container.

For general information about writing PyTorch training scripts and using PyTorch estimators and models with Amazon SageMaker, see Using PyTorch with the SageMaker Python SDK.

For information about PyTorch versions supported by the Amazon SageMaker PyTorch container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/pytorch/README.rst.

Use Chainer with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Chainer code. The Amazon SageMaker Python SDK Chainer estimators and models and the Amazon SageMaker open-source Chainer container make writing a Chainer script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom Chainer model in Amazon SageMaker.

> For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/chainer_mnist.

> For documentation, see Train a Model with Chainer.

I have a Chainer model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

> Deploy Chainer models.

I have a Chainer model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

> Deploy Endpoints from Model Data.

I want to see the API documentation for Amazon SageMaker Python SDK Chainer classes.

> Chainer Classes

I want to see information about Amazon SageMaker Chainer containers.

> https://github.com/aws/sagemaker-chainer-container.

For general information about writing Chainer training scripts and using Chainer estimators and models with Amazon SageMaker, see Using Chainer with the SageMaker Python SDK.

For information about Chainer versions supported by the Amazon SageMaker Chainer container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/chainer/README.rst.

Use SparkML Serving with Amazon SageMaker

The Amazon SageMaker Python SDK SparkML Serving model and predictor and the Amazon SageMaker open-source SparkML Serving container support deploying Apache Spark ML pipelines serialized with MLeap in Amazon SageMaker to get inferences.

For information about using the SparkML Serving container to deploy models to Amazon SageMaker, see https://github.com/aws/sagemaker-sparkml-serving-container. For information about the Amazon SageMaker Python SDK SparkML Serving model and predictors, see https://sagemaker.readthedocs.io/en/stable/sagemaker.sparkml.html.

Reinforcement Learning with Amazon SageMaker RL

Reinforcement learning (RL) is a machine learning technique that attempts to learn a strategy, called a policy, that optimizes an objective for an agent acting in an environment. For example, the agent might be a robot, the environment might be a maze, and the goal might be to successfully navigate the maze in the smallest amount of time. In RL, the agent takes an action, observes the state of the environment, and gets a reward based on the value of the current state of the environment. The goal is to maximize the long-term reward that the agent receives as a result of its actions. RL is well-suited for solving problems where an agent can make autonomous decisions.

Topics

Why is Reinforcement Learning Important?

RL is well-suited for solving large, complex problems. For example, supply chain management, HVAC systems, industrial robotics, game artificial intelligence, dialog systems, and autonomous vehicles. Because RL models learn by a continuous process of receiving rewards and punishments for every action taken by the agent, it is possible to train systems to make decisions under uncertainty and in dynamic environments.

Markov Decision Process (MDP)

RL is based on models called Markov Decision Processes (MDPs). An MDP consists of a series of time steps. Each time step consists of the following:

Environment

Defines the space in which the RL model operates. This can be either a real-world environment or a simulator. For example, if you train a physical autonomous vehicle on a physical road, that would be a real-world environment. If you train a computer program that models an autonomous vehicle driving on a road, that would be a simulator.

State

Specifies all information about the environment and past steps that is relevant to the future. For example, in an RL model in which a robot can move in any direction at any time step, then the

position of the robot at the current time step is the state, because if we know where the robot is, it isn't necessary to know the steps it took to get there.

Action

What the agent does. For example, the robot takes a step forward.

Reward

A number that represents the value of the state that resulted from the last action that the agent took. For example, if the goal is for a robot to find treasure, the reward for finding treasure might be 5, and the reward for not finding treasure might be 0. The RL model attempts to find a strategy that optimizes the cumulative reward over the long term. This strategy is called a *policy*.

Observation

Information about the state of the environment that is available to the agent at each step. This might be the entire state, or it might be just a part of the state. For example, the agent in a chess-playing model would be able to observe the entire state of the board at any step, but a robot in a maze might only be able to observe a small portion of the maze that it currently occupies.

Typically, training in RL consists of many *episodes*. An episode consists of all of the time steps in an MDP from the initial state until the environment reaches the terminal state.

Key Features of Amazon SageMaker RL

To train RL models in Amazon SageMaker RL, use the following components:

- A deep learning (DL) framework. Currently, Amazon SageMaker supports RL in TensorFlow and Apache MXNet.
- An RL toolkit. An RL toolkit manages the interaction between the agent and the environment, and provides a wide selection of state of the art RL algorithms. Amazon SageMaker supports the Intel Coach and Ray RLlib toolkits. For information about Intel Coach, see https://nervanasystems.github.io/coach/. For information about Ray RLlib, see https://ray.readthedocs.io/en/latest/rllib.html.
- An RL environment. You can use custom environments, open-source environments, or commercial environments. For information, see RL Environments in Amazon SageMaker (p. 458).

The following diagram shows the RL components that are supported in Amazon SageMaker RL.

Amazon SageMaker RL

End-to-end examples for classic RL and real-worl

Robotics

Industrial Control

HVAC

Autonomous Vehicles

Operations

RL Environments to model real-world pr

AWS Simulation Environments

Amazon Sumerian

AWS RoboMaker

Open Source Environments

EnergyPlus

RoboSchool

PyBullet

Open AI Gym

RL Toolkits that provide RL agent algorithm imp

RL-Coach

DQN

PPO

HER

Rainbow

...

RL-Ray RLLib

APEX

ES

IMPALA

SageMaker Deep Learning Framewor

TensorFlow

MxNet

PyT

Training Options

Single Machine / Distributed

Local / Remote simulation

SageMaker supported

Customer BYO

Sample RL Workflow Using Amazon SageMaker RL

The following example describes the steps for developing RL models using Amazon SageMaker RL.

For complete code examples, see the sample notebooks at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement-learning.

1. **Formulate the RL problem**—First, formulate the business problem into an RL problem. For example, auto scaling enables services to dynamically increase or decrease capacity depending on conditions that you define. Currently, this requires setting up alarms, scaling policies, and thresholds, and other manual steps. To solve this with RL, we define the components of the Markov Decision Process:

   a. **Objective**—Scale instance capacity so that it matches the desired load profile.

   b. **Environment**—A custom environment that includes the load profile. It generates a simulated load with daily and weekly variations and occasional spikes. The simulated system has a delay between when new resources are requested and when they become available for serving requests.

   c. **State**—The current load, number of failed jobs, and number of active machines

   d. **Action**—Remove, add, or keep the same number of instances.

   e. **Reward**—A positive reward for successful transactions, a high penalty for failing transactions beyond a specified threshold.

2. **Define the RL environment**—The RL environment can be the real world where the RL agent interacts or a simulation of the real world. You can connect open source and custom environments developed using Gym interfaces, and commercial simulation environments such as MATLAB and Simulink.

3. **Define the presets**—The presets configure the RL training jobs and define the hyperparameters for the RL algorithms.

4. **Write the training code**—Write training code as a Python script and pass the script to an Amazon SageMaker training job. In your training code, import the environment files and the preset files, and then define the `main()` function.

5. **Train the RL Model**— Use the Amazon SageMaker `RLEstimator` in the Amazon SageMaker Python SDK to start an RL training job. If you are using local mode, the training job runs on the notebook instance. When you use Amazon SageMaker for training, you can select GPU or CPU instances. Store the output from the training job in a local directory if you train in local mode, or on Amazon S3 if you use Amazon SageMaker training.

   For information about using the Amazon SageMaker Python SDK for RL, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/rl/README.rst .

   The `RLEstimator` requires the following information as parameters.

   a. The source directory where the environment, presets, and training code are uploaded.

   b. The path to the training script.

   c. The RL toolkit and deep learning framework you want to use. This automatically resolves to the Amazon ECR path for the RL container.

   d. The training parameters, such as the instance count, job name, and S3 path for output.

   e. Metric definitions that you want to capture in your logs. These can also be visualized in CloudWatch and in Amazon SageMaker notebooks.

6. **Visualize training metrics and output**—After a training job that uses an RL model completes, you can view the metrics you defined in the training jobs in CloudWatch,. You can also plot the metrics in a notebook by using the Amazon SageMaker Python SDK analytics library. Visualizing metrics helps you understand how the performance of the model as measured by the reward improves over time.

> **Note**
> If you train in local mode, you can't visualize metrics in CloudWatch.

7. **Evaluate the model**—Checkpointed data from the previously trained models can be passed on for evaluation and inference in the checkpoint channel. In local mode, use the local directory. In Amazon SageMaker training mode, you need to upload the data to S3 first.

8. **Deploy RL models**—Finally, deploy the trained model on an endpoint hosted on Amazon SageMaker or on an Edge device by using AWS IoT Greengrass.

RL Environments in Amazon SageMaker

Amazon SageMaker RL uses environments to mimic real-world scenarios. Given the current state of the environment and an action taken by the agent or agents, the simulator processes the impact of the action, and returns the next state and a reward. Simulators are useful in cases where it is not safe to train an agent in the real world (for example, flying a drone) or if the RL algorithm takes a long time to converge (for example, when playing chess).

The following diagram shows an example of the interactions with a simulator for a car racing game.
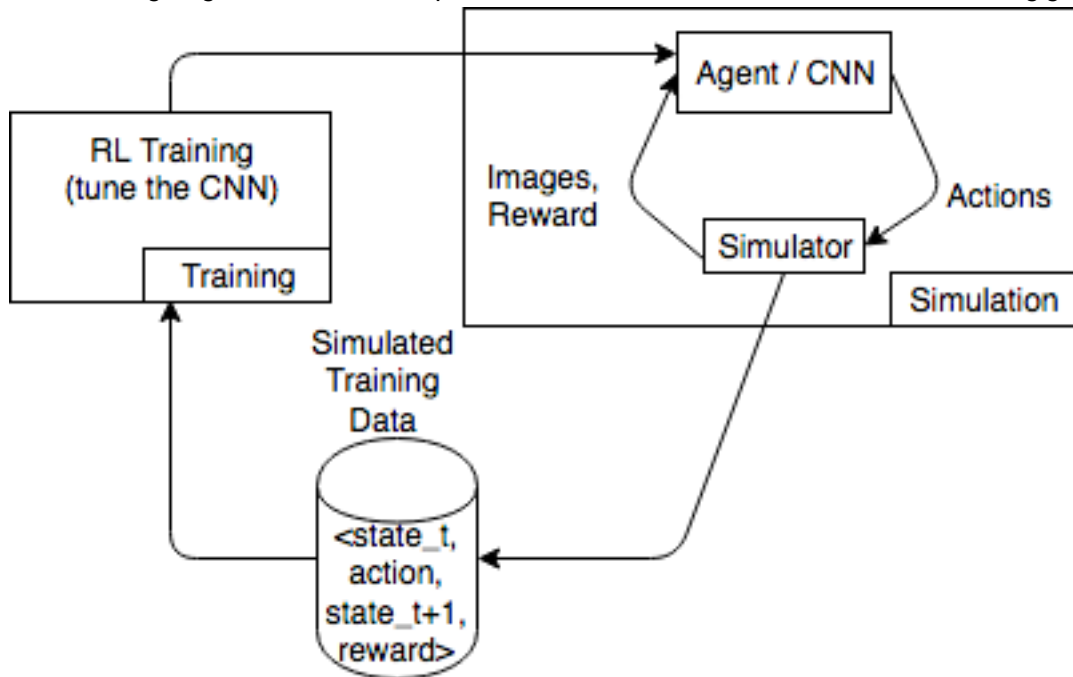
The simulation environment consists of an agent and a simulator. Here, a convolutional neural network (CNN) consumes images from the simulator and generates actions to control the game controller. With multiple simulations, this environment generates training data of the form `state_t`, `action`, `state_t+1`, and `reward_t+1`. Defining the reward is not trivial and impacts the RL model quality. We want to provide a few examples of reward functions, but would like to make it user-configurable.

Topics

- Use OpenAI Gym Interface for Environments in Amazon SageMaker RL (p. 459)
- Use Open Source Environments (p. 459)
- Use Commercial Environments (p. 459)

Use OpenAI Gym Interface for Environments in Amazon SageMaker RL

To use OpenAI Gym environments in Amazon SageMaker RL, use the following API elements. For more information about OpenAI Gym, see https://gym.openai.com/docs/.

- `env.action_space`—Defines the actions the agent can take, specifies whether each action is continuous or discrete, and specifies the minimum and maximum if the action is continuous.
- `env.observation_space`—Defines the observations the agent receives from the environment, as well as minimum and maximum for continuous observations.
- `env.reset()`—Initializes a training episode. The `reset()` function returns the initial state of the environment, and the agent uses the initial state to take its first action. The action is then sent to the `step()` repeatedly until the episode reaches a terminal state. When `step()` returns `done = True`, the episode ends. The RL toolkit re-initializes the environment by calling `reset()`.
- `step()`—Takes the agent action as input and outputs the next state of the environment, the reward, whether the episode has terminated, and an `info` dictionary to communicate debugging information. It is the responsibility of the environment to validate the inputs.
- `env.render()`Used for environments that have visualization. The RL toolkit calls this function to capture visualizations of the environment after each call to the `step()` function.

Use Open Source Environments

You can use open source environments, such as EnergyPlus and RoboSchool, in Amazon SageMaker RL by building your own container. For more information about EnergyPlus, see https://energyplus.net/. For more information about RoboSchool, see https://github.com/openai/roboschool. The HVAC and RoboSchool examples in the samples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning show how to build a custom container to use with Amazon SageMaker RL:

Use Commercial Environments

You can use commercial environments, such as MATLAB and Simulink, in Amazon SageMaker RL by building your own container. You need to manage your own licenses.

Distributed Training with Amazon SageMaker RL

Amazon SageMaker RL supports multi-core and multi-instance distributed training. Depending on your use case, training and/or environment rollout can be distributed. For example, Amazon SageMaker RL works for the following distributed scenarios:

- Single training instance and multiple rollout instances of the same instance type. For an example, see the Neural Network Compression example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance and multiple rollout instances, where different instance types for training and rollouts. For an example, see the AWS DeepRacer / AWS RoboMaker example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance that uses multiple cores for rollout. For an example, see the Roboschool example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning. This is useful if the simulation environment is light-weight and can run on a single thread.

- Multiple instances for training and rollouts. For an example, see the Roboschool example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.

Hyperparameter Tuning with Amazon SageMaker RL

You can run a hyperparameter tuning job to optimize hyperparameters for Amazon SageMaker RL. The Roboschool example in the sample notebooks at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/'reinforcement-learning shows how you can do this with RL Coach. The launcher script shows how you can abstract parameters from the Coach preset file and optimize them.

Monitor Amazon SageMaker

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon SageMaker and your other AWS solutions. AWS provides the following monitoring tools to watch Amazon SageMaker, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from EC2 instances, AWS CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.
- *CloudWatch Events* delivers a near real-time stream of system events that describe changes in AWS resources. Create CloudWatch Events rules react to a status change in a Amazon SageMaker training, hyperparameter tuning, or batch transform job

Topics

Monitor Amazon SageMaker with Amazon CloudWatch

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. However, the Amazon CloudWatch console limits the search to metrics that were updated in the last 2 weeks. This limitation ensures that the most current jobs are shown in your namespace. To graph metrics without using a search, specify its exact name in the source view. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

Amazon SageMaker model training jobs and endpoints write CloudWatch metrics and logs. The following tables list the metrics and dimensions for Amazon SageMaker.

Endpoint Invocation Metrics

The `AWS/SageMaker` namespace includes the following request metrics from calls to
InvokeEndpoint (p. 853) .

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see GetMetricStatistics in the
Amazon CloudWatch API Reference.

Metric	Description
`Invocation4XXErrors`	The number of `InvokeEndpoint` requests where the model returned a 4xx HTTP response code. For each 4xx response, 1 is sent; otherwise, 0 is sent. Units: None Valid statistics: Average, Sum
`Invocation5XXErrors`	The number of `InvokeEndpoint` requests where the model returned a 5xx HTTP response code. For each 5xx response, 1 is sent; otherwise, 0 is sent. Units: None Valid statistics: Average, Sum
`Invocations`	The `number of InvokeEndpoint` requests sent to a model endpoint. To get the total number of requests sent to a model endpoint, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
`InvocationsPerInstance`	The number of invocations sent to a model, normalized by `InstanceCount` in each ProductionVariant. 1/`numberOfInstances` is sent as the value on each request, where `numberOfInstances` is the number of active instances for the ProductionVariant behind the endpoint at the time of the request. Units: None Valid statistics: Sum
`ModelLatency`	The interval of time taken by a model to respond as viewed from Amazon SageMaker. This interval includes the local communication times taken to send the request and to fetch the response from the container of a model and the time taken to complete the inference in the container. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count
`OverheadLatency`	The interval of time added to the time taken to respond to a client request by Amazon SageMaker overheads. This interval is measured from the time Amazon SageMaker receives the request until it returns a response to the client, minus the `ModelLatency`. Overhead latency can vary depending on multiple factors, including request and response payload sizes, request frequency, and authentication/authorization of the request. Units: Microseconds

Metric	Description
	Valid statistics: Average, Sum, Min, Max, Sample Count

Dimensions for Endpoint Invocation Metrics

Dimension	Description
`EndpointName, VariantName`	Filters endpoint invocation metrics for a `ProductionVariant` of the specified endpoint and variant.

Training Job, Batch Transform Job, and Endpoint Instance Metrics

The `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs` and `/aws/sagemaker/Endpoints` namespaces include the following metrics for the training jobs and endpoint instances.

Metrics are available at a 1-minute frequency.

Metric	Description
`CPUUtilization`	The percentage of CPU units that are used by the containers on an instance. The value can range between 0 and 100, and is multiplied by the number of CPUs. For example, if there are four CPUs, `CPUUtilization` can range from 0% to 400%.
	For training jobs, the value is the CPU utilization of the algorithm container on the instance.
	For batch transform jobs, the value is the CPU utilization of the transform container on the instance.
	For endpoint variants, the value is the sum of the CPU utilization of the primary and supplementary containers on the instance.
	Note For multi-instance, each instance reports CPU utilization metrics. However, the default view in CloudWatch shows the average CPU utilization across all instances.
	Units: Percent
`MemoryUtilization`	The percentage of memory that is used by the containers on an instance. This value can range between 0% and 100%.
	For training jobs, the value is the memory utilization of the algorithm container on the instance.
	For batch transform jobs, the value is the memory utilization of the transform container on the instance.
	For endpoint variants, the value is the sum of the memory utilization of the primary and supplementary containers on the instance.
	Units: Percent

Metric	Description
	Note For multi-instance, each instance reports memory utilization metrics. However, the default view in CloudWatch shows the average memory utilization across all instances.
GPUUtilization	The percentage of GPU units that are used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUUtilization can range from 0% to 400%. For training jobs, the value is the GPU utilization of the algorithm container on the instance. For batch transform jobs, the value is the GPU utilization of the transform container on the instance. For endpoint variants, the value is the sum of the GPU utilization of the primary and supplementary containers on the instance. **Note** For multi-instance, each instance reports GPU utilization metrics. However, the default view in CloudWatch shows the average GPU utilization across all instances. Units: Percent
GPUMemoryUtilization	The percentage of GPU memory used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUMemoryUtilization can range from 0% to 400%. For training jobs, the value is the GPU memory utilization of the algorithm container on the instance. For batch transform jobs, the value is the GPU memory utilization of the transform container on the instance. For endpoint variants, the value is the sum of the GPU memory utilization of the primary and supplementary containers on the instance. **Note** For multi-instance, each instance reports GPU memory utilization metrics. However, the default view in CloudWatch shows the average GPU memory utilization across all instances. Units: Percent

Metric	Description
DiskUtilization	The percentage of disk space used by the containers on an instance uses. This value can range between 0% and 100%. This metric is not supported for batch transform jobs.

For training jobs, the value is the disk space utilization of the algorithm container on the instance.

For endpoint variants, the value is the sum of the disk space utilization of the primary and supplementary containers on the instance.

Units: Percent

> **Note**
> For multi-instance, each instance reports disk utilization metrics. However, the default view in CloudWatch shows the average disk utilization across all instances. |

Dimensions for Training Job, Batch Transform Job, and Endpoint Instance Metrics

Dimension	Description
Host	For training jobs, the value for this dimension has the format `[training-job-name]/algo-[instance-number-in-cluster]`. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the `/aws/sagemaker/TrainingJobs` namespace.

For batch transform jobs, the value for this dimension has the format `[transform-job-name]/[instance-id]`. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the `/aws/sagemaker/TransformJobs` namespace.

For endpoints, the value for this dimension has the format `[endpoint-name]/[ production-variant-name ]/[instance-id]`. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the `/aws/sagemaker/Endpoints` namespace. |

Amazon SageMaker Ground Truth Metrics

Metric	Description
DatasetObjectsAutoAnnotated	The number of dataset objects auto-annotated in a labeling job. This metric is only emitted when automated labeling is enabled. To view the labeling job progress, use the Max metric.

Units: None

Valid statistics: Max |
| DatasetObjectsHumanAnnotated | The number of dataset objects annotated by a human in a labeling job. To view the labeling job progress, use the Max metric. |

Metric	Description
	Units: None Valid statistics: Max
`DatasetObjectsLabelingFailed`	The number of dataset objects that failed labeling in a labeling job. To view the labeling job progress, use the Max metric. Units: None Valid statistics: Max
`JobsFailed`	The number of labeling jobs that failed. To get the total number of labeling jobs that failed, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
`JobsSucceeded`	The number of labeling jobs that succeeded. To get the total number of labeling jobs that succeeded, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
`JobsStopped`	The number of labeling jobs that were stopped. To get the total number of labeling jobs that were stopped, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
`TotalDatasetObjectsLabeled`	The number of dataset objects labeled successfully in a labeling job. To view the labeling job progress, use the Max metric. Units: None Valid statistics: Max

Dimensions for Dataset Object Metrics

Dimension	Description
`LabelingJobName`	Filters dataset object count metrics for a labeling job.

Log Amazon SageMaker Events with Amazon CloudWatch

To help you debug your training jobs, endpoints, transform jobs, notebook instances, and notebook instance lifecycle configurations, anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` is also sent to Amazon CloudWatch Logs. In addition to debugging, you can use these for progress analysis.

Logs

The following table lists all of the logs provided by Amazon SageMaker.

Logs

Log Group Name	Log Stream Name
`/aws/sagemaker/` `TrainingJobs`	`[training-job-name]/algo-[instance-number-in-cluster]-` `[epoch_timestamp]`

`/aws/sagemaker/`
`Endpoints/`
`[EndpointName]`
	`[production-variant-name]/[instance-id]`

	`[production-variant-name]/[instance-id]/[container-name` `provided in SageMaker model] (For Inference Pipelines)`
`/aws/sagemaker/` `NotebookInstances`	`[notebook-instance-name]/[LifecycleConfigHook]`
	`[notebook-instance-name]/jupyter.log`
`/aws/sagemaker/` `TransformJobs`	`[transform-job-name]/[instance-id]-[epoch_timestamp]`
	`[transform-job-name]/[instance-id]-[epoch_timestamp]/data-` `log`

	`[transform-job-name]/[instance-id]-[epoch_timestamp]/`
	`[container-name provided in SageMaker model] (For`
	`Inference Pipelines)`

> **Note**
> 1. The `/aws/sagemaker/NotebookInstances/[LifecycleConfigHook]` log stream
> is created when you create a notebook instance with a lifecycle configuration. For more
> information, see Customize a Notebook Instance (p. 40).
> 2. For Inference Pipelines, if you don't provide container names, the platform uses **container-1,
> container-2**, and so on, corresponding to the order provided in the Amazon SageMaker model.

For more information about logging events with CloudWatch logging, see What is Amazon CloudWatch
Logs? in the *Amazon CloudWatch User Guide.*

Log Amazon SageMaker API Calls with AWS CloudTrail

Amazon SageMaker is integrated with AWS CloudTrail, a service that provides a record of actions taken
by a user, role, or an AWS service in Amazon SageMaker. CloudTrail captures all API calls for Amazon
SageMaker, with the exception of InvokeEndpoint (p. 853), as events. The calls captured include
calls from the Amazon SageMaker console and code calls to the Amazon SageMaker API operations.
If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket,
including events for Amazon SageMaker. If you don't configure a trail, you can still view the most recent
events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can
determine the request that was made to Amazon SageMaker, the IP address from which the request was
made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

By default, log data is stored in CloudWatch Logs indefinitely. Howerver, you can configure how long to
store log data in a log group. For information, see Change Log Data Retention in CloudWatch Logs in the
Amazon CloudWatch Logs User Guide.

Amazon SageMaker Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon SageMaker, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon SageMaker, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon SageMaker actions, with the exception of InvokeEndpoint (p. 853), are logged by CloudTrail and are documented in the Actions (p. 616). For example, calls to the `CreateTrainingJob`, `CreateEndpoint` and `CreateNotebookInstance` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Operations Performed by Automatic Model Tuning

Amazon SageMaker supports logging non-API service events to your CloudTrail log files,for automatic model tuning jobs. These events are related to your tuning jobs but, are not the direct result of a customer request to the public AWS API. For example, when you create a hyperparameter tuning job by calling CreateHyperParameterTuningJob (p. 638), Amazon SageMaker creates training jobs to evaluate various combinations of hyperparameters to find the best result. Similarly, when you call StopHyperParameterTuningJob (p. 828) to stop a hyperparameter tuning job, Amazon SageMaker might stop any of the associated running training jobs. Non-API events for your tuning jobs are logged to CloudTrail to help you improve governance, compliance, and operational and risk auditing of your AWS account.

Log entries that result from non-API service events have an `eventType` of `AwsServiceEvent` instead of `AwsApiCall`.

Understanding Amazon SageMaker Log File Entries

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any

source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following examples a log entry for the `CreateEndpoint` action, which creates an endpoint to deploy a trained model.

```
{
    "eventVersion":"1.05",
    "userIdentity": {
        "type":"IAMUser",
        "principalId":"AIXDAYQEXAMPLEUMLYNGL",
        "arn":"arn:aws:iam::123456789012:user/intern",
        "accountId":"123456789012",
        "accessKeyId":"ASXIAGXEXAMPLEQULKNXV",
        "userName":"intern"
    },
    "eventTime":"2018-01-02T13:39:06Z",
    "eventSource":"sagemaker.amazonaws.com",
    "eventName":"CreateEndpoint",
    "awsRegion":"us-west-2",
    "sourceIPAddress":"127.0.0.1",
    "userAgent":"USER_AGENT",
    "requestParameters": {
        "endpointName":"ExampleEndpoint",
        "endpointConfigName":"ExampleEndpointConfig"
    },
    "responseElements": {
        "endpointArn":"arn:aws:sagemaker:us-west-2:123456789012:endpoint/exampleendpoint"
    },
    "requestID":"6b1b42b9-EXAMPLE",
    "eventID":"a6f85b21-EXAMPLE",
    "eventType":"AwsApiCall",
    "recipientAccountId":"444455556666"
}
```

The following example is a log entry for the `CreateModel` action, which creates one or more containers to host a previously trained model.

```
{
    "eventVersion":"1.05",
    "userIdentity": {
        "type":"IAMUser",
        "principalId":"AIXDAYQEXAMPLEUMLYNGL",
        "arn":"arn:aws:iam::123456789012:user/intern",
        "accountId":"123456789012",
        "accessKeyId":"ASXIAGXEXAMPLEQULKNXV",
        "userName":"intern"
    },
    "eventTime":"2018-01-02T15:23:46Z",
    "eventSource":"sagemaker.amazonaws.com",
    "eventName":"CreateModel",
    "awsRegion":"us-west-2",
    "sourceIPAddress":"127.0.0.1",
    "userAgent":"USER_AGENT",
    "requestParameters": {
        "modelName":"ExampleModel",
        "primaryContainer": {
            "image":"174872318107.dkr.ecr.us-west-2.amazonaws.com/kmeans:latest"
        },
        "executionRoleArn":"arn:aws:iam::123456789012:role/EXAMPLEARN"
    },
    "responseElements": {
```

```
        "modelArn":"arn:aws:sagemaker:us-west-2:123456789012:model/
barkinghappy2018-01-02t15-23-32-275z-ivrdog"
    },
    "requestID":"417b8dab-EXAMPLE",
    "eventID":"0f2b3e81-EXAMPLE",
    "eventType":"AwsApiCall",
    "recipientAccountId":"444455556666"
}
```

React to Amazon SageMaker Job Status Changes with CloudWatch Events

To react to a status change in a Amazon SageMaker training, hyperparameter tuning, or batch transform job, create a rule in CloudWatch Events that use the **SageMaker Training Job State Change**, **SageMaker Hyperparameter Tuning Job State Change**, or **SageMaker Transform Job State Change** event type as the event source for the rule.

Every time the status of a Amazon SageMaker job changes, it triggers an event that CloudWatch Events monitors, and you can create a rule that calls a AWS Lambda function when the status changes. For information about the status values and meanings for Amazon SageMaker jobs, see the following:

- TrainingJobStatus
- HyperParameterTuningJobStatus
- TransformJobStatus

For information about creating CloudWatch Events rules, see Creating a CloudWatch Events Rule That Triggers on an Event in the *CloudWatch Events User Guide*. For detailed information about the format of the Amazon SageMaker events that CloudWatch Events monitors, see Amazon SageMaker Events.

Security in Amazon SageMaker

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon SageMaker, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon SageMaker. The following topics show you how to configure Amazon SageMaker to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon SageMaker resources.

Topics

Data Protection in Amazon SageMaker

Amazon SageMaker conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.

- Set up API and user activity logging with AWS CloudTrail.

- Use AWS encryption solutions, along with all default security controls within AWS services.

- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon SageMaker or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon SageMaker or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

Topics

Protecting Data at Rest Using Encryption

You can use encrypted Amazon Simple Storage Service buckets for model artifacts and data, as well as pass a AWS Key Management Service key to Amazon SageMaker notebooks, training jobs, hyperparameter tuning jobs, batch transform jobs, and endpoints, to encrypt the attached machine learning (ML) storage volume. If you do not specify a AWS KMS key, Amazon SageMaker encrypts storage volumes with a transient key. A transient key is discarded immediately after it is used to encrypt the storage volume.

All instance OS volumes are encrypted with an AWS-managed AWS KMS key.

All ML data volumes for all Amazon SageMaker instances may be encrypted with customer specified AWS KMS keys. ML data volumes are mounted as follows:

- Notebooks - `/home/ec2-user/SageMaker`

- Training - `/opt/ml/` and `/tmp/`

- Batch - `/opt/ml/` and `/tmp/`

- Endpoints - `/opt/ml/` and `/tmp/`

Batch and training job containers and their storage are ephemeral in nature. When the job completes, output is uploaded to Amazon S3 (with optional AWS KMS encryption) and the instance is torn down.

Data of a sensitive nature that needs to be encrypted with a customer owned AWS KMS key for compliance reasons should be stored in the ML Amazon EBS volume or Amazon S3, both of which can be KMS encrypted with customer managed keys. Notebook instances mount all default folders used by Jupyter or the algorithm containers onto the ML volume.

The Amazon SageMaker folder in the ML Amazon EBS volume is the default storage location when you open a notebook instance. Amazon SageMaker saves any files within the SageMaker folder. The / sample-notebooks subfolder is located on the OS volume but that location is read only. When you stop a Notebook instance any customizations to the OS (like custom libraries installed or OS level settings)

are lost. Consider utilizing lifecycle options to automate any customizations to the default image. If a Notebook instance is stopped, a snapshot of the ML volume is retained by Amazon in the service platform to support resumption. This snapshot is deleted on termination as well as the ML volume, so any data to be persisted beyond the notebook lifecycle should be transferred to customer Amazon S3 buckets.

Protecting Data in Transit with Encryption

All inter-network data in transit supports TLS 1.2 encryption.

Amazon SageMaker ensures that machine learning (ML) model artifacts and other system artifacts are encrypted in transit and at rest. Requests to the Amazon SageMaker API and console are made over a secure (SSL) connection. You pass AWS Identity and Access Management roles to Amazon SageMaker to provide permissions to access resources on your behalf for training and deployment. You can use encrypted Amazon S3 buckets for model artifacts and data, as well as pass a AWS KMS key to Amazon SageMaker instances to encrypt the attached ML storage volumes.

Some intra-network data in-transit (inside the service platform) is unencrypted. This includes:

- Command and control communications between the service control plane and training job instances (not customer data).
- Communications between nodes in distributed training jobs (intra-network).

There are no inter-node communications for batch processing.

You can choose to encrypt internode training communications. Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. For affected algorithms, adding this additional level of security also increases cost. The training time for most Amazon SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

FIPS validated endpoints are available for the Amazon SageMaker API and request router for hosted models (runtime). For information about FIPS compliant endpoints, see Federal Information Processing Standard (FIPS) 140-2.

Protect Communications Between ML Compute Instances in a Distributed Training Job

By default, Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud (Amazon VPC) to help keep your data secure. You can add another level of security to protect your training containers and data by configuring a *private* VPC. Distributed ML frameworks and algorithms usually transmit information that is directly related to the model such as weights, not the training dataset. When performing distributed training, you can further protect data that is transmitted between instances. This can help you to comply with regulatory requirements. To do this, use inter-container traffic encryption.

Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. Enabling inter-container traffic encryption doesn't affect training jobs with a single compute instance. However, for training jobs with several compute instances, the effect on training time depends on the amount of communication between compute instances. For affected algorithms, adding this additional level of security also increases cost. The training time for most Amazon SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

You can enable inter-container traffic encryption for training jobs or hyperparameter tuning jobs. You can use Amazon SageMaker APIs or console to enable inter-container traffic encryption.

For information about running training jobs in a private VPC, see Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC (p. 522).

Enable Inter-Container Traffic Encryption (API)

Before enabling inter-container traffic encryption on training or hyperparameter tuning jobs with APIs, you need to add inbound and outbound rules to your private VPC's security group.

To enable inter-container traffic encryption (API)

1. Add the following inbound and outbound rules in the security group for your private VPC:

Protocol	Port Range	Source
UDP	500	*Self Security Group ID*
50	N/A	*Self Security Group ID*

2. When you send a request to the CreateTrainingJob (p. 667) or CreateHyperParameterTuningJob (p. 638) API, specify `True` for the `EnableInterContainerTrafficEncryption` parameter.

   Note
   The AWS Security Group Console might show display ports range as "All", however EC2 ignores the specified port range because it is not applicable for the ESP 50 IP protocol.

Enable Inter-Container Traffic Encryption (Console)

Enable Inter-container Traffic Encryption in a Training Job

To enable inter-container traffic encryption in a training job

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Under **Network**, choose a **VPC**. You can use the default VPC or one that you have created.
5. Choose **Enable inter-container traffic encryption**.

After you enable inter-container traffic encryption, finish creating the training job. For more information, see Step 5: Train a Model (p. 21).

Enable Inter-container Traffic Encryption in a Hyperparameter Tuning Job

To enable inter-container traffic encryption in a hyperparameter tuning job

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker.
2. In the navigation pane, choose **Training**, then choose **Hyperparameter tuning jobs**.
3. Choose **Create hyperparameter tuning job**.
4. Under **Network**, choose a **VPC**. You can use the default VPC or one that you created.
5. Choose **Enable inter-container traffic encryption**.

After enabling inter-container traffic encryption, finish creating the hyperparameter tuning job. For more information, see Configure and Launch a Hyperparameter Tuning Job (p. 296).

Key Management

Customers can specify AWS KMS keys, including bring your own keys (BYOK), to use for envelope encryption with Amazon S3 input/output buckets and machine learning (ML) Amazon EBS volumes. ML volumes for notebook instances and training and hosted model Docker containers can be optionally encrypted by using AWS KMS customer-owned keys. All instance OS volumes are encrypted with an AWS-managed AWS KMS key.

For information about AWS KMS keys see What is AWS Key Management Service? in the *AWS Key Management Service Developer Guide*.

Internetwork Traffic Privacy

This topic describes how Amazon SageMaker secures connections from the service to other locations.

Internetwork communications support TLS 1.2 encryption between all components and clients.

Instances can be connected to Customer VPC, providing access to S3 VPC endpoints or customer repositories. Internet egress can be managed through this interface by the customer if service platform internet egress is disabled for notebooks. For training and hosting, egress through the service platform is not available when connected to the customer's VPC.

By default, API calls made to published endpoints traverse the public network to the request router. Amazon SageMaker supports Amazon Virtual Private Cloud interface endpoints powered by AWS PrivateLink for private connectivity between the customer's VPC and the request router to access hosted model endpoints. For information about Amazon VPC, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 517)

Identity and Access Management for Amazon SageMaker

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon SageMaker resources. IAM is an AWS service that you can use with no additional charge.

Topics
- Audience (p. 475)
- Authenticating With Identities (p. 476)
- Managing Access Using Policies (p. 478)
- How Amazon SageMaker Works with IAM (p. 479)
- Amazon SageMaker Identity-Based Policy Examples (p. 481)
- Amazon SageMaker Roles (p. 496)
- AWS Managed (Predefined) Policies for Amazon SageMaker (p. 507)
- Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 508)
- Troubleshooting Amazon SageMaker Identity and Access (p. 512)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon SageMaker.

Service user – If you use the Amazon SageMaker service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon SageMaker features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon SageMaker, see Troubleshooting Amazon SageMaker Identity and Access (p. 512).

Service administrator – If you're in charge of Amazon SageMaker resources at your company, you probably have full access to Amazon SageMaker. It's your job to determine which Amazon SageMaker features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon SageMaker, see How Amazon SageMaker Works with IAM (p. 479).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon SageMaker. To view example Amazon SageMaker identity-based policies that you can use in IAM, see Amazon SageMaker Identity-Based Policy Examples (p. 481).

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key

pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access Control Lists (ACLs)

Access control policies (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) Overview in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify

the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the *IAM User Guide*.

How Amazon SageMaker Works with IAM

Before you use IAM to manage access to Amazon SageMaker, you should understand what IAM features are available to use with Amazon SageMaker. To get a high-level view of how Amazon SageMaker and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

- Amazon SageMaker Identity-Based Policies (p. 479)

Amazon SageMaker Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon SageMaker supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon SageMaker use the following prefix before the action: `sagemaker:`. For example, to grant someone permission to run an Amazon SageMaker training job with the Amazon SageMaker `CreateTrainingJob` API operation, you include the `sagemaker:CreateTrainingJob` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon SageMaker defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "sagemaker:action1",
```

```
        "sagemaker:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "sagemaker:Describe*"
```

To see a list of Amazon SageMaker actions, see Actions Defined by Amazon SageMaker in the *IAM User Guide*.

Resources

Amazon SageMaker does not support specifying resource ARNs in a policy.

Condition Keys

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the *IAM User Guide*.

Amazon SageMaker defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Amazon SageMaker supports a number of service-specific condition keys that you can use for fine-grained access control for the following operations:

- the section called "CreateTrainingJob" (p. 667)
- the section called "CreateModel" (p. 648)
- the section called "CreateEndpointConfig" (p. 635)
- the section called "CreateTransformJob" (p. 673)
- the section called "CreateHyperParameterTuningJob" (p. 638)
- the section called "CreateNotebookInstance" (p. 656)
- the section called "UpdateNotebookInstance" (p. 844)

To see a list of Amazon SageMaker condition keys, see Condition Keys for Amazon SageMaker in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon SageMaker.

For examples of using Amazon SageMaker condition keys, see the following:

Examples

To view examples of Amazon SageMaker identity-based policies, see Amazon SageMaker Identity-Based Policy Examples (p. 481).

Amazon SageMaker Resource-Based Policies

Amazon SageMaker does not support resource-based policies.

Authorization Based on Amazon SageMaker Tags

You can attach tags to Amazon SageMaker resources or pass tags in a request to Amazon SageMaker. To control access based on tags, you provide tag information in the condition element of a policy using the `sagemaker:ResourceTag/`*key-name*, `aws:RequestTag/`*key-name*, or `aws:TagKeys` condition keys. For more information about tagging Amazon SageMaker resources, see Control Access to Amazon SageMaker Resources by Using Tags (p. 493).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Control Access to Amazon SageMaker Resources by Using Tags (p. 493).

Amazon SageMaker IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon SageMaker

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon SageMaker supports using temporary credentials.

Service-Linked Roles

Amazon SageMaker doesn't support service-linked roles.

Service Roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon SageMaker supports service roles.

Choosing an IAM Role in Amazon SageMaker

When you create a notebook instance, training job, hosted endpoint, or batch transform job resource in Amazon SageMaker, you must choose a role to allow Amazon SageMaker to access Amazon SageMaker on your behalf. If you have previously created a service role or service-linked role, then Amazon SageMaker provides you with a list of roles to choose from. It's important to choose a role that allows access to the AWS operations and resources you need. For more information, see Amazon SageMaker Roles (p. 496).

Amazon SageMaker Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon SageMaker resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon SageMaker resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon SageMaker quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Using the Amazon SageMaker Console

To access the Amazon SageMaker console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon SageMaker resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon SageMaker console, also attach the following AWS managed policy to the entities. For more information, see Adding Permissions to a User in the *IAM User Guide*:

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Topics

Permissions Required to Use the Amazon SageMaker Console

The permissions reference table lists the Amazon SageMaker API operations and shows the required permissions for each operation. For more information about Amazon SageMaker API operations, see Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 508).

To use the Amazon SageMaker console, you need to grant permissions for additional actions. Specifically, the console needs permissions that allow the `ec2` actions to display subnets, VPCs, and security groups. Optionally, the console needs permission to create *execution roles* for tasks such as `CreateNotebook`, `CreateTrainingJob`, and `CreateModel`. Grant these permissions with the following permissions policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SageMakerApis",
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VpcConfigurationForCreateForms",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Sid":"KmsKeysForCreateForms",
            "Effect":"Allow",
            "Action":[
                "kms:DescribeKey",
                "kms:ListAliases"
            ],
            "Resource":"*"
        },
        {
            "Sid": "AccessAwsMarketplaceSubscritions",
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:ViewSubscriptions"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:BatchGetRepositories",
                "codecommit:CreateRepository",
                "codecommit:GetRepository",
                "codecommit:ListRepositories",
                "codecommit:ListBranches",
                "secretsmanager:CreateSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        },
```

```
      {
        "Sid":"ListAndCreateExecutionRoles",
        "Effect":"Allow",
        "Action":[
          "iam:ListRoles",
          "iam:CreateRole",
          "iam:CreateRole",
          "iam:CreatePolicy",
          "iam:AttachRolePolicy"
        ],
        "Resource":"*"
      },
      {
        "Sid": "DescribeECRMetaData",
        "Effect": "Allow",
        "Action": [
            "ecr:Describe*"
        ],
        "Resource": "*"
      },
      {
        "Sid": "PassRoleForExecutionRoles",
        "Effect": "Allow",
        "Action": [
          "iam:PassRole"
        ],
        "Resource": "*",
        "Condition": {
          "StringEquals": {
              "iam:PassedToService": "sagemaker.amazonaws.com"
          }
        }
      }
    ]
}
```

Permissions Required to Use the Amazon SageMaker Ground Truth Console

To use the Amazon SageMaker Ground Truth console, you need to grant permissions for additional resources. Specifically, the console needs permissions for the AWS Marketplace to view subscriptions, Amazon Cognito operations to manage your private workforce, Amazon S3 actions for access to your input and output files, and AWS Lambda actions to list and invoke functions. Grant these permissions with the following permissions policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GroundTruthConsole",
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:DescribeListings",
                "aws-marketplace:ViewSubscriptions",

                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:AdminDeleteUser",
                "cognito-idp:AdminDisableUser",
                "cognito-idp:AdminEnableUser",
                "cognito-idp:AdminRemoveUserFromGroup",
                "cognito-idp:CreateGroup",
                "cognito-idp:CreateUserPool",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:CreateUserPoolDomain",
```

```
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp:ListGroups",
                "cognito-idp:ListIdentityProviders",
                "cognito-idp:ListUsers",
                "cognito-idp:ListUsersInGroup",
                "cognito-idp:ListUserPoolClients",
                "cognito-idp:ListUserPools",
                "cognito-idp:UpdateUserPool",
                "cognito-idp:UpdateUserPoolClient",

                "groundtruthlabeling:DescribeConsoleJob",
                "groundtruthlabeling:ListDatasetObjects",
                "groundtruthlabeling:RunFilterOrSampleManifestJob",
                "groundtruthlabeling:RunGenerateManifestByCrawlingJob",

                "lambda:InvokeFunction",
                "lambda:ListFunctions",

                "s3:GetObject",
                "s3:PutObject",
                "s3:SelectObjectContent"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "ViewOwnUserInfo",
                "Effect": "Allow",
                "Action": [
                    "iam:GetUserPolicy",
                    "iam:ListGroupsForUser",
                    "iam:ListAttachedUserPolicies",
                    "iam:ListUserPolicies",
                    "iam:GetUser"
                ],
                "Resource": [
                    "arn:aws:iam::*:user/${aws:username}"
                ]
            },
            {
                "Sid": "NavigateInConsole",
                "Effect": "Allow",
                "Action": [
                    "iam:GetGroupPolicy",
                    "iam:GetPolicyVersion",
                    "iam:GetPolicy",
                    "iam:ListAttachedGroupPolicies",
                    "iam:ListGroupPolicies",
                    "iam:ListPolicyVersions",
                    "iam:ListPolicies",
                    "iam:ListUsers"
                ],
```

```
            "Resource": "*"
        }
    ]
}
```

Control Creation of Amazon SageMaker Resources with Condition Keys

Control fine-grained access to creation of Amazon SageMaker resources by using Amazon SageMaker-specific condition keys. For information about using condition keys in IAM policies, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

The following examples show how to use Amazon SageMaker condition keys to control access.

Topics

Control Access to Amazon SageMaker Resources by Using File System Condition Keys

Amazon SageMaker training provides a secure infrastructure for the training algorithm to run in, but for some cases you may want increased defense in depth. For example, you minimize the risk of running untrusted code in your algorithm, or you have specific security mandates in your organization. For these scenarios, you can use the following service-specific condition keys in the Condition element of an IAM policy to scope down the user to specific file systems, directories, access modes (read-write, read-only) and security groups:

Topics

Restrict an IAM User to Specific Directories and Access Modes

The policy below restricts an IAM user to the `/sagemaker/xgboost-dm/train` and `/sagemaker/xgboost-dm/validation` directories of an EFS file system to `ro` (read-only) AccessMode:

> **Note**
> When a directory is allowed, all of its subdirectories are also accessible by the training algorithm. POSIX permissions are ignored.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessToElasticFileSystem",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
```

```
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/sagemaker/xgboost-dm/train"
                }
            }
        },
        {
            "Sid": "AccessToElasticFileSystemValidation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/sagemaker/xgboost-dm/validation"
                }
            }
        }
    ]
}
```

Restrict an IAM User to Specific File System

To prevent a malicious algorithm using a user space client from accessing any file system directly in your account, you can restrict networking traffic by allowing ingress from a specific security group. In the following example, the IAM user can only use the specified security group to access the file system:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessToLustreFileSystem",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "FSxLustre",
                    "sagemaker:FileSystemDirectoryPath": "/fsx/sagemaker/xgboost/train"
                },
                "ForAllValues:StringEquals": {
                    "sagemaker:VpcSecurityGroupIds": [
                        "sg-12345678"
                    ]
                }
            }
        }
    ]
```

```
}
```

Although the above example can restrict an algorithm to a specific file system, it does not prevent an algorithm from accessing any directory within that file system using the user space client. To mitigate this, you can:

- Ensure that the file system only contains data that you trust your IAM users to access
- Create an IAM role that restricts your IAM users to launching training jobs with algorithms from approved ECR repositories

For more information on how to use roles with Amazon SageMaker, see Amazon SageMaker Roles.

Restrict Training to a Specific VPC

Restrict an AWS user to creating training jobs from within a Amazon VPC. When a training job is created within a VPC, you can use VPC flow logs to monitor all traffic to and from the training cluster. For information about using VPC flow logs, see VPC Flow Logs in the *Amazon Virtual Private Cloud User Guide*.

The following policy enforces that a training job is created by an IAM user calling the section called "CreateTrainingJob" (p. 667) from within a VPC:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowFromVpc",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:VpcSubnets": ["10.24.34.0/23"],
                    "sagemaker:VpcSecurityGroupIds": ["vpc-12345678"]
                }
            }
        }

    ]
}
```

Enforcing Encryption of Input Data

The following policy restricts an IAM user to specify a AWS KMS key to encrypt input data when creating training and hyperparameter tuning jobs by using the `sagemaker:VolumeKmsKey` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceEncryption",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
```

```
        "Condition": {
            "Null": {
                "sagemaker:VolumeKmsKey": "false"
            }
        }
    }

    ]
}
```

Enforcing Encryption of Notebook Instance Storage Volume

The following policy restricts an IAM user to specify a AWS KMS key to encrypt the attached storage volume when creating or updating a notebook instance by using the `sagemaker:VolumeKmsKey` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceEncryption",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateNotebookInstance",
                "sagemaker:UpdateNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "sagemaker:VolumeKmsKey": "false"
                }
            }
        }

    ]
}
```

Enforcing Network Isolation for Training Jobs

The following policy restricts an IAM user to enable network isolation when creating training jobs by using the `sagemaker:NetworkIsolation` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceIsolation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:NetworkIsolation": "True"
                }
            }
        }

    ]
}
```

Enforcing a Specific Instance Type for Training Jobs

The following policy restricts an IAM user to use a specific instance type when creating training jobs by using the `sagemaker:InstanceTypes` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceInstanceType",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "sagemaker:InstanceTypes": ["ml.c5dn.*"]
                }
            }
        }

    ]
}
```

Enforce Disabling Internet Access and Root Access for Creating Notebook Instances

You can disable both internet access and root access to notebook instances to help make them more secure. For information about controling root access to a notebook instance, see Control Root Access to a Notebook Instance (p. 40). for information about disabling internet access for a notebook instance, see Connect a Notebook Instance to Resources in a VPC (p. 516).

The following policy requires an IAM user to disable network access and root access when creating or updating a notebook instance:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LockdownNotebook",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateNotebookInstance",
                "sagemaker:UpdateNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:RootAccess": "False",
                    "sagemaker:DirectInternetAccess": "False"
                },
                "Null": {
                  "sagemaker:VpcSubnets": "false",
                  "sagemaker:VpcSecurityGroupIds": "false"
                }
            }
        }

    ]
}
```

Control Access to the Amazon SageMaker API by Using Identity-based Policies

To control access to Amazon SageMaker API calls and calls to Amazon SageMaker hosted endpoints, use identity-based IAM policies.

Topics

- Restrict Access to Amazon SageMaker API and Runtime to Calls from Within Your VPC (p. 491)

Restrict Access to Amazon SageMaker API and Runtime to Calls from Within Your VPC

If you set up an interface endpoint in your VPC, individuals outside the VPC can still connect to the Amazon SageMaker API and runtime over the internet unless you attach an IAM policy that restricts access to calls coming from within the VPC to all users and groups that have access to your Amazon SageMaker resources. For information about creating a VPC interface endpoint for the Amazon SageMaker API and runtime, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 517).

> **Important**
> If you apply an IAM policy similar to one of the following, users can't access the specified Amazon SageMaker APIs through the console.

To restrict access to only connections made from within your VPC, create an AWS Identity and Access Management policy that restricts access to only calls that come from within your VPC. Then add that policy to every AWS Identity and Access Management user, group, or role used to access the Amazon SageMaker API or runtime.

> **Note**
> This policy allows connections only to callers within a subnet where you created an interface endpoint.

```
{
    "Id": "api-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable API Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpc": "vpc-111bbaaa"
                }
            }
        }
    ]
}
```

If you want to restrict access to the API to only calls made using the interface endpoint, use the `aws:SourceVpce` condition key instead of `aws:SourceVpc`:

```
{
    "Id": "api-example-1",
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "Enable API Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreatePresignedNotebookInstanceUrl"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:sourceVpce": [
                        "vpce-111bbccc",
                        "vpce-111bbddd"
                    ]
                }
            }
        }
    ]
}
```

Limit Access to Amazon SageMaker API and Runtime Calls by IP Address

To allow access to Amazon SageMaker API calls and runtime invocations only from IP addresses in a list that you specify, attach an IAM policy that denies access to the API unless the call comes from an IP address in the list to every AWS Identity and Access Management user, group, or role used to access the API or runtime. For information about creating IAM policies, see Creating IAM Policies in the *AWS Identity and Access Management User Guide*. To specify the list of IP addresses that you want to have access to the API call, use the `IpAddress` condition operator and the `aws:SourceIP` condition context key. For information about IAM condition operators, see IAM JSON Policy Elements: Condition Operators in the *AWS Identity and Access Management User Guide*. For information about IAM condition context keys, see AWS Global Condition Context Keys.

For example, the following policy allows access to the CreateTrainingJob (p. 667) only from IP addresses in the ranges `192.0.2.0-192.0.2.255` and `203.0.113.0-203.0.113.255`:

```
{
    "Version": "2012-10-17",
    "Statement": [

        {
            "Effect": "Allow",
            "Action": "sagemaker:CreateTrainingJob",
            "Resource": "*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            }
        }
    ]
}
```

Limit Access to a Notebook Instance by IP Address

To allow access to a notebook instance only from IP addresses in a list that you specify, attach an IAM policy that denies access to CreatePresignedNotebookInstanceUrl (p. 665) unless the call comes from an IP address in the list to every AWS Identity and Access Management user, group, or role used to access

the notebook instance. For information about creating IAM policies, see Creating IAM Policies in the *AWS Identity and Access Management User Guide*. To specify the list of IP addresses that you want to have access to the notebook instance, use the `IpAddress` condition operator and the `aws:SourceIP` condition context key. For information about IAM condition operators, see IAM JSON Policy Elements: Condition Operators in the *AWS Identity and Access Management User Guide*. For information about IAM condition context keys, see AWS Global Condition Context Keys.

For example, the following policy allows access to a notebook instance only from IP addresses in the ranges `192.0.2.0-192.0.2.255` and `203.0.113.0-203.0.113.255`:

```
{
    "Version": "2012-10-17",
    "Statement": [

        {
            "Effect": "Allow",
            "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
            "Resource": "*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            }
        }
    ]
}
```

The policy restricts access to both the call to `CreatePresignedNotebookInstanceUrl` and to the URL that the call returns. The policy also restricts access to opening a notebook instance in the console and is enforced for every HTTP request and WebSocket frame that attempts to connect to the notebook instance.

> **Note**
> Using this method to filter by IP address is incompatible when connecting to Amazon SageMaker through a VPC interface endpoint.. For information about restricting access to a notebook instance when connecting through a VPC interface endpoint, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 519).

Control Access to Amazon SageMaker Resources by Using Tags

Control access to groups of Amazon SageMaker resources by attaching tags to the resources and specifying `ResourceTag` conditions in IAM policies.

> **Note**
> Tag-based policies do not work to restrict the following API calls:

- ListAlgorithms
- ListCodeRepositories
- ListCompilationJobs
- ListEndpointConfigs
- ListEndpoints
- ListHyperparameterTuningJobs
- ListLabelingJobs
- ListLabelingJobsForWorkteam
- ListModelPackages
- ListModels

- ListNotebookInstanceLifecycleConfigs
- ListNotebookInstances
- ListSubscribedWorkteams
- ListTags
- ListTrainingJobs
- ListTrainingJobsForHyperParameterTuningJob
- ListTransformJobs
- ListWorkteams
- Search

For example, suppose you've defined two different IAM groups, named `DevTeam1` and `DevTeam2`, in your AWS account. Suppose also that you've created 10 notebook instances, 5 of which are used for one project, and 5 of which are used for a second project. You want to allow members of `DevTeam1` to make API calls on notebook instances used for the first project, and members of `DevTeam2` to make API calls on notebook instances used for the second project.

You can control access to API calls by completing the following steps:

1. Add a tag with the key `Project` and value `A` to the notebook instances used for the first project. For information about adding tags to Amazon SageMaker resources, see AddTags (p. 620).
2. Add a tag with the key `Project` and value `B` to the notebook instances used for the second project.
3. Create an IAM policy with a `ResourceTag` condition that denies access to the notebook instances used for the second project, and attach that policy to `DevTeam1`. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of `Project` and a value of `B`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker:*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "sagemaker:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:ResourceTag/Project": "B"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateTags",
        "sagemaker:DeleteTags"
      ],
      "Resource": "*"
    }
  ]
}
```

For information about creating IAM policies and attaching them to identities, see Controlling Access Using Policies in the *AWS Identity and Access Management User Guide*.

4. Create an IAM policy with a `ResourceTag` condition that denies access to the notebook instances used for the first project, and attach that policy to `DevTeam2`. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of `Project` and a value of `A`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "sagemaker:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:ResourceTag/Project": "A"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateTags",
        "sagemaker:DeleteTags"
      ],
      "Resource": "*"
    }
  ]
}
```

Require the Presence or Absence of Tags for API Calls

Require the presence or absence of specific tags or specific tag values by using `RequestTag` condition keys in an IAM policy. For example, if you want to require that every endpoint created by any member of an IAM group to be created with a tag with the key `environment` and value `dev`, create a policy as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "sagemaker:CreateEndpoint",
      "Resource": [
        "arn:aws:sagemaker:*:*:endpoint/*"
      ]
    {
      "Effect": "Allow",
      "Action": "sagemaker:CreateEndpoint",
      "Resource": [
        "arn:aws:sagemaker:*:*:endpoint/*"
      ],
      "Condition": {
```

```
        "StringEquals": {
          "aws:RequestTag/environment": "dev"
        }
      }
    }
  ]
}
```

Use Tags with Hyperparameter Tuning Jobs

You can add tags to a hyperparameter tuning job when you create the tuning job by specifying the tags as the `Tags` parameter when you call CreateHyperParameterTuningJob (p. 638). If you do this, the tags you specify for the hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches.

If you add tags to a hyperparameter tuning job by calling AddTags (p. 620), the tags you add are also added to any training jobs that the hyperparameter tuning job launches after you call `AddTags`, but are not added to training jobs the hyperparameter tuning jobs launched before you called `AddTags`. Similarly, when you remove tags from a hyperparameter tuning job by calling DeleteTags (p. 693), those tags are not removed from training jobs that the hyperparameter tuning job launched previously. Because of this, the tags associated with training jobs can be out of sync with the tags associated with the hyperparameter tuning job that launched them. If you use tags to control access to a hyperparameter tuning job and the training jobs it launches, you might want to keep the tags in sync. To make sure the tags associated with training jobs stay sync with the tags associated with the hyperparameter tuning job that launched them, first call ListTrainingJobsForHyperParameterTuningJob (p. 808) for the hyperparameter tuning job to get a list of the training jobs that the hyperparameter tuning job launched. Then, call `AddTags` or `DeleteTags` for the hyperparameter tuning job and for each of the training jobs in the list of training jobs to add or delete the same set of tags for all of the jobs. The following Python example demonstrates this:

```
tuning_job_arn =
 smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName='MyTuningJob')
['HyperParameterTuningJobArn']
smclient.add_tags(ResourceArn=tuning_job_arn, Tags=[{'Key':'Env', 'Value':'Dev'}])
training_jobs = smclient.list_training_jobs_for_hyper_parameter_tuning_job(
    HyperParameterTuningJobName='MyTuningJob')['TrainingJobSummaries']
    for training_job in training_jobs:
        time.sleep(1) # Wait for 1 second between calls to avoid being throttled
        smclient.add_tags(ResourceArn=training_job['TrainingJobArn'], Tags=[{'Key':'Env',
 'Value':'Dev'}])
```

Amazon SageMaker Roles

As a managed service, Amazon SageMaker performs operations on your behalf on the AWS hardware that is managed by Amazon SageMaker. Amazon SageMaker can perform only operations that the user permits.

An Amazon SageMaker user can grant these permissions with an IAM role (referred to as an execution role). The user passes the role when making these API calls: CreateNotebookInstance (p. 656), CreateHyperParameterTuningJob (p. 638), CreateTrainingJob (p. 667), and CreateModel (p. 648).

You attach the following trust policy to the IAM role which grants Amazon SageMaker principal permissions to assume the role, and is the same for all of the execution roles:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "sagemaker.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

The permissions that you need to grant to the role vary depending on the API that you call. The following sections explain these permissions.

> **Note**
> Instead of managing permissions by crafting a permission policy, you can use the AWS-managed
> `AmazonSageMakerFullAccess` permission policy. The permissions in this policy are fairly
> broad, to allow for any actions you might want to perform in Amazon SageMaker. For a listing
> of the policy including information about the reasons for adding many of the permisions,
> see AmazonSageMakerFullAccess Policy (p. 506). If you prefer to create custom policies and
> manage permissions to scope the permissions only to the actions you need to perform with the
> execution role, see the following topics.

For more information about IAM roles, see IAM Roles in the *IAM User Guide*.

Topics

CreateNotebookInstance API: Execution Role Permissions

The permissions that you grant to the execution role for calling the `CreateNotebookInstance` API depend on what you plan to do with the notebook instance. If you plan to use it to invoke Amazon SageMaker APIs and pass the same role when calling the `CreateTrainingJob` and `CreateModel` APIs, attach the following permissions policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "ecr:SetRepositoryPolicy",
                "ecr:CompleteLayerUpload",
                "ecr:BatchDeleteImage",
                "ecr:UploadLayerPart",
                "ecr:DeleteRepositoryPolicy",
                "ecr:InitiateLayerUpload",
                "ecr:DeleteRepository",
                "ecr:PutImage",
                "ecr:CreateRepository",
                "cloudwatch:PutMetricData",
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
```

```
                    "cloudwatch:ListMetrics",
                    "logs:CreateLogGroup",
                    "logs:CreateLogStream",
                    "logs:DescribeLogStreams",
                    "logs:PutLogEvents",
                    "logs:GetLogEvents",
                    "s3:CreateBucket",
                    "s3:ListBucket",
                    "s3:GetBucketLocation",
                    "s3:GetObject",
                    "s3:PutObject",
                    "s3:DeleteObject",
                    "robomaker:CreateSimulationApplication",
                    "robomaker:DescribeSimulationApplication",
                    "robomaker:DeleteSimulationApplication",
                    "robomaker:CreateSimulationJob",
                    "robomaker:DescribeSimulationJob",
                    "robomaker:CancelSimulationJob",
                    "ec2:CreateVpcEndpoint",
                    "ec2:DescribeRouteTables",
                    "fsx:DescribeFileSystem",
                    "elasticfilesystem:DescribeMountTargets"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "codecommit:GitPull",
                    "codecommit:GitPush"
                ],
                "Resource": [
                    "arn:aws:codecommit:*:*:*sagemaker*",
                    "arn:aws:codecommit:*:*:*SageMaker*",
                    "arn:aws:codecommit:*:*:*Sagemaker*"
                ]
            }
            {
                "Effect": "Allow",
                "Action": [
                    "iam:PassRole"
                ],
                "Resource": "*",
                "Condition": {
                    "StringEquals": {
                        "iam:PassedToService": "sagemaker.amazonaws.com"
                    }
                }
            }
        ]
    }
}
```

To tighten the permissions, limit them to specific Amazon S3 and Amazon ECR resources, by replacing `"Resource": "*"`, as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*",
                "ecr:GetAuthorizationToken",
                "cloudwatch:PutMetricData",
```

```
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "sagemaker.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object1",
                "arn:aws:s3:::outputbucket/path",
                "arn:aws:s3:::inputbucket/object2",
                "arn:aws:s3:::inputbucket/object3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:::repository/my-repo1",
                "arn:aws:ecr:::repository/my-repo2",
                "arn:aws:ecr:::repository/my-repo3"
            ]
        }
    ]
}
```

If you plan to access other resources, such as Amazon DynamoDB or Amazon Relational Database Service, add the relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to the specific bucket that you specify as `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope `s3:GetObject` , `s3:PutObject`, and `s3:DeleteObject` permissions as follows:
  - Scope to the following values that you specify in a `CreateTrainingJob` request:

    `InputDataConfig.DataSource.S3DataSource.S3Uri`

    `OutputDataConfig.S3OutputPath`
  - Scope to the following values that you specify in a `CreateModel` request:

    `PrimaryContainer.ModelDataUrl`

    `SuplementalContainers.ModelDataUrl`
- Scope `ecr` permissions as follows:
  - Scope to the `AlgorithmSpecification.TrainingImage` value that you specify in a `CreateTrainingJob` request.
  - Scope to the `PrimaryContainer.Image` value that you specify in a `CreateModel` request:

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

CreateHyperParameterTuningJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateHyperParameterTuningJob` API request, you can attach the following permission policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Instead of the specifying `"Resource": "*"`, you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object",
                "arn:aws:s3:::outputbucket/path"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:::repository/my-repo"
        }
    ]
}
```

If the training container associated with the hyperparameter tuning job needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the `s3:GetObject` and `s3:PutObject` permissions to the following objects that you specify in the input and output data configuration in a `CreateHyperParameterTuningJob` request:

  `InputDataConfig.DataSource.S3DataSource.S3Uri`

  `OutputDataConfig.S3OutputPath`
- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateHyperParameterTuningJob` request.

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your hyperparameter tuning job, add the following permissions:

```
{
            "Effect": "Allow",
            "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DeleteNetworkInterfacePermission",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcs",
            "ec2:DescribeDhcpOptions",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
```

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
    "kms:Decrypt"
    ]
}
```

If you specify a KMS key in the output configuration of your hyperparameter tuning job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
    "kms:Encrypt"
    ]
}
```

If you specify a volume KMS key in the resource configuration of your hyperparameter tuning job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
    "kms:CreateGrant"
    ]
}
```

CreateTrainingJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateTrainingJob` API request, you can attach the following permission policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
```

```
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Instead of the specifying `"Resource": "*"`, you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object",
                "arn:aws:s3:::outputbucket/path"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:::repository/my-repo"
        }
    ]
}
```

If `CreateTrainingJob.AlgorithSpecifications.TrainingImage` needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the `s3:GetObject` and `s3:PutObject` permissions to the following objects that you specify in the input and output data configuration in a `CreateTrainingJob` request:

  `InputDataConfig.DataSource.S3DataSource.S3Uri`

  `OutputDataConfig.S3OutputPath`
- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateTrainingJob` request.

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your training job, add the following permissions:

```
{
            "Effect": "Allow",
            "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DeleteNetworkInterfacePermission",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcs",
            "ec2:DescribeDhcpOptions",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
```

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
    "kms:Decrypt"
    ]
}
```

If you specify a KMS key in the output configuration of your training job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
    "kms:Encrypt"
    ]
}
```

If you specify a volume KMS key in the resource configuration of your training job, add the following permissions:

```
{
```

```
        "Effect": "Allow",
        "Action": [
        "kms:CreateGrant"
        ]
}
```

CreateModel API: Execution Role Permissions

For an execution role that you can pass in a `CreateModel` API request, you can attach the following permission policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Instead of the specifying `"Resource":  "*"`, you can scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object",
                "arn:aws:s3:::inputbucket/object"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:::repository/my-repo",
                "arn:aws:ecr:::repository/my-repo"
             ]
        }
    ]
}
```

If `CreateModel.PrimaryContainer.Image` need to access other data sources, such as Amazon DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope S3 permissions to objects that you specify in the `PrimaryContainer.ModelDataUrl` in a CreateModel (p. 648) request.
- Scope Amazon ECR permissions to a specific registry path that you specify as the `PrimaryContainer.Image` and `SecondaryContainer.Image` in a `CreateModel` request.

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your model, add the following permissions:

```
{
            "Effect": "Allow",
            "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DeleteNetworkInterfacePermission",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcs",
            "ec2:DescribeDhcpOptions",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
```

AmazonSageMakerFullAccess Policy

The AmazonSageMakerFullAccess managed policy includes all of the necessary permissions to perform most actions in Amazon SageMaker. You can use attach this policy to any role that you pass to an Amazon SageMaker execution role. You can also create more narrowly-scoped policies if you want more granular control of the permissions that you grant to your execution role.

The following list explains why some of the categories of permissions in the `AmazonSageMakerFullAccess` policy are needed.

`application-autoscaling`

Needed for automatically scaling an Amazon SageMaker real-time inference endpoint.

`aws-marketplace`

Needed to view AWS AI Marketplace subscriptions.

```
cloudwatch
```

Needed to post CloudWatch metrics, interact with alarms, and upload CloudWatch Logs logs in your account.

```
codecommit
```

Needed for AWS CodeCommit integration with Amazon SageMaker notebook instances.

```
cognito
```

Needed for Amazon SageMaker Ground Truth to define your private workforce and work teams.

```
ec2
```

Needed to manage elastic network interfaces when you specify a Amazon VPC for your Amazon SageMaker jobs and notebook instances.

```
ec2:DescribeVpcs
```

All Amazon SageMaker services launch Amazon EC2 instances and require this permission set.

```
ecr
```

Needed to pull and store Docker artifacts for training and inference. This is required only if you use your own container in Amazon SageMaker.

```
elastic-inference
```

Needed to integrate Amazon Elastic Inference with Amazon SageMaker.

```
glue
```

Needed for inference pipeline pre-processing from within Amazon SageMaker notebook instances.

```
groundtruthlabeling
```

Needed for Amazon SageMaker Ground Truth.

```
iam:ListRoles
```

Needed to give the Amazon SageMaker console access to list available roles.

```
kms
```

Needed to give the Amazon SageMaker console access to list the avialable AWS KMS keys.

```
logs
```

Needed to allow Amazon SageMaker jobs and endpoints to publish log streams.

AWS Managed (Predefined) Policies for Amazon SageMaker

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate which permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon SageMaker:

- **AmazonSageMakerReadOnly** – Grants read-only access to Amazon SageMaker resources.
- **AmazonSageMakerFullAccess** – Grants full access to Amazon SageMaker resources and the supported operations. (This does not provide unrestricted S3 access, but supports buckets/objects with specific sagemaker tags.)

The following AWS managed policies can also be attached to users in your account:

- AdministratorAccess – Grants all actions for all AWS services and for all resources in the account.
- DataScientist – Grants a wide range of permissions to cover most of the use cases (primarily for analytics and business intelligence) encountered by data scientists.

You can review these permissions policies by signing in to the IAM console and searching for them.

You can also create your own custom IAM policies to allow permissions for Amazon SageMaker actions and resources as you need them. You can attach these custom policies to the IAM users or groups that require them.

Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference

When you are setting up access control and writing a permissions policy that you can attach to an IAM identity (an identity-based policy), use the following as a reference. The each Amazon SageMaker API operation, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource for which you can grant the permissions. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

> **Note**
> Except for the `ListTags` API, resource-level restrictions are not available on `List-` calls . Any user calling a `List-` API will see all resources of that type in the account.

To express conditions in your Amazon SageMaker policies, you can use AWS-wide condition keys. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

Amazon SageMaker API and Required Permissions for Actions

API Operation: AddTags (p. 620)

    Required Permissions (API Action): `sagemaker:AddTags`

    Resources: *

API Operation: CreateEndpoint (p. 632)

    Required Permissions (API Action): `sagemaker:CreateEndpoint`

    Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint/`*`endpointName`*

API Operation: CreateEndpointConfig (p. 635)

    Required Permissions (API Action): `sagemaker:CreateEndpointConfig`

    Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint-config/`*`endpointConfigName`*

API Operation: CreateModel (p. 648)

    Required Permissions (API Action): `sagemaker:CreateModel, iam:PassRole`

    Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:model/`*`modelName`*

API Operation: CreateLabelingJob (p. 643)

    Required Permissions (API Action): `sagemaker:CreateLabelingJob, iam:PassRole`

    Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:labeling-job/`*`labelingJobName`*

API Operation: CreateNotebookInstance (p. 656)

Required Permissions (API Action): `sagemaker:CreateNotebookInstance`,
`iam:PassRole`, `ec2:CreateNetworkInterface`, `ec2:AttachNetworkInterface`,
`ec2:ModifyNetworkInterfaceAttribute`, `ec2:DescribeAvailabilityZones`,
`ec2:DescribeInternetGateways`, `ec2:DescribeSecurityGroups`,
`ec2:DescribeSubnets`, `ec2:DescribeVpcs`, `kms:CreateGrant`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:notebook-`
`instance/`*`notebookInstanceName`*

API Operation: CreateTrainingJob (p. 667)

Required Permissions (API Action): `sagemaker:CreateTrainingJob, iam:PassRole`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:training-job/`*`trainingJobName`*

API Operation: CreateWorkteam (p. 678)

Required Permissions (API Action): `sagemaker:CreateWorkteam, sagemaker:CreateWorkteam`,
`cognito-idp:DescribeUserPoolClient, cognito-idp:UpdateUserPool, cognito-`
`idp:DescribeUserPool, cognito-idp:UpdateUserPoolClient`

Resources:`arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/private-crowd/`*`work team`*
`name`, `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/vendor-crowd/`*`work team`*
`name`, `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/public-crowd/`*`work team`*
`name`

API Operation: DeleteEndpoint (p. 683)

Required Permissions (API Action): `sagemaker:DeleteEndpoint`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint/`*`endpointName`*

API Operation: DeleteEndpointConfig (p. 685)

Required Permissions (API Action): `sagemaker:DeleteEndpointConfig`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint-`
`config/`*`endpointConfigName`*

API Operation: DeleteModel (p. 686)

Required Permissions (API Action): `sagemaker:DeleteModel`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:model/`*`modelName`*

API Operation: DeleteNotebookInstance (p. 690)

Required Permissions (API Action): `sagemaker:DeleteNotebookInstance`,
`ec2:DeleteNetworkInterface`, `ec2:DetachNetworkInterface`,
`ec2:DescribeAvailabilityZones`, `ec2:DescribeInternetGateways`,
`ec2:DescribeSecurityGroups`, `ec2:DescribeSubnets`, `ec2:DescribeVpcs`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:notebook-`
`instance/`*`notebookInstanceName`*

API Operation: DeleteTags (p. 693)

Required Permissions (API Action): `sagemaker:DeleteTags`

Resources: *

API Operation: DeleteWorkteam (p. 695)

Required Permissions (API Action): `sagemaker:DeleteWorkteam`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/*`

API Operation: DescribeEndpoint (p. 709)

Required Permissions (API Action): `sagemaker:DescribeEndpoint`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint/`*`endpointName`*

API Operation: DescribeEndpointConfig (p. 712)

Required Permissions (API Action): `sagemaker:DescribeEndpointConfig`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint-`
`config/`*`endpointConfigName`*

API Operation: DescribeLabelingJob (p. 721)

Required Permissions (API Action): `sagemaker:DescribeLabelingJob`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:labeling-job/`*`labelingJobName`*

API Operation: DescribeModel (p. 727)

Required Permissions (API Action): `sagemaker:DescribeModel`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:model/`*`modelName`*

API Operation: DescribeNotebookInstance (p. 734)

Required Permissions (API Action): `sagemaker:DescribeNotebookInstance`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:notebook-`
`instance/`*`notebookInstanceName`*

API Operation: DescribeSubscribedWorkteam (p. 742)

Required Permissions (API Action): `sagemaker:DescribeSubscribedWorkteam, aws-`
`marketplace:ViewSubscriptions`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/*`

API Operation: DescribeTrainingJob (p. 744)

Required Permissions (API Action): `sagemaker:DescribeTrainingJob`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:training-job/`*`trainingJobName`*

API Operation: DescribeWorkteam (p. 757)

Required Permissions (API Action): `sagemaker:DescribeWorkteam`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/*`

API Operation: CreatePresignedNotebookInstanceUrl (p. 665)

Required Permissions (API Action): `sagemaker:CreatePresignedNotebookInstanceUrl`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:notebook-`
`instance/`*`notebookInstanceName`*

API Operation: InvokeEndpoint (p. 853)

Required Permissions (API Action): `sagemaker:InvokeEndpoint`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint/`*`endpointName`*

API Operation: ListEndpointConfigs (p. 771)

Required Permissions (API Action): `sagemaker:ListEndpointConfigs`

Resources: *

API Operation: ListEndpoints (p. 774)

Required Permissions (API Action): `sagemaker:ListEndpoints`

Resources: *

API Operation: ListLabelingJobs (p. 781)

Required Permissions (API Action): `sagemaker:ListLabelingJobs`

Resources: *

API Operation: ListLabelingJobsForWorkteam (p. 785)

Required Permissions (API Action): `sagemaker:ListLabelingJobsForWorkteam`

Resources: *

API Operation: ListModels (p. 791)

Required Permissions (API Action): `sagemaker:ListModels`

Resources: *

API Operation: ListNotebookInstances (p. 797)

Required Permissions (API Action): `sagemaker:ListNotebookInstances`

Resources: *

API Operation: ListSubscribedWorkteams (p. 801)

Required Permissions (API Action): `sagemaker:ListSubscribedWorkteam`, `aws-marketplace:ViewSubscriptions`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/*`

API Operation: ListTags (p. 803)

Required Permissions (API Action): `sagemaker:ListTags`

Resources: *

API Operation: ListTrainingJobs (p. 805)

Required Permissions (API Action): `sagemaker:ListTrainingJobs`

Resources: *

API Operation: ListWorkteams (p. 814)

Required Permissions (API Action): `sagemaker:ListWorkteams`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/*`

API Operation: StartNotebookInstance (p. 824)

Required Permissions (API Action): `sagemaker:StartNotebookInstance`, `iam:PassRole`, `ec2:CreateNetworkInterface`, `ec2:AttachNetworkInterface`, `ec2:ModifyNetworkInterfaceAttribute`, `ec2:DescribeAvailabilityZones`, `ec2:DescribeInternetGateways`, `ec2:DescribeSecurityGroups`, `ec2:DescribeSubnets`, `ec2:DescribeVpcs`, `kms:CreateGrant`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:notebook-instance/`*`notebookInstanceName`*

API Operation: StopLabelingJob (p. 830)

Required Permissions (API Action): `sagemaker:StopLabelingJob`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:labeling-job/`*`labelingJobName`*

API Operation: StopNotebookInstance (p. 832)

Required Permissions (API Action): `sagemaker:StopNotebookInstance`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:notebook-instance/`*`notebookInstanceName`*

API Operation: StopTrainingJob (p. 834)

Required Permissions (API Action): `sagemaker:StopTrainingJob`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:training-job/`*`trainingJobName`*

API Operation: UpdateEndpoint (p. 840)

Required Permissions (API Action): `sagemaker:UpdateEndpoints`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:endpoint/`*`endpointName`*

API Operation: UpdateNotebookInstance (p. 844)

Required Permissions (API Action): `sagemaker:UpdateNotebookInstance, iam:PassRole`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:notebook-instance/`*`notebookInstanceName`*

API Operation: UpdateWorkteam (p. 850)

Required Permissions (API Action): `sagemaker:UpdateWorkteam`

Resources: `arn:aws:sagemaker:`*`region`*`:`*`account-id`*`:workteam/*`

Troubleshooting Amazon SageMaker Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon SageMaker and IAM.

Topics

- I Am Not Authorized to Perform an Action in Amazon SageMaker (p. 512)
- I Am Not Authorized to Perform iam:PassRole (p. 513)
- I Want to View My Access Keys (p. 513)
- I'm an Administrator and Want to Allow Others to Access Amazon SageMaker (p. 513)
- I Want to Allow People Outside of My AWS Account to Access My Amazon SageMaker Resources (p. 514)

I Am Not Authorized to Perform an Action in Amazon SageMaker

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a training job but does not have `sagemaker:CreateTrainingJob` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not
          authorized to perform: sagemaker:DescribeTrainingJob on resource: my-example-
widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `TrainingJob` resource using the `sagemaker:DescribeTrainingJob` action.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon SageMaker.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon SageMaker. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the *IAM User Guide*.

I'm an Administrator and Want to Allow Others to Access Amazon SageMaker

To allow others to access Amazon SageMaker, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon SageMaker.

To get started right away, see Creating Your First IAM Delegated User and Group in the *IAM User Guide*.

I Want to Allow People Outside of My AWS Account to Access My Amazon SageMaker Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon SageMaker supports these features, see How Amazon SageMaker Works with IAM (p. 479).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Logging and Monitoring

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. For more information, see Monitor Amazon SageMaker with Amazon CloudWatch (p. 461).

Amazon CloudWatch Logs enables you to monitor, store, and access your log files from Amazon EC2 instances, AWS CloudTrail, and other sources. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 466).

AWS CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon SageMaker. Using the information collected by CloudTrail, you can determine the request that was made to Amazon SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, Log Amazon SageMaker API Calls with AWS CloudTrail (p. 467).

> **Note**
> CloudTrail does not monitor calls to InvokeEndpoint (p. 853).

You can create rules in Amazon CloudWatch Events to react to status changes in status in an Amazon SageMaker training, hyperparameter tuning, or batch transform job. For more information, see React to Amazon SageMaker Job Status Changes with CloudWatch Events (p. 470).

Compliance Validation for Amazon SageMaker

Third-party auditors assess the security and compliance of Amazon SageMaker as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon SageMaker is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon SageMaker

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon SageMaker offers several features to help support your data resiliency and backup needs.

Infrastructure Security in Amazon SageMaker

As a managed service, Amazon SageMaker is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon SageMaker through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Topics
- Connect a Notebook Instance to Resources in a VPC (p. 516)
- Training and Inference Containers Run in Internet-Free Mode (p. 516)

Connect a Notebook Instance to Resources in a VPC

Amazon SageMaker notebook instances are internet-enabled by default. This allows you to download popular packages and notebooks, customize your development environment, and work efficiently. However, if you connect a notebook instance to your VPC, the notebook instance provides an additional avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the computer (in the form of a publicly available notebook or a publicly available source code library) could access your data. If you do not want Amazon SageMaker to provide internet access to your notebook instance, you can disable direct internet access when you specify a VPC for your notebook instance. If you disable direct internet access, the notebook instance won't be able to train or host models unless your VPC has an interface endpoint (PrivateLink) or a NAT gateway and your security groups allow outbound connections. For information about creating a VPC interface endpoint to use PrivateLink for your notebook instance, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 519). For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the in the *Amazon Virtual Private Cloud User Guide*. For information about security groups, see Security Groups for Your VPC.

Notebook Instances Provide the Best Experience for a Single User

An Amazon SageMaker notebook instance is designed to work best for an individual user. It is designed to give data scientists and other users the most power for managing their development environment. A notebook instance user has root access for installing packages and other pertinent software. We recommend that you exercise judgement when granting individuals access to notebook instances that are attached to a VPC that contains sensitive information. For example, you might grant a user access to a notebook instance with an IAM policy, as in the following example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
      "Resource": "arn:aws:sagemaker:region:account-id:notebook-instance/
myNotebookInstance"
    }
  ]
}
```

Training and Inference Containers Run in Internet-Free Mode

Amazon SageMaker training and deployed inference containers are internet-enabled by default. This allows containers to access external services and resources on the public internet as part of your training and inference workloads. However, this offers an avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the container (in the

form of a publicly available source code library) could access your data and transfer it to a remote host. If you use an Amazon VPC by specifying a value for the `VpcConfig` parameter when you call CreateTrainingJob (p. 667), CreateHyperParameterTuningJob (p. 638), or CreateModel (p. 648), you can protect your data and resources by managing security groups and restricting internet access from your VPC. However, this comes at the cost of additional network configuration, and has the risk of configuring your network incorrectly. If you do not want Amazon SageMaker to provide external network access to your training or inference containers, you can enable network isolation when you create your training job or model by setting the value of the `EnableNetworkIsolation` parameter to `True` when you call CreateTrainingJob (p. 667), CreateHyperParameterTuningJob (p. 638), or CreateModel (p. 648). If you enable network isolation, the containers are not able to make any outbound network calls, even to other AWS services such as Amazon S3. Additionally, no AWS credentials are made available to the container runtime environment. In the case of a training job with multiple instances, network inbound and outbound traffic is limited to the peers of each training container. Amazon SageMaker still performs download and upload operations against Amazon S3 using your Amazon SageMaker Execution Role in isolation from the training or inference container. Network isolation is required for training jobs and models run using resources from AWS Marketplace. Network isolation can be used in conjunction with a VPC. In this scenario, download and upload of customer data and model artifacts are routed via your VPC subnet. However, the training and inference containers themselves continue to be isolated from the network, and do not have access to any resource within your VPC or on the internet.

Network isolation is not supported by the following managed Amazon SageMaker containers as they require access to Amazon S3:

- TensorFlow
- Chainer
- PyTorch
- Scikit-learn
- Amazon SageMaker Reinforcement Learning

Amazon SageMaker Scans AWS Marketplace Training and Inference Containers for Security Vulnerabilities

To meet our security requirements, algorithms and model packages listed in AWS Marketplace are scanned for Common Vulnerabilities and Exposures (CVE). CVE is a list of publicly known information about security vulnerability and exposure. The National Vulnerability Database (NVD) provides CVE details such as severity, impact rating, and fix information. Both CVE and NVD are available for public consumption and free for security tools and services to use. For more information, see http://cve.mitre.org/about/faqs.html#what_is_cve.

Connect to Amazon SageMaker Through a VPC Interface Endpoint

You can connect directly to the Amazon SageMaker API or to the Amazon SageMaker Runtime through an interface endpoint in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use a VPC interface endpoint, communication between your VPC and the Amazon SageMaker API or Runtime is conducted entirely and securely within the AWS network.

> **Note**
> PrivateLink for Amazon SageMaker is not supported in the `us-gov-west-1` region.

The Amazon SageMaker API and Runtime support Amazon Virtual Private Cloud (Amazon VPC) interface endpoints that are powered by AWS PrivateLink. Each VPC endpoint is represented by one or more Elastic Network Interfaces (ENIs) with private IP addresses in your VPC subnets.

The VPC interface endpoint connects your VPC directly to the Amazon SageMaker API or Runtime without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the Amazon SageMaker API or Runtime.

You can create an interface endpoint to connect to Amazon SageMaker or to Amazon SageMaker Runtime with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see Creating an Interface Endpoint.

After you have created a VPC endpoint, you can use the following example CLI commands that use the `endpoint-url` parameter to specify interface endpoints to the Amazon SageMaker API or Runtime:

```
aws sagemaker list-notebook-instances --endpoint-
url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com

aws sagemaker list-training-jobs --endpoint-
url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com

aws sagemaker-runtime invoke-endpoint --endpoint-
url VPC_Endpoint_ID.runtime.sagemaker.Region.vpce.amazonaws.com  \
    --endpoint-name Endpoint_Name \
    --body "Endpoint_Body" \
    --content-type "Content_Type" \
            Output_File
```

If you enable private DNS hostnames for your VPC endpoint, you don't need to specify the endpoint URL. The Amazon SageMaker API DNS hostname that the CLI and Amazon SageMaker SDK use by default (https://api.sagemaker.*Region*.amazonaws.com) resolves to your VPC endpoint. Similarly, the Amazon SageMaker Runtime DNS hostname that the CLI and Amazon SageMaker Runtime SDK use by default (https://runtime.sagemaker.*Region*.amazonaws.com) resolves to your VPC endpoint.

The Amazon SageMaker API and Runtime support VPC endpoints in all AWS Regions where both Amazon VPC and Amazon SageMaker are available. Amazon SageMaker supports making calls to all of its Actions (p. 616) inside your VPC. The result `AuthorizedUrl` from the CreatePresignedNotebookInstanceUrl (p. 665) is not supported by Private Link. For information about how to enable PrivateLink for the authorized URL that users use to connect to a notebook instance, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 519).

To learn more about AWS PrivateLink, see the AWS PrivateLink documentation . Refer to VPC Pricing for the price of VPC Endpoints. To learn more about VPC and Endpoints, see Amazon VPC. For information about how to use identity-based AWS Identity and Access Management policies to restrict access to the Amazon SageMaker API and runtime, see Control Access to the Amazon SageMaker API by Using Identity-based Policies (p. 491).

Create a VPC Endpoint Policy for Amazon SageMaker

You can create a policy for Amazon VPC endpoints for Amazon SageMaker to specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling Access to Services with VPC Endpoints in the *Amazon VPC User Guide*.

> **Note**
> VPC endpoint policies aren't supported for Federal Information Processing Standard (FIPS) Amazon SageMaker runtime endpoints for InvokeEndpoint (p. 853).

The following example VPC endpoint policy specifies that all users who have access to the VPC interface endpoint are allowed to invoke the Amazon SageMaker hosted endpoint named `myEndpoint`.

```
{
   "Statement": [
       {
           "Action": "sagemaker:InvokeEndpoint",
           "Effect": "Allow",
           "Resource": "arn:aws:sagemaker:us-west-2:123456789012:endpoint/myEndpoint",
           "Principal": "*"
       }
   ]
}
```

In this example, the following are denied:

- Other Amazon SageMaker API actions, such as `sagemaker:CreateEndpoint` and `sagemaker:CreateTrainingJob`.
- Invoking Amazon SageMaker hosted endpoints other than `myEndpoint`.

> **Note**
> In this example, users can still take other Amazon SageMaker API actions from outside the VPC. For information about how to restrict API calls to those from within the VPC, see Control Access to the Amazon SageMaker API by Using Identity-based Policies (p. 491).

Connect to a Notebook Instance Through a VPC Interface Endpoint

You can connect to your notebook instance from your VPC through an interface endpoint in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use a VPC interface endpoint, communication between your VPC and the notebook instance is conducted entirely and securely within the AWS network.

Amazon SageMaker notebook instances support Amazon Virtual Private Cloud (Amazon VPC) interface endpoints that are powered by AWS PrivateLink. Each VPC endpoint is represented by one or more Elastic Network Interfaces (ENIs) with private IP addresses in your VPC subnets.

> **Note**
> Before you create an interface VPC endpoint to connect to a notebook instance, create an interface VPC endpoint to connect to the Amazon SageMaker API. That way, when users call CreatePresignedNotebookInstanceUrl (p. 665) to get the URL to connect to the notebook instance, that call also goes through the interface VPC endpoint. For information, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 517).

You can create an interface endpoint to connect to your notebook instance with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see Creating an Interface Endpoint. Make sure that you create an interface endpoint for all of the subnets in your VPC from which you want to connect to the notebook instance.

When you create the interface endpoint, specify **aws.sagemaker.*region*.notebook** as the service name. After you create a VPC endpoint, enable private DNS for your VPC endpoint. Anyone using the Amazon SageMaker API, the AWS CLI, or the console to connect to the notebook instance from within the VPC will connect to the notebook instance through the VPC endpoint instead of the public internet.

Amazon SageMaker notebook instances support VPC endpoints in all AWS Regions where both Amazon VPC and Amazon SageMaker are available.

Topics

- Connect Your Private Network to Your VPC (p. 520)
- Create a VPC Endpoint Policy for Amazon SageMaker Notebook Instances (p. 520)
- Restrict Access to Connections from Within Your VPC (p. 520)

Connect Your Private Network to Your VPC

To connect to your notebook instance through your VPC, you either have to connect from an instance that is inside the VPC, or connect your private network to your VPC by using an Amazon Virtual Private Network (VPN) or AWS Direct Connect. For information about Amazon VPN, see VPN Connections in the *Amazon Virtual Private Cloud User Guide*. For information about AWS Direct Connect, see Creating a Connection in the *AWS Direct Connect User Guide*.

Create a VPC Endpoint Policy for Amazon SageMaker Notebook Instances

You can create a policy for Amazon VPC endpoints for Amazon SageMaker notebook instances to specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling Access to Services with VPC Endpoints in the *Amazon VPC User Guide*.

The following example of a VPC endpoint policy specifies that all users that have access to the endpoint are allowed to access the notebook instance named `myNotebookInstance`.

```
{
   "Statement": [
       {
           "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
           "Effect": "Allow",
           "Resource": "arn:aws:sagemaker:us-west-2:123456789012:notebook-instance/
myNotebookInstance",
           "Principal": "*"
       }
   ]
}
```

Access to other notebook instances is denied.

Restrict Access to Connections from Within Your VPC

Even if you set up an interface endpoint in your VPC, individuals outside the VPC can connect to the notebook instance over the internet.

Important
If you apply an IAM policy similar to one of the following, users can't access the specified Amazon SageMaker APIs or the notebook instance through the console.

To restrict access to only connections made from within your VPC, create an AWS Identity and Access Management policy that restricts access to only calls that come from within your VPC. Then add that policy to every AWS Identity and Access Management user, group, or role used to access the notebook instance.

Note
This policy allows connections only to callers within a subnet where you created an interface endpoint.

```
{
    "Id": "notebook-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable Notebook Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreatePresignedNotebookInstanceUrl",
                "sagemaker:DescribeNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpc": "vpc-111bbaaa"
                }
            }
        }
    ]
}
```

If you want to restrict access to the notebook instance to only connections made using the interface endpoint, use the `aws:SourceVpce` condition key instead of `aws:SourceVpc`:

```
{
    "Id": "notebook-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable Notebook Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreatePresignedNotebookInstanceUrl",
                "sagemaker:DescribeNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:sourceVpce": [
                        "vpce-111bbccc",
                        "vpce-111bbddd"
                    ]
                }
            }
        }
    ]
}
```

Both of these policy examples assume that you have also created an interface endpoint for the Amazon SageMaker API. For more information, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 517). In the second example, one of the values for `aws:SourceVpce` is the ID of the interface endpoint for the notebook instance. The other is the ID of the interface endpoint for the Amazon SageMaker API.

The policy examples here include DescribeNotebookInstance (p. 734) because typically you would call `DescribeNotebookInstance` to make sure that the `NotebookInstanceStatus` is `InService` before you try to connect to it. For example:

```
aws sagemaker describe-notebook-instance \
                --notebook-instance-name myNotebookInstance
```

```
{
    "NotebookInstanceArn":
    "arn:aws:sagemaker:us-west-2:1234567890ab:notebook-instance/mynotebookinstance",
    "NotebookInstanceName": "myNotebookInstance",
    "NotebookInstanceStatus": "InService",
    "Url": "mynotebookinstance.notebook.us-west-2.sagemaker.aws",
    "InstanceType": "ml.m4.xlarge",
    "RoleArn":
    "arn:aws:iam::1234567890ab:role/service-role/AmazonSageMaker-
ExecutionRole-12345678T123456",
    "LastModifiedTime": 1540334777.501,
    "CreationTime": 1523050674.078,
    "DirectInternetAccess": "Disabled"
}
aws sagemaker create-presigned-notebook-instance-url --notebook-instance-name
 myNotebookInstance


{
    "AuthorizedUrl": "https://mynotebookinstance.notebook.us-west-2.sagemaker.aws?
authToken=AuthToken
}
```

For both of these calls, if you did not enable private DNS hostnames for your VPC endpoint, or if you are using a version of the AWS SDK that was released before August 13, 2018, you must specify the endpoint URL in the call. For example, the call to `create-presigned-notebook-instance-url` would be:

```
aws sagemaker create-presigned-notebook-instance-url
    --notebook-instance-name myNotebookInstance --endpoint-url
    VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
```

Connect Your Private Network to Your VPC

To call the Amazon SageMaker API and runtime through your VPC, you have to connect from an instance that is inside the VPC or connect your private network to your VPC by using an Amazon Virtual Private Network (VPN) or AWS Direct Connect. For information about Amazon VPN, see VPN Connections in the *Amazon Virtual Private Cloud User Guide*. For information about AWS Direct Connect, see Creating a Connection in the *AWS Direct Connect User Guide*.

Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud by default. However, training containers access AWS resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To control access to your data and training containers, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the *Amazon VPC User Guide*. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see VPC Flow Logs in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create training jobs by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates *elastic*

network interfaces (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your training containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the *Amazon VPC User Guide*.

> **Note**
> For training jobs, you can configure only subnets with a default tenancy VPC in which your instance runs on shared hardware. For more information on the tenancy attribute for VPCs, see Dedicated Instances.

Configure a Training Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the CreateTrainingJob (p. 667) API, or provide this information when you create a training job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your training containers. The ENIs provide your training containers with a network connection within your VPC that is not connected to the internet. They also enable your training job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateTrainingJob`:

```
VpcConfig: {
    "Subnets": [
        "subnet-0123456789abcdef0",
        "subnet-0123456789abcdef1",
        "subnet-0123456789abcdef2"
        ],
    "SecurityGroupIds": [
        "sg-0123456789abcdef0"
        ]
      }
```

Configure Your Private VPC for Amazon SageMaker Training

When configuring the private VPC for your Amazon SageMaker training jobs, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the *Amazon VPC User Guide*.

Topics

- Ensure That Subnets Have Enough IP Addresses (p. 523)
- Create an Amazon S3 VPC Endpoint (p. 523)
- Use a Custom Endpoint Policy to Restrict Access to S3 (p. 524)
- Configure Route Tables (p. 525)
- Configure the VPC Security Group (p. 525)
- Connect to Resources Outside Your VPC (p. 525)

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a training job. For more information, see VPC and Subnet Sizing for IPv4 in the *Amazon VPC User Guide*.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that training containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your training data unless you create a VPC endpoint that allows

access. By creating a VPC endpoint, you allow your training containers to access the buckets where you store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

To create an S3 VPC endpoint:

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**

3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the region where your VPC resides.

4. For **VPC**, choose the VPC you want to use for this endpoint.

5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.

6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see Use a Custom Endpoint Policy to Restrict Access to S3 (p. 524).

Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for Amazon S3. You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see Using Amazon S3 Bucket Policies.

Restrict Package Installation on the Training Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{
    "Statement": [
      {
        "Sid": "AmazonLinuxAMIRepositoryAccess",
        "Principal": "*",
        "Action": [
            "s3:GetObject"
        ],
        "Effect": "Deny",
        "Resource": [
            "arn:aws:s3:::packages.*.amazonaws.com/*",
            "arn:aws:s3:::repo.*.amazonaws.com/*"
        ]
      }
    ]
}

{
    "Statement": [
        { "Sid": "AmazonLinux2AMIRepositoryAccess",
          "Principal": "*",
          "Action": [
              "s3:GetObject"
              ],
          "Effect": "Deny",
```

```
            "Resource": [
                "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
                ]
            }
    ]
}
```

Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your training jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the *Amazon VPC User Guide*.

Configure the VPC Security Group

In distributed training, you must allow communication between the different containers in the same training job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group For information, see Security Group Rules.

Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, training jobs that use that VPC do not have access to resources outside your VPC. If your training job needs access to resources outside your VPC, provide access with one of the following options:

- If your training job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

- If your training job needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the *Amazon Virtual Private Cloud User Guide*.

Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC

Amazon SageMaker hosts models in an Amazon Virtual Private Cloud by default. However, models access AWS resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To avoid making your data and model containers accessible over the internet, we recommend that you create a private VPC and configure it to control access to them. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the *Amazon VPC User Guide*. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see VPC Flow Logs in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your

model containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the *Amazon VPC User Guide*.

Configure a Model for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the CreateModel (p. 648) API, or provide this information when you create a model in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your model to connect to resources in your private VPC.

> **Note**
> You must create at least two subnets in different availability zones in your private VPC, even if you have only one hosting instance.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateModel`:

```
VpcConfig: {
     "Subnets": [
         "subnet-0123456789abcdef0",
         "subnet-0123456789abcdef1",
         "subnet-0123456789abcdef2"
         ],
     "SecurityGroupIds": [
         "sg-0123456789abcdef0"
         ]
      }
```

Configure Your Private VPC for Amazon SageMaker Hosting

When configuring the private VPC for your Amazon SageMaker models, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the *Amazon VPC User Guide*.

Topics

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each model instance. For more information, see VPC and Subnet Sizing for IPv4 in the *Amazon VPC User Guide*.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

To create an Amazon S3 VPC endpoint:

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the AWS Region where your VPC resides.
4. For **VPC**, choose the VPC that you want to use for this endpoint.
5. For **Configure route tables**, choose the route tables that the endpoint will use. The VPC service automatically adds a route to each route table that you choose that points Amazon S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the Amazon S3 service by any user or service within the VPC. To restrict access further, choose **Custom**. For more information, see Use a Custom Endpoint Policy to Restrict Access to Amazon S3 (p. 527).

Use a Custom Endpoint Policy to Restrict Access to Amazon S3

The default endpoint policy allows full access to Amazon Simple Storage Service (Amazon S3) for any user or service in your VPC. To further restrict access to Amazon S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for Amazon S3.

You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see Using Amazon S3 Bucket Policies.

Restrict Package Installation on the Model Container with a Custom Endpoint Policy

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the model container. If you don't want users to install packages from those repositories, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{
    "Statement": [
      {
        "Sid": "AmazonLinuxAMIRepositoryAccess",
        "Principal": "*",
        "Action": [
            "s3:GetObject"
        ],
        "Effect": "Deny",
        "Resource": [
            "arn:aws:s3:::packages.*.amazonaws.com/*",
            "arn:aws:s3:::repo.*.amazonaws.com/*"
        ]
      }
    ]
}

{
    "Statement": [
        { "Sid": "AmazonLinux2AMIRepositoryAccess",
          "Principal": "*",
          "Action": [
              "s3:GetObject"
              ],
          "Effect": "Deny",
          "Resource": [
              "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
              ]
        }
```

```
    ]
}
```

Add Permissions for Endpoint Access for Containers Running in a VPC to Custom IAM Policies

The `SageMakerFullAccess` managed policy includes the permissions that you need to use models configured for Amazon VPC access with an endpoint. These permissions allow Amazon SageMaker to create an elastic network interface and attach it to model containers running in a VPC. If you use your own IAM policy, you must add the following permissions to that policy to use models configured for VPC access.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:CreateNetworkInterface"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information about the `SageMakerFullAccess` managed policy, see AmazonSageMakerFullAccess Policy (p. 506).

Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your models resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the *Amazon VPC User Guide*.

Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, models that use that VPC do not have access to resources outside your VPC. If your model needs access to resources outside your VPC, provide access with one of the following options:

- If your model needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.
- If your model needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the *Amazon Virtual Private Cloud User Guide*.

Give Batch Transform Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs batch transform jobs in an Amazon Virtual Private Cloud by default. However, model containers access AWS resources—such as the Amazon S3 buckets where you store your data and model artifacts—over the internet.

To control access to your model containers and data, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the *Amazon VPC User Guide*. Using a VPC helps to protect your model containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your model containers by using VPC flow logs. For more information, see VPC Flow Logs in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. You then specify the same model when you create a batch transform job. When you specify the subnets and security groups, Amazon SageMaker creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your model containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the *Amazon VPC User Guide*.

Configure a Batch Transform Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the CreateModel (p. 648) API, or provide this information when you create a transform job in the Amazon SageMaker console. Then specify the same model in the `ModelName` request parameter of the CreateTransformJob (p. 673) API, or when you create a transform job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your transform job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateModel`:

```
VpcConfig: {
      "Subnets": [
          "subnet-0123456789abcdef0",
          "subnet-0123456789abcdef1",
          "subnet-0123456789abcdef2"
          ],
      "SecurityGroupIds": [
          "sg-0123456789abcdef0"
          ]
        }
```

Configure Your Private VPC for Amazon SageMaker Batch Transform

When configuring the private VPC for your Amazon SageMaker batch transform jobs, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the *Amazon VPC User Guide*.

Topics

- Ensure That Subnets Have Enough IP Addresses (p. 530)

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a transform job. For more information, see VPC and Subnet Sizing for IPv4 in the *Amazon VPC User Guide*.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

To create an S3 VPC endpoint:

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the region where your VPC resides.
4. For **VPC**, choose the VPC you want to use for this endpoint.
5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see Use a Custom Endpoint Policy to Restrict Access to S3 (p. 530).

Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for Amazon S3. You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see Using Amazon S3 Bucket Policies.

Restrict Package Installation on the Model Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{
    "Statement": [
      {
        "Sid": "AmazonLinuxAMIRepositoryAccess",
        "Principal": "*",
        "Action": [
            "s3:GetObject"
```

```
            ],
            "Effect": "Deny",
            "Resource": [
                "arn:aws:s3:::packages.*.amazonaws.com/*",
                "arn:aws:s3:::repo.*.amazonaws.com/*"
            ]
        }
    ]
}

{
    "Statement": [
        { "Sid": "AmazonLinux2AMIRepositoryAccess",
          "Principal": "*",
          "Action": [
              "s3:GetObject"
              ],
          "Effect": "Deny",
          "Resource": [
              "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
              ]
        }
    ]
}
```

Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your batch transform jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the *Amazon VPC User Guide*.

Configure the VPC Security Group

In distributed batch transform, you must allow communication between the different containers in the same batch transform job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group For information, see Security Group Rules.

Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, batch transform jobs that use that VPC do not have access to resources outside your VPC. If your batch transform job needs access to resources outside your VPC, provide access with one of the following options:

- If your batch transform job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.
- If your batch transform job needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the *Amazon Virtual Private Cloud User Guide*.

Amazon SageMaker Ground Truth

To train a machine learning model, you need a large, high-quality, labeled dataset. Ground Truth helps you build high-quality training datasets for your machine learning models. With Ground Truth, you can use workers from either Amazon Mechanical Turk, a vendor company that you choose, or an internal, private workforce along with machine learning to enable you to create a labeled dataset. You can use the labeled dataset output from Ground Truth to train your own models. You can also use the output as a training dataset for an Amazon SageMaker model.

In order to automate labeling your training dataset, you can optionally use *automated data labeling*, a Ground Truth process that uses machine learning to decide which data needs to be labeled by humans. Automated data labeling may reduce the labeling time and manual effort required. For more information, see Using Automated Data Labeling (p. 539).

Use either pre-built or custom tools to assign the labeling tasks for your training dataset. A *labeling UI template* is a webpage that Ground Truth uses to present tasks and instructions to your workers. The Amazon SageMaker console provides built-in templates for labeling data. You can use these templates to get started , or you can build your own tasks and instructions by using our HTML 2.0 components. For more information, see Creating Custom Labeling Workflows (p. 557).

Use the workforce of your choice to label your dataset. You can choose your workforce from:

- The Amazon Mechanical Turk workforce of over 500,000 independent contractors worldwide.
- A private workforce that you create from your employees or contractors for handling data within your organization.
- A vendor company that you can find in the AWS Marketplace that specializes in data labeling services.

For more information, see Managing Your Workforce (p. 551).

You store your datasets in Amazon S3 buckets. The buckets contain three things: The data to be labeled, an input manifest file that Ground Truth uses to read the data files, and an output manifest file. The output file contains the results of the labeling job. For more information, see Using Input and Output Data (p. 543).

Events from your labeling jobs appear in Amazon CloudWatch under the `/aws/sagemaker/ LabelingJobs` group. CloudWatch uses the labeling job name as the name for the log stream.

Are You a First-time User of Ground Truth?

If you are a first-time user of Ground Truth, we recommend that you do the following:

1. **Read Getting started (p. 533)**—This section walks you through setting up your first Ground Truth labeling job.
2. **Explore other topics**—Depending on your needs, do the following:
    - **Create instruction pages for your labeling jobs**—Create a custom instruction page that makes it easier for your workers to understand the requirements of the job. For more information, see Creating Instruction Pages (p. 549).

- **Manage your labeling workforce**—Create new work teams and manage your existing workforce. For more information, see Managing Your Workforce (p. 551).
- **Create a custom UI**—Make it easier for your workers to quickly and correctly label your data by creating a custom UI for them to use. For more information, see Creating Custom Labeling Workflows (p. 557).

3. **See the API Reference (p. 616)**—This section describes operations to automate Ground Truth operations.

Getting started

To get started using Amazon SageMaker Ground Truth, follow the instructions in the following sections. The sections here explain how to use the console to create a labeling job, assign a public or private workforce, and send the labeling job to your workforce. You can also learn how to monitor the progress of a labeling job.

If you want to create a custom labeling job, see Creating Custom Labeling Workflows (p. 557) for instructions.

Before you create a labeling job, you must upload your dataset to an Amazon S3 bucket. For more information, see Using Input and Output Data (p. 543).

Topics

- Step 1: Before You Begin (p. 533)
- Step 2: Create a Labeling Job (p. 534)
- Step 3: Select Workers (p. 535)
- Step 4: Configure the Bounding Box Tool. (p. 535)
- Step 5: Monitoring Your Labeling Job (p. 536)

Step 1: Before You Begin

Before you begin using the Amazon SageMaker console to create a labeling job, you must set up the dataset for use. Do this:

1. Save two images at publicly available HTTP URLs. The images are used when creating instructions for completing a labeling task. The images should have an aspect ratio of around 2:1. For this exercise, the content of the images is not important.
2. Create an Amazon S3 bucket to hold the input and output files. The bucket must be in the same Region where you are running Ground Truth. Make a note of the bucket name because you use it during step 2.
3. Place 5–10 PNG images in the bucket.
4. Create a manifest file for the dataset and store it in the S3 bucket. Use these steps:

   a. Using a text editor, create a new text file.

   b. Add a line similar to the following for each image file in your dataset:

   ```
   {"source-ref": "s3://bucket/path/imageFile.png"}
   ```

   Add one line for each PNG file in your S3 bucket.

   c. Save the file in the S3 bucket containing your source files. Record the name because you use it in step 2.

> **Note**
> It is not necessary to store the manifest file in the same bucket as the source file. You
> use the same bucket in this exercise because it is easier.

For more information, see Input Data (p. 543).

Assign the following permissions policy to the user that is creating the labeling job:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "sagemakergroundtruth",
            "Effect": "Allow",
            "Action": [
                "cognito-idp:CreateGroup",
                "cognito-idp:CreateUserPool",
                "cognito-idp:CreateUserPoolDomain",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:UpdateUserPool"
            ],
            "Resource": "*"
        }
    ]
}
```

Next

Step 2: Create a Labeling Job

In this step you use the console to create a labeling job. You tell Amazon SageMaker Ground Truth the
Amazon S3 bucket where the manifest file is stored and configure the parameters for the job. For more
information about storing data in an Amazon S3 bucket, see Using Input and Output Data (p. 543).

To create a labeling job

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. From the left navigation, choose **Labeling jobs**.
3. Choose **Create labeling job** to start the job creation process.
4. In the **Job overview** section, provide the following information:

    - **Job name** — Give the labeling job a name that describes the job. This name is shown in your job
      list. The name must be unique in your account in an AWS Region.
    - **Label attribute name** — Leave this unchecked as the default value is the best option for this
      introductory job.
    - **Input dataset location** — Enter the S3 location of the manifest file that you created in step 1.
    - **Output dataset location** — the location where your output data is written.
    - **IAM role** — Create or choose an IAM role with the SageMakerFullAccess IAM policy attached.
5. In the **Task type** section, for the **Dataset type** field, choose **Bounding box** as the task type.

6. Choose **Next** to move on to configuring your labeling job.

Next

Step 3: Select Workers

In this step you choose a workforce for labeling your dataset. You can create your own private workforce or you can use the Amazon Mechanical Turk workforce. If you create a private workforce in this step you won't be able to import your Amazon Cognito user pool later. For more information, see Managing a Private Workforce (p. 553). Use the Amazon Mechanical Turk workforce for this exercise instead.

You can create a private workforce to test Amazon SageMaker Ground Truth. Use email addresses to invite the members of your workforce.

To create a private workforce

1. In the **Workers** section, choose **Private**.
2. If this is your first time using a private workforce, in the **Email addresses** field, enter up to 100 email addresses. The addresses must be separated by a comma. You should include your own email address so that you are part of the workforce and can see data object labeling tasks.
3. In the **Organization name** field, enter the name of your organization. This information is used to customize the email sent to invite a person to your private workforce.
4. In the **Contact email** field enter an email address that members of the workforce use to report problems with the task.

If you choose to use the Amazon Mechanical Turk workforce to label the dataset, you are charged for labeling tasks completed on the dataset.

To use the Amazon Mechanical Turk workforce

1. In the **Workers** section, choose **Public**.
2. Choose **The dataset does not contain PII** to acknowledge that the dataset does not contain any personally identifiable information.
3. Choose **The dataset does not contain adult content.** to acknowledge that the sample dataset has no adult content.
4. Review and accept the statement that the dataset will be viewed by the public workforce.

Next

Step 4: Configure the Bounding Box Tool.

Finally you configure the bounding box tool to give instructions to your workers. You can configure a task title that describes the task and provides high-level instructions for the workers. You can provide both quick instructions and full instructions. Quick instructions are displayed next to the image to be labeled. Full instructions contain detailed instructions for completing the task. In this example, you only provide quick instructions. You can see an example of full instructions by choosing **Full instructions** at the bottom of the section.

To configure the bounding box tool

1. In the **Task description** field type in brief instructions for the task. For example:

   `Draw a box around any `*`objects`*` in the image.`

   Replace *objects* with the name of an object that appears in your images.

2. In the **Labels** field, type a category name for the objects that the worker should draw a bounding box around. For example, if you are asking the worker to draw boxes around football players, you could use "FootballPlayer" in this field.

3. The **Short instructions** section enables you to create instructions that are displayed on the page with the image that your workers are labeling. We suggest that you include an example of a correctly drawn bounding box and an example of an incorrectly drawn box. To create your own instructions, use these steps:

   a. Select the text between **GOOD EXAMPLE** and the image placeholder. Replace it with the following text:

      `Draw the box around the object with a small border.`

   b. Select the first image placeholder and delete it.

   c. Choose the image button and then enter the HTTPS URL of one of the images that you created in step 1.

   d. Select the text between **BAD EXAMPLE** and the image placeholder. Replace it with the following text:

      `Don't make the bounding box too large or cut into the object.`

   e. Select the second image placeholder and delete it.

   f. Choose the image button and then enter the HTTPS URL of the other image that you created in step 1.

Configuration of your labeling job is complete. To start your job, choose **Submit**.

Next

Step 5: Monitoring Your Labeling Job

After you create your labeling job, you see a list of all the jobs that you have created. You can use this list to monitor that status of your labeling jobs. The list has the following fields:

- **Name**—The name that you assigned the job when you created it.
- **Status**—The completion status of the job. The status can be one of Complete, Failed, In progress, or Stopped.
- **Labeled objects/total**—Shows the total number of objects in the labeling job and how many of them have been labeled.
- **Creation time**—The date and time that you created the job.

You can also clone, chain, or stop a job. Select a job and then select one of the following from the **Actions** menu:

- **Clone**—Creates a new labeling job with the configuration copied from the selected job. You can clone a job when you want to change to the job and run it again. For example, you can clone a job that was

sent to a private workforce so that you can send it to the Amazon Mechanical Turk workforce. Or you
can clone a job to rerun it against a new dataset stored in the same location as the original job.

- **Chain**—Creates a new labeling job that can build upon the data and models (if any) of a stopped,
  failed, or completed job. For more information about the use cases and how to use it, see Chaining
  labeling jobs (p. 540).
- **Stop**—Stops a running job. You cannot restart a stopped job. You can clone a job to start over or chain
  the job to continue from where it left off. Labels for any already labeled objects are written to the
  output file location. For more information, see Output Data (p. 545).

Data Labeling

Amazon SageMaker Ground Truth manages sending your data objects to workers to be labeled. Labeling
each data object is a *task*. Workers complete each task until the entire labeling job is complete. Ground
Truth divides the total number of tasks into smaller *batches* that are sent to workers. A new batch is sent
to workers when the previous one is finished.

Ground Truth provides two features that help improve the accuracy of your data labels and reduce the
total cost of labeling your data.

The first feature is *annotation consolidation*. This helps to improve the accuracy of your data object's
labels. It combines the results of multiple worker's annotation tasks into one high-fidelity label.

The second feature is *automated data labeling*. This uses machine learning to label portions of your data
automatically without having to send them to human workers.

Topics
- Batches for Labeling Tasks (p. 537)
- Annotation Consolidation (p. 537)
- Using Automated Data Labeling (p. 539)
- Chaining labeling jobs (p. 540)

Batches for Labeling Tasks

Amazon SageMaker Ground Truth sends data objects to your workers in batches. There are one or more
tasks for each data object. For each task, a worker annotates one of your data objects. A batch provides
the following:

- It sets the number of data objects that are available to workers. After the objects are annotated
  another batch is sent.
- It breaks the work into smaller chunks to avoid overloading your workforce.
- It provides chunks of data for the iterative training of automated labeling models.

Ground Truth first sends a batch of 10 tasks to your workers. It uses this small batch to set up the
labeling job and to make sure that the job is correctly configured.

After the small batch, Ground Truth sends larger batches to your workers. You can configure the
batch size when you create the job using the CreateLabelingJob (p. 643). When you use the Amazon
SageMaker console, your job uses 1,000 tasks in each batch.

Annotation Consolidation

Annotation consolidation combines the annotations of two or more workers into a single label for your
data objects. An *annotation* is the result of a single worker. *Annotation consolidation* combines multiple

annotations from different workers to come up with a probabilistic estimate of what the true label should be. The *label* is assigned to each object in the dataset. Each object in the dataset typically has multiple annotations but only one label or set of labels.

You can decide how many workers should annotate each object in your dataset. More workers can increase the accuracy of your labels but also increases the cost of labeling. Amazon SageMaker Ground Truth uses the following defaults in the Amazon SageMaker console. When you use the CreateLabelingJob (p. 643) operation, you set the number of workers that should annotate each data object using the `NumberOfHumanWorkersPerDataObject` parameter.

- **Text classification**—3 workers
- **Image classification**—3 workers
- **Bounding boxes**—5 workers
- **Semantic segmentation**—3 workers
- **Named entity recognition**—3 workers

You can override the default number of workers that label a data object using the console or the CreateLabelingJob (p. 643) operation.

Ground Truth provides an annotation consolidation function for each of its predefined labeling tasks: Bounding box, image classification, semantic segmentation, and text classification. These are the functions:

- Multi-class annotation consolidation for image and text classification uses a variant of the Expectation Maximization approach to annotations. It estimates parameters for each worker and uses Bayesian inference to estimate the true class based on the class annotations from individual workers.
- Bounding box annotation consolidates bounding boxes from multiple workers. This function finds the most similar boxes from different workers based on the Jaccard index, or intersection over union, of the boxes and averages them.
- Semantic segmentation annotation consolidation treats each pixel in a single image as a multi-class classification. This function treats the pixel annotations from workers as "votes," with more information from surrounding pixels incorporated by applying a smoothing function to the image.
- Named entity recognition clusters text selections by Jaccard similarity and calculates selection boundaries based on the mode, or median if the mode isn't clear. The label resolves to the most assigned entity label in the cluster, breaking ties by random selection.

> **Note**
> If you want to run worker responses through different algorithms on your own, that data is stored in the `[project-name]/annotations/worker-response` folder of the Amazon S3 bucket where you direct the job output.

Creating Your Own Annotation Consolidation Function

There are many possible approaches for writing an annotation consolidation function. The approach that you take depends on the nature of the annotations to consolidate. Broadly, consolidation functions look at the annotations from workers, measure the similarity between them, and then use some form of probabilistic judgment to determine what the most probable label should be.

Assessing Similarity

To assess the similarity between labels, you can use one of the following strategies or you can use one that meets your data labeling needs:

- For label spaces that consist of discrete, mutually exclusive categories, such as multi-class classification, assessing similarity can be straightforward. Discrete labels either match or not.

- For label spaces that don't have discrete values, such as bounding box annotations, find a broad measure of similarity. For bounding boxes, one such measure is the Jaccard index. This measures the ratio of the intersection of two boxes with the union of the boxes to assess how similar they are. For example, if there are three annotations, then there can be a function that determines whichannotations represent the same object and should be consolidated.

Assessing the Most Probable Label

With one of the above strategies in mind, make some sort of probabilistic judgment on what the consolidated label should be. In the case of discrete, mutually exclusive categories this can be straightforward. One of the most common ways to do this is to take the results of a majority vote between the annotations. This weights the annotations equally.

Some approaches attempt to estimate the accuracy of different annotators and weight their annotations in proportion to the probability of correctness. An example of this is the Expectation Maximization method, which is used in the default Ground Truth consolidation function for multi-class annotations.

For more information about creating an annotation consolidation function, see .

Using Automated Data Labeling

Ground Truth can use *active learning* to automate the labeling of your input data. Active learning is a machine learning technique that identifies data that should be labeled by your workers.

Automated data labeling is optional. Turn it on when you create a labeling job. Automated data labeling incurs Amazon SageMaker training and inference costs, but it helps to reduce the cost and time that it takes to label your dataset compared to humans alone.

Use automated data labeling on large datasets. The neural networks used with active learning require a significant amount of data for every new dataset. With larger datasets, there is more potential to automatically label the data and therefore reduce the total cost of labeling. We recommend that you use thousands of data objects when using automated data labeling. The system minimum for automated labeling is 1,250 objects, but to get a meaningful amount of your data automatically labeled, we strongly suggest a minimum with 5,000 or more objects.

The potential benefit of automated data labeling also depends on the accuracy that you require. Higher accuracy levels generally reduce the number of data objects that are automatically labeled.

When Amazon SageMaker Ground Truth starts an automated data labeling job, it first selects a random sample of the input data. Then it sends the sample to human workers. When the labeled data are returned, Ground Truth uses this set of data as validation data. It is used to validate the machine learning models that Ground Truth trains for automated data labeling.

Next, Ground Truth runs an Amazon SageMaker batch transform using the validation set. This generates a quality metric that Ground Truth uses to estimate the potential quality of auto-labeling the rest of the unlabeled data.

Ground Truth next runs an Amazon SageMaker batch transform on the unlabeled data in the dataset. Any data where the expected quality of automatically labeling the data is above the requested level of accuracy is considered labeled.

After performing the auto-labeling step, Ground Truth selects a new sample of the most ambiguous unlabeled data points in the dataset. It sends those to human workers. Ground Truth uses the existing labeled data and this additional labeled data from human workers to train a new model. The process is repeated until the dataset is fully labeled.

For automated semantic segmentation, please note these job limits

- Label Categories (max): 20
- Dataset Size (max): 20k items
- Image Resolution (max): 720p (1280 x 720 pixels)

Ensure the automated-labeling model is ready for production use
The model generatphoneed by your labeling job needs fine-tuning and/or testing before you use it in production. Fine-tune the model generated by Ground Truth (or create and tune another supervised model of your choice) on the dataset produced by your labeling job. Optimize the model's architecture and hyperparameters. If you decide to use the model for inference without fine-tuning, we strongly recommend making sure its accuracy is evaluated on a representative (e.g. randomly selected) subset of the dataset labeled with Ground Truth and matches your expectations.

Amazon EC2 Instances Required for Automated Data Labeling

To run automated data labeling, Ground Truth requires the following Amazon EC2 resources for training and batch inference jobs:

Automated labeling action	Training instance type	Inference instance type
Image classification	ml.p3.2xlarge*	ml.c5.xlarge
Object detection	ml.p3.2xlarge*	ml.c5.4large
Text classification	ml.c5.2xlarge	ml.m4.xlarge
Semantic Segmentation	ml.p3.2xlarge*	ml.p3.2xlarge*

* ml.p2.8xlarge is substituted in the following regions: Mumbai (ap-south-1)

A note about pricing
Automated labeling incurs two separate charges: the per item charge (Ground Truth pricing), and the charge for the Amazon EC2 instance required to run the model (Amazon EC2 pricing).

These instances are managed by Ground Truth. They are created, configured, and destroyed as needed to perform your job. They do not appear in your Amazon EC2 instance dashboard.

Chaining labeling jobs

Amazon SageMaker Ground Truth can reuse datasets from prior jobs in two ways: cloning and chaining.

Cloning is a relatively straightforward operation. Cloning copies the set-up of a prior labeling job and allows you to make additional changes, before setting it to run.

Chaining is a more complex operation. Chaining uses not only the setup of the prior job, but the results. This allows you to continue an incomplete job,and add labels or data objects to a completed job.

When it comes to the data being processed:

- **Cloning** — uses the prior job's *input* manifest as the new job's input manifest.
- **Chaining** — uses the prior job's *output* manifest as the new job's input manifest.

Chaining labeling jobs

Some situations where chaining is useful include:

- Continue a labeling job that was manually stopped.
- Continue a labeling job that failed mid-job, with issues fixed..
- Switch to automated labeling after manually labeling part of a job (or vice-versa).
- Add more data objects to a completed job and start the job from there.
- Add another annotation to a completed job. For example, you have a collection of phrases labeled for topic, then want to run the set again, categorizing them by the topic's implied audience.

In Amazon SageMaker Ground Truth you can configure a chained labeling job via either the console or API.

Key Term: Label attribute name

The label attribute name (`LabelAttributeName` in the API) is a string used as the key for the key-value pair formed with the label that a worker assigns to the data object.

There are a few rules for the label attribute name.

- It cannot end with `-metadata`.
- The names `source` and `source-ref` are reserved and cannot be used.
- Semantic-segmentation labeling jobs require it to end with `-ref`. All other labeling jobs require it to *not* end with `-ref`. The adding of `-ref` is managed automatically for you in jobs configured via the console.
- If you're using the same label attribute name from the originating job and you configure the chained job to use auto-labeling, then if it had been in auto-labeling mode at any point, the model from the originating job is used.

In an output manifest, it can appear something like this:

```
"source-ref": "<S3 URI>",
"<label attribute name>": {
  "annotations": [{
    "class_id": 0,
    "width": 99,
    "top": 87,
    "height": 62,
    "left": 175
  }],
  "image_size": [{
    "width": 344,
    "depth": 3,
    "height": 234
  }]
},
"<label attribute name>-metadata": {
  "job-name": "<job name>",
  "class-map": {
    "0": "<label attribute name>"
  },
  "human-annotated": "yes",
  "objects": [{
    "confidence": 0.09
  }],
  "creation-date": "<timestamp>",
```

```
    "type": "groundtruth/object-detection"
}
```

If you're creating a job in the console, the job name is used as the label attribute name for the job if you don't explicitly set the label attribute name value.

Starting a chained job in the console

Select a stopped, failed, or completed labeling job from the list of your existing jobs. This enables the **Actions** menu.

From the **Actions** menu, select **Chain**.

Job overview panel

In the **Job overview** panel, a new **Job name** is set based on the title of the job from which you are chaining this one. You can change it.

You may also specify a label attribute name different from the labeling job name.

If you're chaining from a completed job, the label attribute name uses the name of the new job you're configuring. To change the name, select the check box.

If you're chaining from a stopped or failed job, the label attribute name uses to the name of the job from which you're chaining. Its easy to see and edit the value because the name check box is checked.

> **Attribute label naming considerations**
>
> - **The default** uses the label attribute name Ground Truth has selected. All data objects without data connected to that label attribute name are labeled.
> - **Using a label attribute name** not present in the manifest causes the job to process *all* the objects in the dataset.

The **input dataset location** in this case is automatically selected as the output manifest of the chained job. The input field is not available, so you cannot change it.

> **Adding data objects to a labeling job**
> You cannot specify an alternate manifest file. Manually edit the output manifest from the previous job to add new items before starting a chained job. The S3 URI helps you locate where you are storing the manifest in your S3 bucket. Download the manifest file from there, edit it locally on your computer, then upload the new version to replace it. Make sure you are not introducing errors during editing. We recommend you use JSON linter to check your JSON. Many popular text editors and IDEs have linter plugins available.

Starting a chained job with the API

The procedure is almost the same as setting up a new labeling job with `CreateLabelingJob`, except for two primary differences.

- **Manifest location:** Rather than use your original manifest from the prior job, the value for the `ManifestS3Uri` in the `DataSource` should point to the S3 URI of the *output manifest* from the prior labeling job.
- **Label attribute name:** Setting the correct `LabelAttributeName` value is important here. As pointed out, this is the key portion of a key-value pair where labeling data is the value. Sample use cases include:
  - **Adding new or more-specific labels to a completed job** — set a new label attribute name.
  - **Labeling the unlabeled items from a prior job** — use the label attribute name from the prior job.

Using a partially labeled dataset

You can get some chaining benefits if you use an augmented manifest that has already been partially labeled. Check the **Label attribute name** check box and set the name so that it matches the name in your manifest.

If you're using the API, the instructions are the same as starting a chained job. However, be sure to upload your manifest to an S3 bucket and use it instead of using the output manifest from a prior job.

The **Label attribute name** value in the manifest has to conform to the naming considerations discussed above.

Using Input and Output Data

The input data that your provide Amazon SageMaker Ground Truth is sent to your workers for labeling. You can choose the data to send to your workers by creating a manifest file that defines the data to label.

The output data is the result of your labeling job. The output data file contains one entry that specifies the label for each object in the input dataset.

Topics

Input Data

The input data are the data objects that you send to your workforce to be labeled. Each object in the input data is described in a manifest file. Each line in the manifest is an entry containing an object to label and may contain labels from previous jobs.

- The input data is stored in an Amazon S3 bucket. The bucket must be in the same region as you are running Amazon SageMaker Ground Truth. You must give access to Amazon SageMaker for the data to be read. In order to read the data, give access to Amazon SageMaker. For more information about Amazon S3 buckets, see Working with Amazon S3 buckets.
- The manifest file must be in the same region the data files but does not need to be in the same location as the data files. It can be in any Amazon S3 bucket accessible to the role that you assigned to Ground Truth when you created the labeling job with either the console or the CreateLabelingJob (p. 643) operation.

The manifest is a UTF-8 encoded file where each line is a complete and valid JSON object. Each line is delimited by a standard line break, \n or \r\n. Since each line must be a valid JSON object, you can't have unescaped line break characters. For more information about the data format, see JSON Lines.

Limits: Each JSON object in the manifest file can be no larger than 100k characters and no single attribute within the object can be larger than 20,000 characters. Attribute names cannot begin with `$` (dollar sign).

Each JSON object in the manifest file must contain a key, either `source-ref` or `source`. The value of the keys are interpreted as follows:

- source-ref—The source of the object is the S3 object specified in the value. This can be used when the object is a binary object, such as an image, or when you have text in individual files.
- source—The source of the object is the value. This can be used when the object is a text value.

You use the `source-ref` key for image files for a bounding box or semantic segmentation labeling job. Each image file must be:

- 6 Mb or smaller
- 1920 x 1080 pixels or smaller for semantic segmentation

The following is an example of a manifest file for files stored in an S3 bucket:

```
{"source-ref": "S3 bucket location 1"}
{"source-ref": "S3 bucket location 2"}
    ...
{"source-ref": "S3 bucket location n"}
```

The following is an example of a manifest file with the input data stored in the manifest:

```
{"source": "Lorem ipsum dolor sit amet"}
{"source": "consectetur adipiscing elit"}
    ...
{"source": "mollit anim id est laborum"}
```

You can include other key-value pairs in the manifest file. These pairs are passed to the output file unchanged. This is useful when you want to pass information between your applications. For more information, see Output Data (p. 545).

Filtering and Selecting Data (Console)

You can use the Amazon SageMaker console to select a portion of your dataset for labeling. The data must be stored in an Amazon S3 bucket. You have three options:

- Use the full dataset.
- Choose a randomly selected sample of the dataset.
- Specify a subset of the dataset using a query.

Using the Full Dataset

When you choose to use the full dataset you must provide a manifest file for your data objects. You can provide the S3 bucket location of the manifest file or you can have the Amazon SageMaker console create the file for you. Choose **Create a manifest file** to create the file. The file is stored in the S3 bucket specified in the **Input data location** field.

Choosing a Random Sample

Use a random sample of your dataset when you want to label a random subset of your data. The dataset is stored in the S3 bucket specified in the **Input dataset location** field.

Once you have specified the percentage of data objects that you want to include in the sample, choose **Create subset**. The Amazon SageMaker console randomly picks the data objects for your labeling job. Once the objects are selected, choose **Use this subset**.

The Amazon SageMaker console create a manifest file for the selected data objects. The **Input dataset location** field is modified to point to the new manifest file.

Specifying a Subset

You can specify a subset of your data objects using an Amazon S3 `SELECT` query on the object file names.

The `SELECT` statement of the SQL query is defined for you. You provide the `WHERE` clause to specify which data objects should be returned.

For more information about the Amazon S3 `SELECT` statement, see [Selecting Content from Objects](#)

Choose **Create subset** to start the selection, and then choose **Use this subset** to use the selected data.

The Amazon SageMaker console create a manifest file for the selected data objects. The **Input dataset location** field is modified to point to the new manifest file.

Output Data

The output from a labeling job is placed in the location that you specified in the console or in the call to the [CreateLabelingJob (p. 643)](#) operation.

Each line in the output data file is identical to the manifest file with the addition of an attribute and value for the label assigned to the input object. The attribute name for the value is defined in the console or in the call to the `CreateLabelingJob` operation. You can't use `-metadata` in the label attribute name. If you are running a semantic segmentation job, the label attribute must end with `-ref`. For any other type of job, the attribute name can't end with `-ref`.

The output of the labeling job is the value of the key/value pair with the label. The label and the value overwrites any existing JSON data in the input file with the new value.

For example, the following is the output from an image classification labeling job where the input data files were stored in an Amazon S3 bucket and the label attribute name was defined as "sport". In this example the JSON object is formatted for readability, in the actual output file the JSON object is on a single line. For more information about the data format, see [JSON Lines](#).

```
{
    "source-ref": "S3 bucket location",
    "sport":0,
    "sport-metadata":
    {
        "class-name": "football",
        "confidence": 0.8,
        "type":"groundtruth/image-classification",
        "job-name": "identify-sport",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256"
    }
}
```

The value of the label can be any valid JSON. In this case the label's value is the index of the class in the classification list. Other job types, such as bounding box, have more complex values.

Any key-value pair in the input manifest file other than the label attribute is unchanged in the output file. You can use this to pass data to your application.

The output from a labeling job can be used as the input to another labeling job. You can use this when you are chaining together labeling jobs. For example, you can send one labeling job to determine the sport that is being played. Then you send another using the same data to determine if the sport is being played indoors or outdoors. By using the output data from the first job as the manifest for the second job, you can consolidate the results of the two jobs into one output file for easier processing by your applications.

The output data file is written to the output location periodically while the job is in progress. These intermediate files contain one line for each line in the manifest file. If an object is labeled, the label is

included, if the object has not been labeled it is written to the intermediate output file identically to the manifest file.

Output Directories

Ground Truth creates several directories in your Amazon S3 output path. These directories contain the results of your labeling job and other artifacts of the job. The top-level directory for a labeling job is given the same name as your labeling job, the output directories are placed beneath it. For example, if you named your labeling job **find-people** you output would be in the following directories:

```
s3://bucket/find-people/activelearning
s3://bucket/find-people/annotations
s3://bucket/find-people/inference
s3://bucket/find-people/manifests
s3://bucket/find-people/training
```

Each directories contain the following output:

activelearning Directory

The `activelearning` directory is only present when you are using automated data labeling. It contains the input and output validation set for automated data labeling, and the input and output folder for automatically labeled data.

annotations Directory

The `annotations` directory contains all of the annotations made by the workforce. These are the responses from individual workers that have not been consolidated into a single label for the data object.

There are three subdirectories in the `annotations` directory. The first, `worker-response` contains the responses from individual workers. There may be more than one annotation for each data object in this directory, depending on how many workers you want to annotate each object.

The second, `consolidated-annotation` contains information required to consolidate the annotations in the current batch into labels for your data objects.

The third, `intermediate`, contains the output manifest for the current batch with any completed labels. This file is updated as the label for each data object is completed.

inference Directory

The `inference` directory contains the input and output files for the Amazon SageMaker batch transform used while labeling data objects.

manifest Directory

The `manifest` directory contains the output manifest from your labeling job. There are two subdirectories in the manifest directory, `output` and `intermediate`. The `output` directory contains the output manifest file for your labeling job. The file is named `output.manifest`.

The other directory, `intermediate`, contains the results of labeling each batch of data objects. The intermediate data is in a directory numbered for each iteration. An iteration directory contains an output.manifest file that contains the results of that iteration and all previous iterations.

intermediate Directory

The `training` directory contains the input and output files used to train the automated data labeling model.

Confidence Score

Ground Truth calculates a confidence score for each label. A *confidence score* is a number between 0 and 1 that indicates how confident Ground Truth is in the label. You can use the confidence score to compare labeled data objects to each other, and to identify the least or most confident labels.

You should not interpret the value of the confidence scores as an absolute value, or compare them across labeling jobs. For example, if all of the confidence scores are between 0.98 and 0.998, you should only compare the data objects with each other and not rely on the high confidence scores.

You should not compare the confidence scores of human-labeled data objects and auto-labeled data objects. The confidence scores for humans are calculated using the annotation consolidation function for the task, the confidence scores for automated labeling are calculated using a model that incorporates object features. The two models generally have different scales and average confidence.

For a bounding box labeling job, Ground Truth calculates a confidence score per box. You can compare confidence scores within one image or across images for the same labeling type (human or auto). You can't compare confidence scores across labeling jobs.

Output Metadata

The output from each job contains metadata about the label assigned to data objects. These elements are the same for all jobs with minor variations. The following are the metadata elements:

```
"confidence": 0.93,
"type": "groundtruth/image-classification",
"job-name": "identify-animal-species",
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256"
```

The elements have the following meaning:

- `confidence` — The confidence that Ground Truth has that the label is correct. For more information, see Confidence Score (p. 547).
- `type` — The type of classification job. For job types, see the descriptions of the individual job types.
- `job-name` — The name assigned to the job when it was created.
- `human-annotated` — Indicates whether the data object was labeled by a human or by automated data labeling. For more information, see Using Automated Data Labeling (p. 539).
- creation-date — The date and time that the label was created.

Classification Job Output

The following are sample output from an image classification job and a text classification job. It includes the label that Ground Truth assigned to the data object, the value for the label, and metadata that describes the labeling task.

In addition to the standard metadata elements, the metadata for a classification job includes the text value of the label's class. For more information, see Image Classification Algorithm (p. 108).

```
{
    "source-ref":"S3 bucket location",
    "species":"0",
    "species-metadata":
    {
        "class-name": "dog",
        "confidence": 0.93,
        "type": "groundtruth/image-classification",
```

```
        "job-name": "identify-animal-species",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256"
    }
}
```

```
{
    "source":"a bad day",
    "mood":"1",
    "mood-metadata":
    {
        "class-name": "sad",
        "confidence": 0.8,
        "type": "groundtruth/text-classification",
        "job-name": "label-mood",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256"
    }
}
```

Bounding Box Job Output

The following is sample output from a bounding box job. For this task, there are three bounding boxes returned. The value of the label contains information about the size of the image, and the location of the bounding boxes.

The `class_id` element is the index of the box's class in the list of available classes for the task. You can see the text of the class in the `class-map` metadata element.

In the metadata there is a separate confidence score for each bounding box. The metadata also includes the `class-map` element that maps the `class_id` to the text value of the class. For more information, see Object Detection Algorithm (p. 199).

```
{
    "source-ref": "S3 bucket location",
    "bounding-box":
    {
        "image_size": [{ "width": 500, "height": 400, "depth":3}],
        "annotations":
        [
            {"class_id": 0, "left": 111, "top": 134,
                    "width": 61, "height": 128},
            {"class_id": 5, "left": 161, "top": 250,
                    "width": 30, "height": 30},
            {"class_id": 5, "left": 20, "top": 20,
                    "width": 30, "height": 30}
        ]
    },
    "bounding-box-metadata":
    {
        "objects":
        [
            {"confidence": 0.8},
            {"confidence": 0.9},
            {"confidence": 0.9}
        ],
        "class-map":
        {
            "0": "dog",
            "5": "bone"
        },
        "type": "groundtruth/object_detection",
```

```
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256",
        "job-name": "identify-dogs-and-toys"
    }
 }
```

For an example notebook, see object_detection_augmented_manifest_training.ipynb

Semantic Segmentation Job Output

The following is the output from a semantic segmentation labeling job. The value of the label for this job is a reference to a PNG file in an S3 bucket.

In addition to the standard elements, the metadata for the label includes a color map that defines which color was used to label the image, the class name associated with the color, and the confidence score for each color. For more information, see Semantic Segmentation Algorithm (p. 234).

```
{
    "source-ref": "S3 bucket location",
    "city-streets-ref": "S3 bucket location",
    "city-streets-metadata": {
      "internal-color-map": {
        "5": {
            "class-name": "buildings",
            "confidence": 0.9
        },
        "2":  {
            "class-name": "road",
            "confidence": 0.9
        }
      },
      "type": "groundtruth/semantic-segmentation",
      "human-annotated": "yes",
      "creation-date": "2018-10-18T22:18:13.527256",
      "job-name": "label-city-streets",
      }
}
```

Confidence scores will be the same across all classes within an image. Confidence is scored on a per-image basis.

After you create an augmented manifest file, you can use it in a training job. For more information, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 308).

Creating Instruction Pages

Create custom instructions for labeling jobs to improve your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console or you can create your own. The instructions are shown to the worker on the page where they complete their labeling task.

There are two kinds of instructions:

- *Short instructions*—instructions that are shown on the same webpage where the worker completes their task. These instructions should provide an easy reference to show the worker the correct way to label an object.
- *Full instructions*—instructions that are shown on a dialog box that overlays the page where the worker completes their task. We recommend that you provide detailed instructions for completing the task with multiple examples showing edge cases and other difficult situations for labeling objects.

Create instructions in the console when you are creating your labeling job. Start with the existing instructions for the task and use the editor to modify them to suit your labeling job.

Short Instructions

Short instructions appear on the same webpage that workers use to label your data object. For example, the following is the editing page for a bounding box task. The short instructions panel is on the left.

Keep in mind that a worker will only spend seconds looking at the short instructions. Workers must be able to scan and understand your information quickly. In all cases it should take less time to understand the instructions than it takes to complete the task. Keep these points in mind:

- Your instructions should be clear and simple.
- Pictures are better than words. Create a simple illustration of your task that your workers can immediately understand.
- If you must use words, use short, concise examples.
- Your short instructions are more important than your full instructions.

The Amazon SageMaker Ground Truth console provides an editor so that you can create your short instructions. Replace the placeholder text and images with instructions for your task. Preview the worker's task page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

Full Instructions

You can provide additional instructions for your workers in a dialog box that overlays the page where workers label your data objects. Use full instructions to explain more complex tasks and to show workers the proper way to label edge cases or other difficult objects.

You can create full instructions using an editor in the Ground Truth console. As with quick instructions, keep the following in mind:

- Workers will want detailed instruction the first few times that the complete your task. Any information that they *must* have should be in the quick instructions.
- Pictures are more important than words.
- Text should be concise.
- Full instructions should supplement the short instructions. Don't repeat information that appears in the short instructions.

The Ground Truth console provides an editor so that you can create your full instructions. Replace the placeholder text and images with instructions for your task. Preview the full instruction page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

Add example images to your instructions

Images provide useful examples for your workers. To add a publicly accessible image to your instructions:

- Place the cursor where the image should go in the instructions editor.
- Click the image icon in the editor toolbar.
- Enter the URL of your image.

If your instruction image in Amazon S3 is not publicly accessible:

- As the image URL, enter: `{{ 'https://s3.amazonaws.com/`*`your-bucket-name`*`/`*`image-file-name`*`' | grant_read_access }}`.
- This renders the image URL with a short-lived, one-time access code appended so the worker's browser can display it. A broken image icon is displayed in the instructions editor, but previewing the tool displays the image in the rendered preview.

Managing Your Workforce

A *workforce* is the group of workers that you have selected to label your dataset. You can choose either the Amazon Mechanical Turk workforce, a vendor-managed workforce, or you can create your own private workforce to label your dataset. Whichever you choose, Amazon SageMaker Ground Truth takes care of sending tasks to workers.

When you use a private workforce, you also create *work teams*, a group of workers from your workforce that are assigned to specific labeling jobs. You can have multiple work teams and can assign one or more work teams to each labeling job.

Ground Truth uses Amazon Cognito to manage your workforce and work teams. For more information about the permissions required to manage this way, see Permissions Required to Use the Amazon SageMaker Ground Truth Console (p. 484).

Topics

Using the Amazon Mechanical Turk Workforce

The Amazon Mechanical Turk workforce provides the most workers for your labeling job.

You can use the console to choose the Amazon Mechanical Turk workforce for your labeling job, or you can provide the Amazon Resource Name (ARN) for the Amazon Mechanical Turk workforce when you use the CreateLabelingJob (p. 643) operation.

Any Amazon Mechanical Turk workforce billing is handled as part of your Amazon SageMaker Ground Truth billing. You do not need to create a separate Mechanical Turk account to use the Amazon Mechanical Turk workforce.

The ARN for the Amazon Mechanical Turk workforce is:

- `arn:aws:sagemaker:region:394669845002:workteam/public-crowd/default`

The Amazon Mechanical Turk workforce is a world-wide resource. Workers are available 24 hours a day, 7 days a week. You typically get the fastest turn-around for your labeling jobs when you use the Amazon Mechanical Turk workforce.

Adjust the number of workers that annotate each data object based on the complexity of the job and the quality that you need. Ground Truth uses annotation consolidation to improve the quality of the labels. More workers can make a difference in the quality of the labels for more complex labeling jobs, but might not make a difference for simpler jobs. For more information, see Annotation Consolidation (p. 537).

> **Note**
> Only use the Amazon Mechanical Turk workforce to label data that is public or has been stripped of any sensitive information. You should not use the Amazon Mechanical Turk workforce for a dataset that contains personally identifiable information, such as names or email addresses.

To choose the Amazon Mechanical Turk workforce when you are creating a labeling job using the console, do the following during the **Select workers and configure tool** step:

To use the Amazon Mechanical Turk workforce

1. Choose **Public** from **Worker types**.
2. Choose **The dataset does not contain adult content** if your dataset doesn't contain potentially offensive content. This enables workers to opt out if they don't want to work with it.
3. Acknowledge that your data will be viewed by the Amazon Mechanical Turk workforce and that all personally identifiable information (PII) has been removed.
4. Choose **Additional configuration** to set optional parameters.
5. Optional. Enable automated data labeling to have Ground Truth automatically label some of your dataset. For more information, see Using Automated Data Labeling (p. 539).
6. Optional. Set the number of workers that should see each object in your dataset. Using more workers can increase the quality of your labels but also increases the cost.

Your labeling job will now be sent to the Amazon Mechanical Turk workforce. You can use the console to continue configuring your labeling job.

Managing Vendor Workforces

You can use a vendor-managed workforce to label your data using Amazon SageMaker Ground Truth. Vendors have extensive experience in providing data labeling services for the purpose of machine learning.

Vendors make their services available via the AWS Marketplace. You can find details of the vendor's services on their detail page, such as the number of workers and the hours that they work. You can use these details to make estimates of how much the labeling job will cost and the amount of time that you can expect the job to take. Once you have chosen a vendor you subscribe to their services using the AWS Marketplace.

A subscription is an agreement between you and the vendor. The agreement spells out the details of the agreement, such as price, schedule, or refund policy. You work directly with the vendor if there are any issues with your labeling job.

You can subscribe to any number of vendors to meet your data annotation needs. When you create a labeling job you can specify that the job be routed to a specific vendor.

Before you send sensitive data to a vendor, check the vendor's security practices on their detail page and review the end user license agreement (EULA) that is part of your subscription agreement.

You must use the console to subscribe to a vendor workforce. Once you have a subscription, you can use the ListSubscribedWorkteams (p. 801) operation to list your subscribed vendors.

To subscribe to a vendor workforce

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Labeling workforces**, choose **Vendor**, and then choose **Find data labeling services**.
3. The console opens the AWS Marketplace with the data labeling services category selected. You see a list of the data labeling services available.
4. Choose a vendor. The AWS Marketplace shows detailed information about the data labeling service. Use this information to determine if the vendor meets your data labeling requirements.
5. If the vendor meets your requirements, choose **Continue to subscribe**.
6. Review the details of the subscription. If you agree to the terms, choose **Subscribe** to complete your subscription to the service.

Managing a Private Workforce

A *private workforce* is a group of workers that you choose. These can be employees of your company or a group of subject matter experts from your industry. For example, if the task is to label medical images, you could create a private workforce of people knowledgeable about the images in question.

You create *work teams* from your private workforce. If desired, you can assign each work team to a separate labeling job. A single worker can be in more than one work team.

Amazon SageMaker Ground Truth uses Amazon Cognito to define your private workforce and your work teams. Amazon Cognito is a service that you can use to create identities for your workers and authenticate these identities with identity providers. You can use providers such as the following:

- Amazon Cognito identity provider
- Social sign-in providers such as Facebook and Google
- OpenID Connect (OIDC) providers
- Security Assertion Markup Language (SAML) providers such as Active Directory

After your workers are set up, you use Amazon Cognito to manage them. For more information about Amazon Cognito, see What Is Amazon Cognito?

Your workers are organized into two groups. The *workforce* is the entire set of workers that are available to work on your labeling jobs. The workforce corresponds to an Amazon Cognito user pool. A *work team* is a group of workers within your workforce that you can assign jobs to. The work team corresponds to an Amazon Cognito user group. A worker can be in more than one work team.

> **Note**
> You can only use one Amazon Cognito user pool as your Ground Truth labeling workforce. If you plan on using an existing Amazon Cognito user pool for your Ground Truth workforce, be sure to import the user pool and create a work team before you create your first labeling job.

For more information, see Amazon Cognito User Pools.

Creating a private workforce

There are three ways that you can create a private workforce:

1. Import an existing Amazon Cognito user pool before you create your first labeling job.
2. Use the Amazon SageMaker console to create a new Amazon Cognito user pool before you create your first labeling job.
3. Create a new Amazon Cognito user pool while you are creating your first labeling job.

If you are using a pre-existing Amazon Cognito user pool for your private workforce, you must import the user pool into Ground Truth and create at least one work team, either by adding work team members in the Amazon SageMaker console or by importing an Amazon Cognito work group before you create your first labeling job for your private workforce.

After you create or import your private workforce you will see the **Private workforce summary** screen. On this screen you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce.

If you created your workforce using the Amazon SageMaker console, you can manage the members of your workforce in the Amazon SageMaker console or in the Amazon Cognito console. For example, you can use the Amazon SageMaker console to add, delete, and disable workers in the pool. If you imported the workforce from an existing Amazon Cognito user pool, you must use the Amazon Cognito console to manage the workforce.

Creating a workforce when creating a labeling job

If you have not created a private workforce when you create your labeling job, you are prompted to create one. Creating the workforce also creates a default work team containing all of the members of the workforce. If you have already created a private workforce, you instead enter the work team that should handle the labeling job.

You provide the following information to create the workforce and work team.

- Up to 100 email addresses of the workforce members. Email addresses are case-sensitive. Your workers must log in using the same case as the address was initially entered. You can add additional workforce members after the job is created.
- The name of your organization. This is used to customize the email sent to the workers.
- A contact email address for workers to report issues related to the task.

When you create the labeling job an email is sent to each worker inviting them to join the workforce. Once the workforce is created, you can add, delete, and disable workers using the Amazon SageMaker console or the Amazon Cognito console.

Creating a private workforce using the console

To create a private workforce using the console

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Labeling workforces** from the left menu.
3. Select the **Private** tab.
4. Click the **Create private team** button. This process will create a private workforce and a work team.
5. Choose to **Invite new workers by email** or **Import workers from existing Amazon Cognito user groups**.
6. **To invite new workers by email**.

   - Paste or type a list of email addresses, separated by commas, into the email addresses box. You may have up to 50 email addresses in this list.
   - Enter an organization name and contact email.
   - Optionally select an SNS topic to which to subscribe the team so workers are notified by email when new labeling jobs become available.
   - Click the **Create private team** button.

7. **To Import workers from existing Amazon Cognito user groups**.

   - Choose a user pool you have created. User pools require a domain and an existing user group. If you get an error that the domain is missing, set it in the **Domain name** options within the **App integration** section of the Amazon Cognito control console for your group.
   - Select an app client. We recommend using a SageMaker generated client.
   - Select a user group from your pool to import its members.
   - Optionally select an SNS topic to which to subscribe the team so workers are notified by email when new labeling jobs become available.
   - Click the **Create private team** button.

After you have created the workforce, Ground Truth automatically creates a work team called **Everyone-in-private-workforce**. You can use this work team to assign a labeling job to your entire workforce.

Creating a Work Team

Use a work team to assign members of your private workforce to a labeling job. When you create your workforce using the Amazon SageMaker Ground Truth console, there is a work team called **Everyone-in-private-workforce** that you can use if you want to assign your entire workforce to a labeling job. Because an imported Amazon Cognito user pool may contain members that you don't want to include in your work teams, a similar work team is not created for Amazon Cognito user pools.

You have two choices to create a new work team. You can create a user group by using the Amazon Cognito console and then create a work team by importing the user group. You can import more than one user group into each work team. You manage the members of the work team by updating the user group in the Amazon Cognito console.

You can also create a work team in the Ground Truth console and add members from your workforce to the team.

To create a work team using the Ground Truth console

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Labeling workforces** from the left menu.
3. Under **Private teams**, choose **Create private team**.

Amazon SageMaker Developer Guide
Create and manage Amazon
SNS topics for your work teams

4. Under **Team details**, give the team a name. The name must be unique in your account in an AWS Region.

5. Under **Create a team using Amazon Cognito user groups**, choose the method to use to create the group.

6. If you chose **Create a team by adding workers to a new Amazon Cognito user group**, select the workers to add to the team.

   If you chose **Create a team by importing existing Amazon Cognito user groups**, choose the user groups that are part of the new team.

7. If you select an **SNS topic**, all workers added to the team are subscribed to the Amazon SNS topic and notified when new work items are available to the team. Select from a list of your existing Ground Truth related Amazon SNS topics or select **Create new topic** to open a topic-creation dialog.

   Workers in a workteam subscribed to a topic receive notifications when a new job for that team becomes available and when one is about to expire.

   Read for more information on creating and managing the Amazon SNS topic.

After you have created a work team, you can choose its name in the list of work teams to see more information about the team and change or set the Amazon SNS topic to which its members are subscribed. Any members of the team who were added to the team prior to the team being subscribed to a topic will need to be subscribed to that topic manually. Teams made from more than one imported user group must be edited in the Amazon Cognito console; otherwise you can add and remove workers using the team detail page.

Create and manage Amazon SNS topics for your work teams

Create the Amazon SNS topic

These instructions are needed when you:

- Create a topic to which you will subscribe an existing work team.
- Create a topic prior to creating a work team.
- Create or modify the work team with an API call and need to specify a topic ARN.

If you create a work team using the console, that process provides an option to create a new topic for the team, and if used, handles these steps for you.

The creation of Amazon SNS topics for work team notification is similar to the steps in the Amazon SNS Getting Started document, with one significant addition: adding an access policy so the Sagemaker service can publish messages to the topic on your behalf.

Add the policy while creating the topic

1. In the **Access policy** panel, select the advanced method.

2. In the **JSON editor**, find the `Resource` property, which displays what the ARN of the topic will be when it is created. If the value ends in "undefined," change "undefined" to the name of the topic.

3. Copy the `Resource` value.

4. Before the final closing bracket (`]`), add the following policy.

```
{
```

```
        "Sid": "AWSSagemaker_AccessPolicy",
        "Effect": "Allow",
        "Principal": {
          "Service": "sagemaker.amazonaws.com"
        },
        "Action": "sns:Publish",
        "Resource": "resource ARN you copied in the previous step"
      }
```

5. Create the topic.

For more details on creating topics, see the Amazon SNS "Creating a Topic" tutorial.

Manage worker subscriptions

Workers are auto-subscribed to your topic only when a Amazon Cognito user group is created or imported during the creation of a work team that is subscribed to the topic at creation.

If a work team is subscribed to a topic after creation, the individual members at the time of creation are not automatically subscribed. See "Subscribing an Endpoint to an Amazon SNS Topic" for information on subscribing the workers' email addresses to the topic.

Creating Custom Labeling Workflows

This document guides you through the process of setting up a workflow with a custom labeling template. For more information about starting a labeling job, see Getting started (p. 533). In that section, when you choose the **Task type**, select **Custom labeling task**, and then follow this section's instructions to configure it.

Next

Step 1: Setting up your workforce

In this step you use the console to establish which worker type to use and make the necessary sub-selections for the worker type. It assumes you have already completed the steps up to this point in the Getting started (p. 533) section and have chosen the **Custom labeling task** as the **Task type**.

To configure your workforce.

1. First choose an option from the **Worker types**. There are three types currently available:

    *   **Public** uses an on-demand workforce of independent contractors, powered by Amazon Mechanical Turk. They are paid on a per-task basis.
    *   **Private** uses your employees or contractors for handling data that needs to stay within your organization.
    *   **Vendor** uses third party vendors that specialize in providing data labeling services, available via the AWS Marketplace.

2. If you choose the **Public** option, you are asked to set the **number of workers per dataset object**. Having more than one worker perform the same task on the same object can help increase the accuracy of your results. The default is three. You can raise or lower that depending on the accuracy you need.

    You are also asked to set a **price per task** by using a drop-down menu. The menu recommends price points based on how long it will take to complete the task.

The recommended method to determine this is to first run a short test of your task with a **private** workforce. The test provides a realistic estimate of how long the task takes to complete. You can then select the range your estimate falls within on the **Price per task** menu. If your average time is more than 5 minutes, consider breaking your task into smaller units.

Next

Step 2: Creating your custom labeling task template

Topics

Starting with a base template

To get you started, the **Task type** starts with a drop-down menu listing a number of our more common task types, plus a custom type. Choose one and the code editor area will be filled with a sample template for that task type. If you prefer not to start with a sample, choose **Custom HTML** for a minimal template skeleton.

If you've already created a template, upload the file directly using the **Upload file** button in the upper right of the task setup area or paste your template code into the editor area.

Developing templates locally

While you need to be in the console to test how your template will process incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding this code to the top of your HTML file.

Example

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this if you want to develop your template's look and feel in your preferred editor rather than in the console.

Remember, though, this will not parse your variables. You may want to replace them with sample content while developing locally.

Using External Assets

Amazon SageMaker Ground Truth custom templates allow external scripts and style sheets to be embedded.

Example

```
<script src="https://www.example.com/my-enhancment-script.js"></script>
<link rel="stylesheet" type="text/css" href="https://www.example.com/my-enhancement-
styles.css">
```

If you encounter errors, ensure that your originating server is sending the correct MIME type and encoding headers with the assets.

For example, the MIME and encoding types for remote scripts: `application/javascript;CHARSET=UTF-8`.

The MIME and encoding type for remote stylesheets: `text/css;CHARSET=UTF-8`.

Track your variables

In the process of building the sample below, there will be a step that adds variables to it to represent the pieces of data that may change from task to task, worker to worker. If you're starting with one of the sample templates, you will need to make sure you're aware of the variables it already uses. When you create your pre-annotation AWS Lambda script, its output will need to contain values for any of those variables you choose to keep.

The values you use for the variables can come from your manifest file. All the key-value pairs in your data object are provided to your pre-annotation Lambda. If it's a simple pass-through script, matching keys for values in your data object to variable names in your template is the easiest way to pass those values through to the tasks forms your workers see.

A simple sample

All tasks begin and end with the `<crowd-form>` `</crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between them.

For a simple tweet-analysis task, use the `<crowd-classifier>` element. It requires the following attributes:

- *name* - the variable name to use for the result in the form output.
- *categories* - a JSON formatted array of the possible answers.
- *header* - a title for the annotation tool

As children of the `<crowd-classifier>` element, you must have three regions.

- *<classification-target>* - the text the worker will classify based on the options specified in the `categories` attribute above.
- *<full-instructions>* - instructions that are available from the "View full instructions" link in the tool. This can be left blank, but it is recommended that you give good instructions to get better results.
- *<short-instructions>* - a more brief description of the task that appears in the tool's sidebar. This can be left blank, but it is recommended that you give good instructions to get better results.

A simple version of this tool would look like this.

Example of using `crowd-classifier`

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
```

```
    categories="['positive','negative','neutral', 'unclear']"
    header="Which term best describes this tweet?"
  >

    <classification-target>
      My favorite football team won today!
      Bring on the division finals!
    </classification-target>

    <full-instructions header="Sentiment Analysis Instructions">
      Try to determine the sentiment the author
      of the tweet is trying to express.
      If none seem to match, choose "cannot determine."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment
      of the tweet.
    </short-instructions>

  </crowd-classifier>
</crowd-form>
```

You can copy and paste the code into the editor in the Ground Truth labeling job creation workflow to preview the tool, or try out a demo of this code on CodePen.
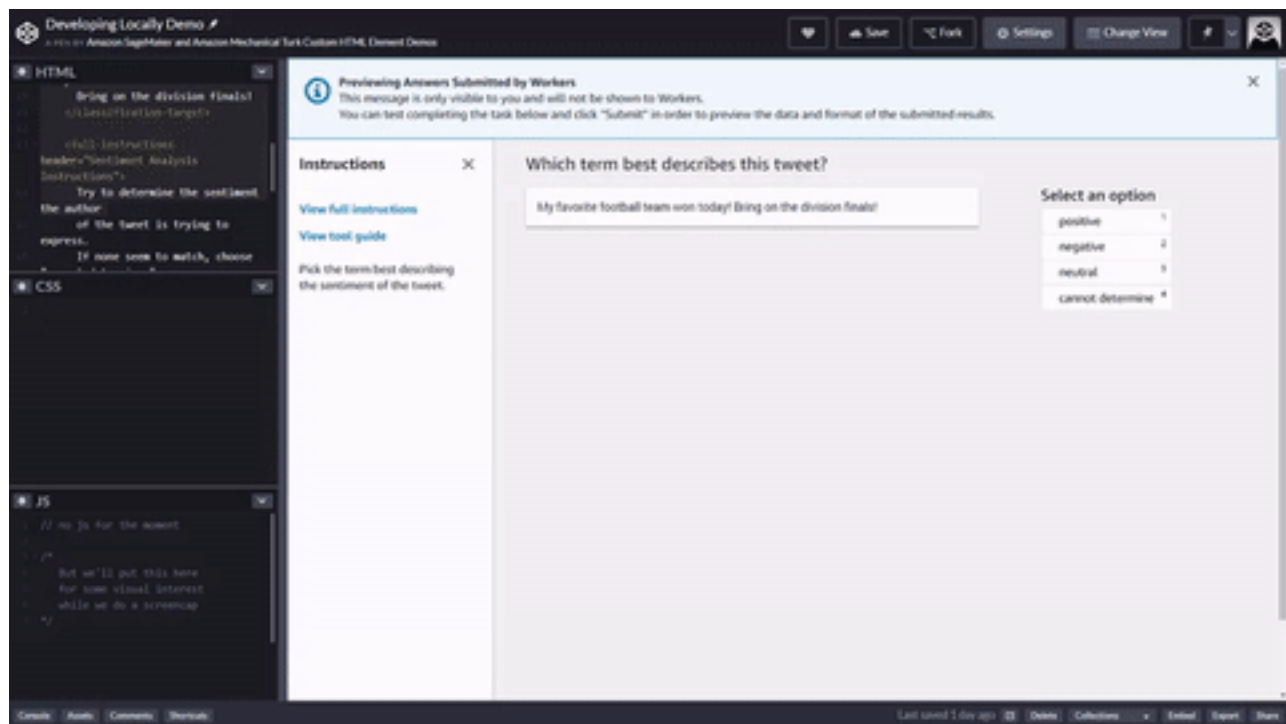
Adding automation with Liquid

Our custom template system uses Liquid for automation. It is an open source inline markup language. For more information and documentation, visit the Liquid homepage.

The most common use of Liquid will be to parse the data coming from your **pre-annotation Lambda** and pull out the relevant variables to create the task. In Liquid, the text between single curly braces and percent symbols is an instruction or "tag" that creates control flow. Text between double curly braces is a variable or "object" which outputs its value.

The `taskInput` object returned by your Pre-annotation Lambda (p. 574) will be available as the `task.input` object in your templates.

The properties in your manifest's data objects are passed into your Pre-annotation Lambda (p. 574) as the `event.dataObject`. A simple pass-through script simply returns that object as the `taskInput` object. You would represent values from your manifest as variables as follows.

Example Manifest data object

```
{
  "source": "This is a sample text for classification",
  "labels": [ "angry" , "sad" , "happy" , "inconclusive" ],
  "header": "What emotion is the speaker feeling?"
}
```

Example Sample HTML using variables

```
<crowd-classifier
  name='tweetFeeling'
  categories='{{ task.input.labels | to_json }}'
  header='{{ task.input.header }}' >
<classification-target>
  {{ task.input.source }}
</classification-target>
```

Note the addition of " `| to_json`" to the `labels` property above. That's a filter to turn the array into a JSON representation of the array. Variable filters are explained next.

Variable filters

In addition to the standard Liquid filters and actions, Ground Truth offers a few additional filters. Filters are applied by placing a pipe (|) character after the variable name, then specifying a filter name. Filters can be chained in the form of:

Example

```
{{ <content> | <filter> | <filter> }}
```

Autoescape and explicit escape

By default, inputs will be HTML escaped to prevent confusion between your variable text and HTML. You can explicitly add the `escape` filter to make it more obvious to someone reading the source of your template that the escaping is being done.

escape_once

`escape_once` ensures that if you've already escaped your code, it doesn't get re-escaped on top of that. For example, so that & doesn't become &amp;.

skip_autoescape

`skip_autoescape` is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

> **Use `skip_autoescape` sparingly**
> The best practice in templates is to avoid passing in functional code or markup with `skip_autoescape` unless you are absolutely sure you have strict control over what's being

passed. If you're passing user input, you could be opening your workers up to a Cross Site
Scripting attack.

to_json

`to_json` will encode what you feed it to JSON (JavaScript Object Notation). If you feed it an object, it
will serialize it.

grant_read_access

`grant_read_access` takes an S3 URI and encodes it into an HTTPS URL with a short-lived access token
for that resource. This makes it possible to display to workers photo, audio, or video objects stored in S3
buckets that are not otherwise publicly accessible.

Example of the filters

Input

```
auto-escape: {{ "Have you read 'James & the Giant Peach'?" }}
explicit escape: {{ "Have you read 'James & the Giant Peach'?" | escape }}
explicit escape_once: {{ "Have you read 'James &amp; the Giant Peach'?" | escape_once }}
skip_autoescape: {{ "Have you read 'James & the Giant Peach'?" | skip_autoescape }}
to_json: {{ jsObject | to_json }}
grant_read_access: {{ "s3://mybucket/myphoto.png" | grant_read_access }}
```

Example

Output

```
auto-escape: Have you read &#39;James &amp; the Giant Peach&#39;?
explicit escape: Have you read &#39;James &amp; the Giant Peach&#39;?
explicit escape_once: Have you read &#39;James &amp; the Giant Peach&#39;?
skip_autoescape: Have you read 'James & the Giant Peach'?
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
grant_read_access: https://s3.amazonaws.com/mybucket/myphoto.png?<access token and other
 params>
```

Example of an automated classification template.

To automate the simple text classification sample, replace the tweet text with a variable.

The text classification template is below with automation added. The changes/additions are highlighted
in bold.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="['positive', 'negative', 'neutral', 'cannot determine']"
    header="Which term best describes this tweet?"
  >
    <classification-target>
        {{ task.input.source }}
    </classification-target>

    <full-instructions header="Analyzing a sentiment">
      Try to determine the feeling the author
      of the tweet is trying to express.
      If none seem to match, choose "other."
    </full-instructions>
```

```
      <short-instructions>
        Pick the term best describing the sentiment
        of the tweet.
      </short-instructions>

  </crowd-classifier>
</crowd-form>
```

The tweet text that was in the prior sample is now replaced with an object. The `entry.taskInput` object uses `source` (or another name you specify in your pre-annotation Lambda) as the property name for the text and it is inserted directly in the HTML by virtue of being between double curly braces.

End-to-end demos

You can view the following end-to-end demos which include sample Lambdas:

Next

Demo Template: Annotation of Images with `crowd-bounding-box`

When you chose to use a custom template, you reach the **Custom labeling task panel**. There you can choose from multiple base templates. The templates represent some of the most common tasks and provide a sample to work from as you create your customized labeling task's template.

This demonstration works with the **BoundingBox** template. The demonstration also works with the AWS Lambda functions needed for processing your data before and after the task.

Topics

Starter Bounding Box custom template

This is the starter bounding box template that is provided.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-bounding-box
    name="annotatedResult"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="{{ task.input.header }}"
    labels="{{ task.input.labels | to_json | escape }}"
```

```
  >

    <!-- The <full-instructions> tag is where you will define the full instructions of your
 task. -->
    <full-instructions header="Bounding Box Instructions" >
      <p>Use the bounding box tool to draw boxes around the requested target of interest:</
p>
      <ol>
        <li>Draw a rectangle using your mouse over each instance of the target.</li>
        <li>Make sure the box does not cut into the target, leave a 2 - 3 pixel margin</li>
        <li>
          When targets are overlapping, draw a box around each object,
          include all contiguous parts of the target in the box.
          Do not include parts that are completely overlapped by another object.
        </li>
        <li>
          Do not include parts of the target that cannot be seen,
          even though you think you can interpolate the whole shape of the target.
        </li>
        <li>Avoid shadows, they're not considered as a part of the target.</li>
        <li>If the target goes off the screen, label up to the edge of the image.</li>
      </ol>
    </full-instructions>

    <!-- The <short-instructions> tag allows you to specify instructions that are displayed
 in the left hand side of the task interface.
    It is a best practice to provide good and bad examples in this section for quick
 reference. -->
    <short-instructions>
      Use the bounding box tool to draw boxes around the requested target of interest.
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>
```

The custom templates use the [Liquid template language](#), and each of the items between double curly braces is a variable. The pre-annotation AWS Lambda function should provide an object named `taskInput` and that object's properties can be accessed as `{{ task.input.<property name> }}` in your template.

Your own Bounding Box custom template

As an example, assume you have a large collection of animal photos in which you know the kind of animal in an image from a prior image-classification job. Now you want to have a bounding box drawn around it.

In the starter sample, there are three variables: `taskObject`, `header`, and `labels`.

Each of these would be represented in different parts of the bounding box.

- `taskObject` is an HTTP(S) URL or S3 URI for the photo to be annotated. The added `| grant_read_access` is a filter that will convert an S3 URI to an HTTPS URL with short-lived access to that resource. If you're using an HTTP(S) URL, it's not needed.
- `header` is the text above the photo to be labeled, something like "Draw a box around the bird in the photo."
- `labels` is an array, represented as `['item1', 'item2', ...]`. These are labels that can be assigned by the worker to the different boxes they draw. You can have one or many.

Each of the variable names come from the JSON object in the response from your pre-annotation Lambda, The names above are merely suggested, Use whatever variable names make sense to you and will promote code readability among your team.

Only use variables when necessary
If a field will not change, you can remove that variable from the template and replace it with that text, otherwise you have to repeat that text as a value in each object in your manifest or code it into your pre-annotation Lambda function.

Example : Final Customized Bounding Box Template

To keep things simple, this template will have one variable, one label, and very basic instructions. Assuming your manifest has an "animal" property in each data object, that value can be re-used in two parts of the template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-bounding-box
    name="annotatedResult"
    labels="[ '{{ task.input.animal }}' ]"
    src="{{ task.input.source-ref | grant_read_access }}"
    header="Draw a box around the {{ task.input.animal }}."
  >
    <full-instructions header="Bounding Box Instructions" >
      <p>Draw a bounding box around the {{ task.input.animal }} in the image. If
      there is more than one {{ task.input.animal }} per image, draw a bounding
      box around the largest one.</p>
      <p>The box should be tight around the {{ task.input.animal }} with
      no more than a couple of pixels of buffer around the
      edges.</p>
      <p>If the image does not contain a {{ task.input.animal }}, check the <strong>
      Nothing to label</strong> box.
    </full-instructions>
    <short-instructions>
      <p>Draw a bounding box around the {{ task.input.animal }} in each image. If
      there is more than one {{ task.input.animal }} per image, draw a bounding
      box around the largest one.</p>
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>
```

Note the re-use of `{{ task.input.animal }}` throughout the template. If your manifest had all of the animal names beginning with a capital letter, you could use `{{ task.input.animal | downcase }}`, incorporating one of Liquid's built-in filters in sentences where it needed to be presented lowercase.

Your manifest file

Your manifest file should provide the variable values you're using in your template. You can do some transformation of your manifest data in your pre-annotation Lambda, but if you don't need to, you maintain a lower risk of errors and your Lambda will run faster. Here's a sample manifest file for the template.

```
{"source-ref": "<S3 image URI>", "animal": "horse"}
{"source-ref": "<S3 image URI>", "animal" : "bird"}
{"source-ref": "<S3 image URI>", "animal" : "dog"}
{"source-ref": "<S3 image URI>", "animal" : "cat"}
```

Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda function that can be called to process your manifest entries and pass them to the template engine.

Naming your Lambda function
The best practice in naming your function is to use one of the following four strings as part of the function name: `SageMaker`, `Sagemaker`, `sagemaker`, or `LabelingFunction`. This applies to both your pre-annotation and post-annotation functions.

When you're using the console, if you have AWS Lambda functions that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.

In this very basic example, you're just passing through the information from the manifest without doing any additional processing on it. This sample pre-annotation function is written for Python 3.7.

```
import json

def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }
```

The JSON object from your manifest will be provided as a child of the `event` object. The properties inside the `taskInput` object will be available as variables to your template, so simply setting the value of `taskInput` to `event['dataObject']` will pass all the values from your manifest object to your template without having to copy them individually. If you wish to send more values to the template, you can add them to the `taskInput` object.

Your post-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer consolidation and scoring as it comes in, you can apply the scoring and/or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

Provide permissions to your post-annotation Lambda
The annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple pass through function, you need to assign `S3ReadOnly` access to your Lambda so it can read the annotation files.
In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is in Python 2.7.

```
import json
import boto3
from urlparse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []

    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
    textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:])
    filecont = textFile['Body'].read()
    annotations = json.loads(filecont);

    for dataset in annotations:
        for annotation in dataset['annotations']:
            new_annotation = json.loads(annotation['annotationData']['content'])
            label = {
                'datasetObjectId': dataset['datasetObjectId'],
```

```
                'consolidatedAnnotation' : {
                'content': {
                    event['labelAttributeName']: {
                        'workerId': annotation['workerId'],
                        'boxesInfo': new_annotation,
                        'imageSource': dataset['dataObject']
                        }
                    }
                }
            }
            consolidated_labels.append(label)

    return consolidated_labels
```

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the `payload` object the Lambda should iterate through. What you send back will be an object meeting the .

The output of your labeling job

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named `manifests`.

For a bounding box task, the output you find in the output manifest will look a bit like the demo below. The example has been cleaned up for printing. The actual output will be a single line per record.

Example : JSON in your output manifest

```
{
  "source-ref":"<URL>",
  "<label attribute name>":
    {
        "workerId":"<URL>",
        "imageSource":"<image URL>",
        "boxesInfo":"{\"annotatedResult\":{\"boundingBoxes\":[{\"height\":878, \"label
\":\"bird\", \"left\":208, \"top\":6, \"width\":809}], \"inputImageProperties\":{\"height
\":924, \"width\":1280}}}"},
  "<label attribute name>-metadata":
    {
      "type":"groundTruth/custom",
      "job_name":"<Labeling job name>",
      "human-annotated":"yes"
    },
  "animal" : "bird"
}
```

Note how the additional `animal` attribute from your original manifest is passed to the output manifest on the same level as the `source-ref` and labeling data. Any properties from your input manifest, whether they were used in your template or not, will be passed to the output manifest.

This should help you create your own custom template.

Demo Template: Labeling Intents with `crowd-classifier`

If you choose a custom template, you'll reach the **Custom labeling task panel**. There you can select from multiple starter templates that represent some of the more common tasks. The templates provide a starting point to work from in building your customized labeling task's template.

In this demonstration, you work with the **Intent Detection** template, which uses the `crowd-classifier (p. 582)` element, and the AWS Lambda functions needed for processing your data before and after the task.

Topics

Starter Intent Detection custom template

This is the intent detection template that is provided as a starting point.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="intent"
    categories="{{ task.input.labels | to_json | escape }}"
    header="Pick the most relevant intention expressed by the below text"
  >
    <classification-target>
      {{ task.input.utterance }}
    </classification-target>

    <full-instructions header="Intent Detection Instructions">
        <p>Select the most relevant intention expressed by the text.</p>
        <div>
            <p><strong>Example: </strong>I would like to return a pair of shoes</p>
            <p><strong>Intent: </strong>Return</p>
        </div>
    </full-instructions>

    <short-instructions>
      Pick the most relevant intention expressed by the text
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

The custom templates use the Liquid template language, and each of the items between double curly braces is a variable. The pre-annotation AWS Lambda function should provide an object named `taskInput` and that object's properties can be accessed as `{{ task.input.<property name> }}` in your template.

Your Intent Detection custom template

In the starter template, there are two variables: the `task.input.labels` property in the `crowd-classifier` element opening tag and the `task.input.utterance` in the `classification-target` region's content.

Unless you need to offer different sets of labels with different utterances, avoiding a variable and just using text will save processing time and creates less possibility of error. The template used in this demonstration will remove that variable, but variables and filters like `to_json` are explained in more detail in the `crowd-bounding-box demonstration` article.

Styling Your Elements

Two parts of these custom elements that sometimes get overlooked are the `<full-instructions>` and `<short-instructions>` regions. Good instructions generate good results.

In the elements that include these regions, the `<short-instructions>` appear automatically in the "Instructions" pane on the left of the worker's screen. The `<full-instructions>` are linked from the "View full instructions" link near the top of that pane. Clicking the link opens a modal pane with more detailed instructions.

You can not only use HTML, CSS, and JavaScript in these sections, you are encouraged to if you believe you can provide a strong set of instructions and examples that will help workers complete your tasks with better speed and accuracy.

Example Try out a sample with JSFiddle
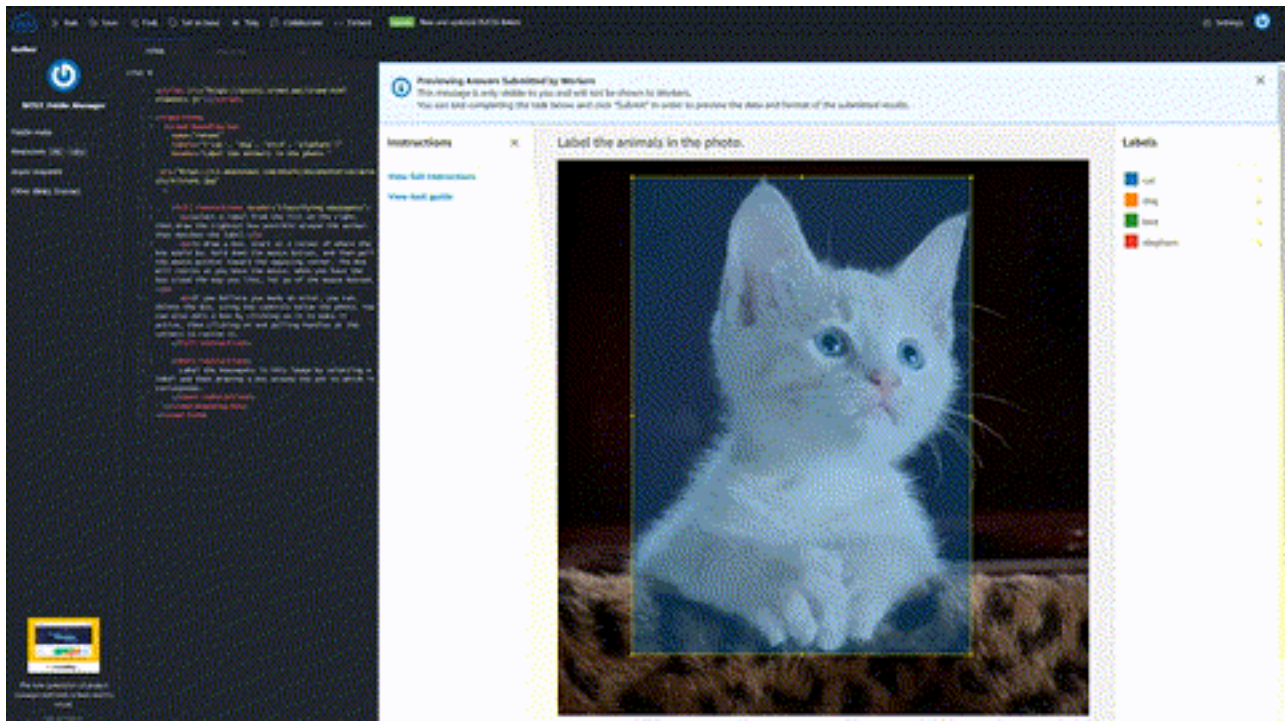
Try out an example `<crowd-classifier>` task. The example is rendered by JSFiddle, therefore all the template variables are replaced with hard-coded values. Click the "View full instructions" link to see a set of examples with extended CSS styling. You can fork the project to experiment with your own changes to the CSS, adding sample images, or adding extended JavaScript functionality.

Example : Final Customized Intent Detection Template

This uses the example `<crowd-classifier>` task, but with a variable for the `<classification-target>`. If you are trying to keep a consistent CSS design among a series of different labeling jobs, you can include an external stylesheet using a `<link rel...>` element the same way you'd do in any other HTML document.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="intent"
    categories="['buy', 'eat', 'watch', 'browse', 'leave']"
```

```
      header="Pick the most relevant intent expressed by the text below"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Emotion Classification Instructions">
      <p>In the statements and questions provided in this exercise, what category of action
 is the speaker interested in doing?</p>
          <table>
            <tr>
              <th>Example Utterance</th>
              <th>Good Choice</th>
            </tr>
            <tr>
              <td>When is the Seahawks game on?</td>
              <td>
                eat<br>
                <greenbg>watch</greenbg>
                <botchoice>browse</botchoice>
              </td>
            </tr>
            <tr>
              <th>Example Utterance</th>
              <th>Bad Choice</th>
            </tr>
            <tr>
              <td>When is the Seahawks game on?</td>
              <td>
                buy<br>
                <greenbg>eat</greenbg>
                <botchoice>watch</botchoice>
              </td>
            </tr>
          </table>
    </full-instructions>

    <short-instructions>
      What is the speaker expressing they would like to do next?
    </short-instructions>
  </crowd-classifier>
</crowd-form>
<style>
  greenbg {
    background: #feee23;
    display: block;
  }

  table {
    *border-collapse: collapse; /* IE7 and lower */
    border-spacing: 0;
  }

  th, tfoot, .fakehead {
    background-color: #8888ee;
    color: #f3f3f3;
    font-weight: 700;
  }

  th, td, tfoot {
      border: 1px solid blue;
  }

  th:first-child {
    border-radius: 6px 0 0 0;
  }
```

```
  th:last-child {
    border-radius: 0 6px 0 0;
  }

  th:only-child{
    border-radius: 6px 6px 0 0;
  }

  tfoot:first-child {
    border-radius: 0 0 6px 0;
  }

  tfoot:last-child {
    border-radius: 0 0 0 6px;
  }

  tfoot:only-child{
    border-radius: 6px 6px;
  }

  td {
    padding-left: 15px ;
    padding-right: 15px ;
  }

  botchoice {
    display: block;
    height: 17px;
    width: 490px;
    overflow: hidden;
    position: relative;
    background: #fff;
    padding-bottom: 20px;
  }

  botchoice:after {
    position: absolute;
    bottom: 0;
    left: 0;
    height: 100%;
    width: 100%;
    content: "";
    background: linear-gradient(to top,
        rgba(255,255,255, 1) 55%,
        rgba(255,255,255, 0) 100%
    );
    pointer-events: none; /* so the text is still selectable */
  }
</style>
```

Example : Your manifest file

If you are preparing your manifest file manually for a text-classification task like this, have your data formatted in the following manner.

```
{"source": "Roses are red"}
{"source": "Violets are Blue"}
{"source": "Ground Truth is the best"}
{"source": "And so are you"}
```

This differs from the manifest file used for the "Demo Template: Annotation of Images with `crowd-bounding-box` (p. 563)" demonstration in that `source-ref` was used as the property name instead

of `source`. The use of `source-ref` designates S3 URIs for images or other files that must be converted to HTTP. Otherwise, `source` should be used like it is with the text strings above.

Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda that can be called to process your manifest entries and pass them to the template engine.

This Lambda function is required to have one of the following four strings as part of the function name: `SageMaker`, `Sagemaker`, `sagemaker`, or `LabelingFunction`.

This applies to both your pre-annotation and post-annotation Lambdas.

When you're using the console, if you have Lambdas that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.

In this very basic sample, where you have only one variable, it's primarily a pass-through function. Here's a sample pre-labeling Lambda using Python 3.7.

```python
import json

def lambda_handler(event, context):
    return {
        "taskInput":  event['dataObject']
    }
```

The `dataObject` property of the `event` contains the properties from a data object in your manifest.

In this demonstration, which is a simple pass through, you just pass that straight through as the `taskInput` value. If you add properties with those values to the `event['dataObject']` object, they will be available to your HTML template as Liquid variables with the format `{{ task.input.`*`<property name>`*` }}`.

Your post-annotation Lambda function

As part of the job set up, provide the ARN of an Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer-consolidation and scoring as data comes in, you can apply the scoring or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

Set permissions for your post-annotation Lambda function
The annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple pass through function, you need to assign `S3ReadOnly` access to your Lambda so it can read the annotation files.
In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is for Python 3.7.

```python
import json
import boto3
from urllib.parse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []
```

```
        parsed_url = urlparse(event['payload']['s3Uri']);
        s3 = boto3.client('s3')
        textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:])
        filecont = textFile['Body'].read()
        annotations = json.loads(filecont);

        for dataset in annotations:
            for annotation in dataset['annotations']:
                new_annotation = json.loads(annotation['annotationData']['content'])
                label = {
                    'datasetObjectId': dataset['datasetObjectId'],
                    'consolidatedAnnotation' : {
                    'content': {
                        event['labelAttributeName']: {
                            'workerId': annotation['workerId'],
                            'result': new_annotation,
                            'labeledContent': dataset['dataObject']
                            }
                        }
                    }
                }
                consolidated_labels.append(label)

    return consolidated_labels
```

Your labeling job output

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the `payload` object the Lambda should iterate through.

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named `manifests`.

For an intent detection task, the output in the output manifest will look a bit like the demo below. The example has been cleaned up and spaced out to be easier for humans to read. The actual output will be more compressed for machine reading.

Example : JSON in your output manifest

```
[
  {
    "datasetObjectId":"<Number representing item's place in the manifest>",
     "consolidatedAnnotation":
     {
       "content":
       {
         "<name of labeling job>":
         {
           "workerId":"private.us-east-1.XXXXXXXXXXXXXXXXXXXXX",
           "result":
           {
             "intent":
             {
                 "label":"<label chosen by worker>"
             }
           },
           "labeledContent":
           {
             "content":"<text content that was labeled>"
           }
         }
       }
     }
```

```
          }
        },
    "datasetObjectId":"<Number representing item's place in the manifest>",
        "consolidatedAnnotation":
        {
          "content":
          {
            "<name of labeling job>":
            {
              "workerId":"private.us-east-1.6UDLPKQZHYWJQSCA4MBJBB7FWE",
              "result":
              {
                "intent":
                {
                    "label": "<label chosen by worker>"
                }
              },
              "labeledContent":
              {
                "content": "<text content that was labeled>"
              }
            }
          }
        }
      },
      ...
      ...
      ...
]
```

This should help you create and use your own custom template.

Step 3: Processing with AWS Lambda

In this step, you set which AWS Lambda functions to trigger on each dataset object prior to sending it to workers and which function will be used to process the results once the task is submitted. These functions are required.

You will first need to visit the AWS Lambda console or use AWS Lambda's APIs to create your functions. The AmazonSageMakerFullAccess policy is restricted to invoking AWS Lambda functions with one of the following four strings as part of the function name: `SageMaker`, `Sagemaker`, `sagemaker`, or `LabelingFunction`. This applies to both your pre-annotation and post-annotation Lambdas. If you choose to use names without those strings, you must explicitly provide `lambda:InvokeFunction` permission to the IAM role used for creating the labeling job.

Select your lambdas from the **Lambda functions** section that comes after the code editor for your custom HTML in the Ground Truth console.

If you need an example, there is an end-to-end demo, including Python code for the Lambdas, in the "Demo Template: Annotation of Images with `crowd-bounding-box` (p. 563)" document.

Pre-annotation Lambda

Before a labeling task is sent to the worker, your AWS Lambda function will be sent a JSON formatted request to provide details.

Example of a Pre-annotation request

```
{
    "version": "2018-10-16",
```

```
    "labelingJobArn": <labelingJobArn>
    "dataObject" : {
        "source-ref": "s3://mybucket/myimage.png"
    }
}
```

The `dataObject` will contain the JSON formatted properties from your manifest's data object. For a very basic image annotation job, it might just be a `source-ref` property specifying the image to be annotated. The JSON line objects in your manifest can be up to 100 kilobytes in size and contain a variety of data.

In return, Ground Truth will require a response formatted like this:

Example of expected return data

```
{
    "taskInput": <json object>,
    "isHumanAnnotationRequired": <boolean> # Optional
}
```

That `<json object>` may be a bit deceiving. It needs to contain *all* the data your custom form will need. If you're doing a bounding box task where the instructions stay the same all the time, it may just be the HTTP(S) or S3 resource for your image file. If it's a sentiment analysis task and different objects may have different choices, it would be the object reference as a string and the choices as an array of strings.

Implications of `isHumanAnnotationRequired`
This value is optional because it will default to `true`. The primary use case for explicitly setting it is when you want to exclude this data object from being labeled by human workers.

If you have a mix of objects in your manifest, with some requiring human annotation and some not needing it, you can include a `isHumanAnnotationRequired` value in each data object. You can then use code in your pre-annotation Lambda to read the value from the data object and set the value in your Lambda output.

The pre-annotation Lambda runs first
Before any tasks are available to workers, your entire manifest will be processed into an intermediate form, using your Lambda. This means you won't be able to change your Lambda part of the way through a labeling job and see that have an impact on the remaining tasks.

Post-annotation Lambda

When the worker has completed the task, Ground Truth will send the results to your **Post-annotation Lambda**. This Lambda is generally used for Annotation Consolidation (p. 537). The request object will come in like this:

Example of a post-labeling task request

```
{
    "version": "2018-10-16",
    "labelingJobArn": <labelingJobArn>,
    "labelCategories": [<string>],
    "labelAttributeName": <string>,
    "roleArn" : "string",
    "payload": {
        "s3Uri": <string>
    }
 }
```

Post-labeling task Lambda permissions

The actual annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple pass through function, you need to assign the necessary permissions to your Lambda to read files from your S3 bucket.

In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

Example of an annotation data file

```
[
    {
        "datasetObjectId": <string>,
        "dataObject": {
            "s3Uri": <string>,
            "content": <string>
        },
        "annotations": [{
            "workerId": <string>,
            "annotationData": {
                "content": <string>,
                "s3Uri": <string>
            }
        }]
    }
]
```

Essentially, all the fields from your form will be in the `content` object. At this point you can start running data consolidation algorithms on the data, using an AWS database service to store results. Or you can pass some processed/optimized results back to Ground Truth for storage in your consolidated annotation manifests in the S3 bucket you specify for output during the configuration of the labeling job.

In return, Ground Truth will require a response formatted like this:

Example of expected return data

```
[
  {
        "datasetObjectId": <string>,
        "consolidatedAnnotation": {
            "content": {
                "<labelattributename>": {
                    # ... label content
                }
            }
        }
    },
    {
        "datasetObjectId": <string>,
        "consolidatedAnnotation": {
            "content": {
                "<labelattributename>": {
                    # ... label content
                }
            }
        }
    }
    .
```

```
    .
    .
]
```

At this point, all the data you're sending to your S3 bucket, other than the `datasetObjectId` will be in the `content` object.

That will result in an entry in your job's consolidation manifest like this:

Example of label format in output manifest

```
{   "source-ref"/"source" : "<s3uri or content>",
    "<labelAttributeName>": {
        # ... label content from you
    },
    "<labelAttributeName>-metadata": { # This will be added by Ground Truth
        "job_name": <labelingJobName>,
        "type": "groundTruth/custom",
        "human-annotated": "yes",
        "creation_date": <date> # Timestamp of when received from Post-labeling Lambda
    }
}
```

Because of the potentially complex nature of a custom template and the data it collects, Ground Truth does not offer further processing of the data or insights into it.

Next

Custom Workflows via the API

When you have created your custom UI template (Step 2) and processing Lambda functions (Step 3), you should place the template in an Amazon S3 bucket with a file name format of: `<FileName>.liquid.html`.

Use the CreateLabelingJob (p. 643) action to configure your task. You'll use the location of a custom template (Step 2: Creating your custom labeling task template (p. 558)) stored in a `<filename>.liquid.html` file on S3 as the value for the `UiTemplateS3Uri` field in the UiConfig (p. 1036) object within the HumanTaskConfig (p. 907) object.

For the AWS Lambda tasks described in Step 3: Processing with AWS Lambda (p. 574), the post-annotation task's ARN will be used as the value for the `AnnotationConsolidationLambdaArn` field, and the pre-annotation task will be used as the value for the `PreHumanTaskLambdaArn.`

HTML Elements Reference

Amazon SageMaker Ground Truth provides customers with the ability to design their own custom task templates in HTML. Documentation of how to implement custom templates is available in Creating Custom Labeling Workflows (p. 557). Below is a list of enhanced HTML elements that make building a custom template easier and provide a familiar UI for workers.

Topics

- crowd-bounding-box (p. 578)
- crowd-image-classifier (p. 581)
- crowd-classifier (p. 582)

crowd-bounding-box

A widget for drawing rectangles on an image and assigning a label to the portion of the image that is enclosed in each rectangle.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a rectangle. **Limit:** 10 labels.

name

The name of this widget. It's used as a key for the widget's input in the form output.

src

The URL of the image on which to draw bounding boxes.

initial-value

An array of JSON objects, each of which sets a bounding box when the component is loaded. Each JSON object in the array contains the following properties.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the <crowd-bounding-box> element.
- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: full-instructions (p. 579), short-instructions (p. 579)

Regions

The following regions are required by this element.

full-instructions

General instructions about how to draw bounding boxes.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

boundingBoxes

An array of JSON objects, each of which specifies a bounding box that has been created by the worker. Each JSON object in the array contains the following properties.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the <crowd-bounding-box> element.
- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following are samples of outputs from common use scenarios for this element.

Single Label, Single Box / Multiple Label, Single Box

```
[
  {
    "annotatedResult": {
      "boundingBoxes": [
        {
          "height": 401,
          "label": "Dog",
          "left": 243,
          "top": 117,
          "width": 187
        }
      ],
      "inputImageProperties": {
        "height": 533,
        "width": 800
      }
    }
  }
]
```

Single Label, Multiple Box

```
[
  {
    "annotatedResult": {
      "boundingBoxes": [
        {
          "height": 401,
          "label": "Dog",
          "left": 243,
          "top": 117,
          "width": 187
        },
        {
          "height": 283,
          "label": "Dog",
          "left": 684,
          "top": 120,
          "width": 116
        }
      ],
      "inputImageProperties": {
        "height": 533,
        "width": 800
      }
    }
  }
]
```

Multiple Label, Multiple Box

```
[
  {
    "annotatedResult": {
      "boundingBoxes": [
        {
```

```
          "height": 395,
          "label": "Dog",
          "left": 241,
          "top": 125,
          "width": 158
        },
        {
          "height": 298,
          "label": "Cat",
          "left": 699,
          "top": 116,
          "width": 101
        }
      ],
      "inputImageProperties": {
        "height": 533,
        "width": 800
      }
    }
  }
]
```

You could have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

crowd-image-classifier

A widget for classifying an image, which can be a JPG, PNG, or GIF, with no size limit.

Attributes

The following attributes are required by this element.

categories

A JSON formatted array of strings, each of which is a category that a worker can assign to the image. You should include "other" as a category, so that the worker can provide an answer. You can specify up to 10 categories.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

name

The name of this widget. It is used as a key for the widget's input in the form output.

src

The URL of the image to be classified.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: full-instructions (p. 582), short-instructions (p. 582)

Regions

The following regions are required by this element.

full-instructions

General instructions about how to do image classification.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The output of this element is a string that specifies one of the values defined in the *categories* attribute of the <crowd-image-classifier> element.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[
  {
    "<name>": {
      "label": "<value>"
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-classifier

A widget for classifying non-image content, such as audio, video, or text.

Attributes

The following attributes are supported by this element.

categories

A JSON formatted array of strings, each of which is a category that a worker can assign to the to the text. You should include "other" as a category, otherwise the worker my not be able to provide an answer.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

name

The name of this widget. It is used as a key for the widget's input in the form output.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: classification-target (p. 583), full-instructions (p. 583), short-instructions (p. 583)

Regions

The following regions are supported by this element.

classification-target

The content to be classified by the worker. This can be plain text or HTML. Examples of how the HTML can be used include *but are not limited to* embedding a video or audio player, embedding a PDF, or performing a comparison of two or more images.

full-instructions

General instructions about how to do text classification.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The output of this element is an object using the specified `name` value as a property name, and a string from the `categories` as the property's value.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[
  {
    "<name>": {
      "label": "<value>"
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-instance-segmentation

A widget for identifying individual instances of specific objects within an image and creating a colored overlay for each labeled instance.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to an instance of an object in the image. Workers can generate different overlay colors for each relevant instance by selecting "add instance" under the label in the tool.

name

The name of this widget. It is used as a key for the labeling data in the form output.

src

The URL of the image that is to be labeled.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: full-instructions (p. 584), short-instructions (p. 584)

Regions

The following regions are supported by this element.

full-instructions

General instructions about how to do image segmentation.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

labeledImage

A JSON Object containing a Base64 encoded PNG of the labels.

instances

A JSON Array containing objects with the instance labels and colors.

- **color** – The hexadecimal value of the label's RGB color in the `labeledImage` PNG.
- **label** – The label given to overlay(s) using that color. This value may repeat, because the different instances of the label are identified by their unique color.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 533,
        "width": 800
      },
      "instances": [
        {
          "color": "#1f77b4",
          "label": "<Label 1>":
        },
        {
          "color": "#2ca02c",
          "label": "<Label 1>":
        },
        {
          "color": "#ff7f0e",
          "label": "<Label 3>":
        },
      ],
      "labeledImage": {
        "pngImageData": "<Base-64 Encoded Data>"
      }
    }
  }
]
```

See Also

For more information, see the following.

crowd-semantic-segmentation

A widget for segmenting an image and assigning a label to each image segment.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to a segment of the image.

name

The name of this widget. It is used as a key for the widget's input in the form output.

src

The URL of the image that is to be segmented.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: full-instructions (p. 586), short-instructions (p. 586)

Regions

The following regions are supported by this element.

full-instructions

General instructions about how to do image segmentation.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

labeledImage

A JSON Object containing a Base64 encoded PNG of the labels.

labelMappings

A JSON Object containing objects with named with the segmentation labels.

- **color** – The hexadecimal value of the label's RGB color in the `labeledImage` PNG.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 533,
        "width": 800
      },
      "labelMappings": {
        "<Label 2>": {
          "color": "#ff7f0e"
        },
        "<label 3>": {
          "color": "#2ca02c"
        },
        "<label 1>": {
          "color": "#1f77b4"
        }
      },
      "labeledImage": {
        "pngImageData": "<Base-64 Encoded Data>"
      }
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-entity-annotation

A widget for labeling words, phrases, or character strings within a longer text.

Important: Self-contained Widget
Do not use `<crowd-entity-annotation>` element with the `<crowd-form>` element. It contains its own form submission logic and **Submit** button.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

A JSON formatted array of objects, each of which defines an annotation to apply to the text at initialization. Objects contain a `label` value that matches one in the `labels` attribute, an integer `startOffset` value for labeled span's starting unicode offset, and an integer `endOffset` value for the ending unicode offset.

Example

```
[
  {
```

```
        label: 'person',
        startOffset: 0,
        endOffset: 16
    },
    ...
]
```

labels

A JSON formatted array of objects, each of which contains:

- **`label`** (required): The name used to identify entities.
- **`fullDisplayName`** (optional): Used for the label list in the task widget. Defaults to the label value if not specified.
- **`shortDisplayName`** (optional): An abbreviation of 3-4 letters to display above selected entities. Defaults to the label value if not specified.

  > **`shortDisplayName` is highly recommended**
  > Values displayed above the selections can overlap and create difficulty managing labeled entities in the workspace. Providing a 3-4 character `shortDisplayName` for each label is highly recommended to prevent overlap and keep the workspace manageable for your workers.

Example

```
[
  {
    label: 'person',
    shortDisplayName: 'per',
    fullDisplayName: 'person'
  }
]
```

name

Serves as the widget's name in the DOM. It is also used as the label attribute name in form output and the output manifest.

text

The text to be annotated. The templating system escapes quotes and HTML strings by default. If your code is already escaped or partially escaped, see Variable filters (p. 561) for more ways to control escaping.

Element Hierarchy

This element has the following parent and child elements.

- **Child elements**: full-instructions (p. 588), short-instructions (p. 589)

Regions

The following regions are supported by this element.

full-instructions

General instructions about how to work with the widget.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

entities

A JSON object that specifies the start, end, and label of an annotation. This object contains the following properties.

- **label** – The assigned label.
- **startOffset** – The Unicode offset of the beginning of the selected text.
- **endOffset** – The Unicode offset of the first character after the selection.

Example : Sample Element Outputs

The following is a sample of the output from this element.

```
{
  "myAnnotatedResult": {
    "entities": [
      {
        "endOffset": 54,
        "label": "person",
        "startOffset": 47
      },
      {
        "endOffset": 97,
        "label": "event",
        "startOffset": 93
      },
      {
        "endOffset": 219,
        "label": "date",
        "startOffset": 212
      },
      {
        "endOffset": 271,
        "label": "location",
        "startOffset": 260
      }
    ]
  }
}
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-alert

A message that alerts the worker to a current situation.

Attributes

The following attributes are supported by this element.

dismissible

A Boolean switch that, if present, allows the message to be closed by the worker.

type

A string that specifies the type of message to be displayed. The possible values are "info" (the default), "success", "error", and "warning".

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-badge

An icon that floats over the top right corner of another element to which it is attached.

Attributes

The following attributes are supported by this element.

for

A string that specifies the ID of the element to which the badge is attached.

icon

A string that specifies the icon to be displayed in the badge. The string must be either the name of an icon from the open-source *iron-icons* set, which is pre-loaded, or the URL to a custom icon.

This attribute overrides the *label* attribute.

label

The text to display in the badge. Three characters or less is recommended because text that is too large will overflow the badge area. An icon can be displayed instead of text by setting the *icon* attribute.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

-
-

crowd-button

A styled button that represents some action.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

form-action

A switch that either submits its parent element, if set to "submit", or resets its parent <crowd-form> element, if set to "reset".

href

The URL to an online resource. Use this property if you need a link styled as a button.

icon

A string that specifies the icon to be displayed next to the button's text. The string must be either the name of an icon from the open-source *iron-icons* set, which is pre-loaded, or the URL to a custom icon.

The icon is positioned to either the left or the right of the text, as specified by the *icon-align* attribute.

icon-align

The left or right position of the icon relative to the button's text. The default is "left".

icon-url

A URL to a custom image for the icon. A custom image can be used in place of a standard icon that is specified by the *icon* attribute.

loading

A Boolean switch that, if present, displays the button as being in a loading state. This attribute has precedence over the *disabled* attribute if both attributes are present.

target

When you use the `href` attribute to make the button act as a hyperlink to a specific URL, the `target` attribute optionally targets a frame or window where the linked URL should load.

variant

The general style of the button. Use "primary" for primary buttons, "normal" for secondary buttons, "link" for tertiary buttons, or "icon" to display only the icon without text.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-card

A box with an elevated appearance for displaying information.

Attributes

The following attributes are supported by this element.

heading

The text displayed at the top of the box.

image

A URL to an image to be displayed within the box.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-checkbox

A UI component that can be checked or unchecked allowing a user to select multiple options from a set.

Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the check box as checked.

A Boolean switch that, if present, displays the check box as disabled and prevents it from being checked.

name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

required

A Boolean switch that, if present, requires the worker to provide input.

value

A string used as the name for the check box state in the output. Defaults to "on" if not specified.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

Output

Provides a JSON object. The `name` string is the object name and the `value`string is the property name for a Boolean value based on the check box state; true if checked, false if not checked.

Example : Sample Element Outputs

Using the same `name` value for multiple boxes.

```
<!-- INPUT -->
<div><crowd-checkbox name="myformbit" value="Red"> Red </div>
<div><crowd-checkbox name="myformbit" value="Yellow"> Yellow </div>
<div><crowd-checkbox name="myformbit" value="Green"> Green </div>
```

```
//Output with "Red" checked
[
  {
    "myformbit": {
      "Green": false,
      "Red": true,
      "Yellow": false
    }
  }
]
```

Note that all three color values are properties of a single object.

Using different `name` values for each box.

```
<!-- INPUT -->
<div><crowd-checkbox name="Stop" value="Red"> Red </div>
<div><crowd-checkbox name="Slow" value="Yellow"> Yellow </div>
```

```
<div><crowd-checkbox name="Go" value="Green"> Green </div>
```

```
//Output with "Red" checked
[
  {
    "Go": {
      "Green": false
    },
    "Slow": {
      "Yellow": false
    },
    "Stop": {
      "Red": true
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-fab

A floating button with an image in its center.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the floating button as disabled and prevents clicks.

icon

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source *iron-icons* set, which is pre-loaded, or the URL to a custom icon.

label

A string consisting of a single character that can be used instead of an icon. Emojis or multiple characters may result in the button displaying an ellipsis instead.

title

A string that will display as a tool tip when the mouse hovers over the button.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

crowd-form

The form wrapper for all custom tasks. Sets and implements important actions for the proper submission of your form data.

If a crowd-button (p. 591) of type "submit" is not included inside the <crowd-form> element, it will automatically be appended within the <crowd-form> element.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: none
- **Child elements**: Any of the UI Template (p. 577) elements

Element Events

The `crowd-form` element extends the standard HTML `form` element and inherits its events, such as `onclick` and `onsubmit`.

See Also

For more information, see the following.

crowd-icon-button

A button with an image placed in the center. When the user touches the button, a ripple effect emanates from the center of the button.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

icon

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source *iron-icons* set, which is pre-loaded, or the URL to a custom icon.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-input

A box that accepts input data.

> **Cannot be self-closing**
> Unlike the `input` element in the HTML standard, this element cannot be self-closed by putting a slash before the ending bracket, e.g. `<crowd-input ... />`. It must be followed with a `</crowd-input>` to close the element.

Attributes

The following attributes are supported by this element.

allowed-pattern

A regular expression that is used with the *auto-validate* attribute to ignore non-matching characters as the worker types.

auto-focus

When the value is set to true, the browser places focus inside the input area after loading. This way, the worker can start typing without having to select it first.

auto-validate

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the *error-message* and *allowed-pattern* attributes.

disabled

A Boolean switch that, if present, displays the input area as disabled.

error-message

The text to be displayed below the input field, on the left side, if validation fails.

label

A string that is displayed inside a text field.

This text shrinks and rises up above a text field when the worker starts typing in the field or when the *value* attribute is set.

max-length

A maximum number of characters the input will accept. Input beyond this limit is ignored.

min-length

A minimum length for the input in the field

name

Sets the name of the input to be used in the DOM and the output of the form.

placeholder

A string value that is used as placeholder text, displayed until the worker starts entering data into the input, It is not used as a default value.

required

A Boolean switch that, if present, requires the worker to provide input.

type

Takes a string to set the HTML5 `input-type` behavior for the input. Examples include `file` and `date`.

value

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

Output

Provides a `name` string as the property name, and the text that was entered in the field as its value.

Example : Sample JSON Output

The values for multiple elements are output in the same object, with their `name` attribute value as their property name. Elements with no input do not appear in the output. For example, let's use three inputs:

```
<crowd-input name="tag1" label="Word/phrase 1"></crowd-input>
<crowd-input name="tag2" label="Word/phrase 2"></crowd-input>
<crowd-input name="tag3" label="Word/phrase 3"></crowd-input>
```

This is the output if only two have input:

```
[
  {
    "tag1": "blue",
    "tag2": "red"
  }
]
```

This means any code built to parse these results should be able to handle the presence or absence of each input in the answers.

See Also

For more information, see the following.

-
-

crowd-instructions

An element that displays instructions on three tabbed pages, **Summary**, **Detailed Instructions**, and **Examples**, when the worker clicks on a link or button.

Attributes

The following attributes are supported by this element.

link-text

The text to display for opening the instructions. The default is **Click for instructions**.

link-type

A string that specifies the type of trigger for the instructions. The possible values are "link" (default) and "button".

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**:
- **Child elements**: none

Regions

The following regions are supported by this element.

detailed-instructions

Content that provides specific instructions for a task. This appears on the page of the "Detailed Instructions" tab.

negative-example

Content that provides examples of inadequate task completion. This appears on the page of the "Examples" tab. More than one example may be provided within this element.

positive-example

Content that provides examples of proper task completion. This appears on the page of the "Examples" tab.

short-summary

A brief statement that summarizes the task to be completed. This appears on the page of the "Summary" tab. More than one example may be provided within this element.

See Also

For more information, see the following.

crowd-keypoint

Generates a tool to select and annotate key points on an image.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

An array, in JSON format, of keypoints to be applied to the image on start. For example:

```
initial-value="[
  {
    'label': 'Left Eye',
    'x': 1022,
    'y': 429
  },
  {
    'label': 'Beak',
    'x': 941,
    'y': 403
  }
]
```

> **Note**
> Please note that label values used in this attribute must have a matching value in the `labels` attribute or the point will not be rendered.

labels

An array, in JSON format, of strings to be used as keypoint annotation labels.

name

A string used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

src

The source URI of the image to be annotated.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)

- **Child elements**: full-instructions (p. 600), short-instructions (p. 600)

Regions

The following regions are required by this element.

full-instructions

General instructions about how to annotate the image.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

keypoints

An array of JSON objects containing the coordinates and label of a keypoint. Each object contains the following properties.

- **label** – The assigned label for the keypoint.
- **x** – The X coordinate, in pixels, of the keypoint on the image.
- **y** – The Y coordinate, in pixels, of the keypoint on the image.

> **Note**
> X and Y coordinates are based on 0,0 being the top left corner of the image.

Example : Sample Element Outputs

The following is a sample output from using this element.

```
[
  {
    "crowdKeypoint": {
      "inputImageProperties": {
        "height": 1314,
        "width": 962
      },
      "keypoints": [
        {
          "label": "dog",
          "x": 155,
          "y": 275
        },
        {
          "label": "cat",
```

```
        "x": 341,
        "y": 447
      },
      {
        "label": "cat",
        "x": 491,
        "y": 513
      },
      {
        "label": "dog",
        "x": 714,
        "y": 578
      },
      {
        "label": "cat",
        "x": 712,
        "y": 763
      },
      {
        "label": "cat",
        "x": 397,
        "y": 814
      }
    ]
  }
 }
]
```

You may have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-modal

A small window that pops up on the display when it is opened.

Attributes

The following attributes are supported by this element.

link-text

The text to display for opening the modal. The default is "Click to open modal".

link-type

A string that specifies the type of trigger for the modal. The possible values are "link" (default) and "button".

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)

- **Child elements**: none

See Also

For more information, see the following.

crowd-polygon

A widget for drawing polygons on an image and assigning a label to the portion of the image that is enclosed in each polygon.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a polygon.

name

The name of this widget. It's used as a key for the widget's input in the form output.

src

The URL of the image on which to draw polygons.

initial-value

An array of JSON objects, each of which defines a polygon to be drawn when the component is loaded. Each JSON object in the array contains the following properties.

- **label** – The text assigned to the polygon as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the <crowd-polygon> element.
- **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon.

Example

An `initial-value` attribute might look something like this.

```
initial-value =
  '[
      {
      "label": "dog",
      "vertices":
        [
```

```
              {
                  "x": 570,
                  "y": 239
              },
            ...
              {
                  "x": 759,
                  "y": 281
              }
          ]
      }
  ]'
```

Because this will be within an HTML element, the JSON array must be enclosed in single or double quotes. The example above uses single quotes to encapsulate the JSON and double quotes within the JSON itself. If you must mix single and double quotes inside your JSON, replace them with their HTML entity codes (" for double quote, ' for single) to safely escape them.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: full-instructions (p. 603), short-instructions (p. 603)

Regions

The following regions are required.

full-instructions

General instructions about how to draw polygons.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

polygons

An array of JSON objects, each of which describes a polygon that has been created by the worker. Each JSON object in the array contains the following properties.

- **label** – The text assigned to the polygon as part of the labeling task.
- **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon. The top left corner of the image is 0,0.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following are samples of outputs from common use scenarios for this element.

Single Label, Single Polygon

```
{
    "annotatedResult":
    {
      "inputImageProperties": {
        "height": 853,
        "width": 1280
      },
      "polygons":
      [
        {
          "label": "dog",
          "vertices":
          [
            {
              "x": 570,
              "y": 239
            },
            {
              "x": 603,
              "y": 513
            },
            {
              "x": 823,
              "y": 645
            },
            {
              "x": 901,
              "y": 417
            },
            {
              "x": 759,
              "y": 281
            }
          ]
        }
      ]
    }
  }
]
```

Single Label, Multiple Polygons

```
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 853,
        "width": 1280
      },
      "polygons": [
        {
          "label": "dog",
          "vertices": [
            {
              "x": 570,
              "y": 239
            },
            {
```

```
              "x": 603,
              "y": 513
            },
            {
              "x": 823,
              "y": 645
            },
            {
              "x": 901,
              "y": 417
            },
            {
              "x": 759,
              "y": 281
            }
          ]
        },
        {
          "label": "dog",
          "vertices": [
            {
              "x": 870,
              "y": 278
            },
            {
              "x": 908,
              "y": 446
            },
            {
              "x": 1009,
              "y": 602
            },
            {
              "x": 1116,
              "y": 519
            },
            {
              "x": 1174,
              "y": 498
            },
            {
              "x": 1227,
              "y": 479
            },
            {
              "x": 1179,
              "y": 405
            },
            {
              "x": 1179,
              "y": 337
            }
          ]
        }
      ]
    }
  }
]
```

Multiple Labels, Multiple Polygons

```
[
  {
    "annotatedResult": {
      "inputImageProperties": {
```

```
          "height": 853,
          "width": 1280
        },
        "polygons": [
          {
            "label": "dog",
            "vertices": [
              {
                "x": 570,
                "y": 239
              },
              {
                "x": 603,
                "y": 513
              },
              {
                "x": 823,
                "y": 645
              },
              {
                "x": 901,
                "y": 417
              },
              {
                "x": 759,
                "y": 281
              }
            ]
          },
          {
            "label": "cat",
            "vertices": [
              {
                "x": 870,
                "y": 278
              },
              {
                "x": 908,
                "y": 446
              },
              {
                "x": 1009,
                "y": 602
              },
              {
                "x": 1116,
                "y": 519
              },
              {
                "x": 1174,
                "y": 498
              },
              {
                "x": 1227,
                "y": 479
              },
              {
                "x": 1179,
                "y": 405
              },
              {
                "x": 1179,
                "y": 337
              }
            ]
          }
```

```
        ]
      }
    }
]
```

You could have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-radio-button

A button that can be either checked or unchecked. When radio buttons are inside a radio group, exactly one radio button in the group can be checked at any time.

Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the radio button as checked.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents it from being checked.

name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

> **Note**
> If you use the buttons outside of a crowd-radio-group (p. 608) element, but with the same
> `name` string and different `value` strings, the `name` object in the output will contain a Boolean
> value for each `value` string. To ensure that only one button in a group is selected, make them
> children of a crowd-radio-group (p. 608) element and use different name values.

value

A property name for the element's boolean value. If not specified, it uses "on" as the default, e.g.
`{ "<name>": { "<value>": <true or false> } }`.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-radio-group (p. 608)
- **Child elements**: none

Output

Outputs an object with the following pattern: `{ "<name>": { "<value>": <true or false> } }`.
If you use the buttons outside of a crowd-radio-group (p. 608) element, but with the same `name`

string and different `value` strings, the name object will contain a Boolean value for each `value` string. To ensure that only one in a group of buttons is selected, make them children of a crowd-radio-group (p. 608) element and use different name values.

Example Sample output of this element

```
[
  {
    "btn1": {
      "yes": true
    },
    "btn2": {
      "no": false
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-radio-group

A group of radio buttons. Only one radio button within the group can be selected. Choosing one radio button clears any previously chosen radio button within the same group.

Attributes

No special attributes are supported by this element.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: crowd-radio-button (p. 607)

Output

Outputs an array of objects representing the crowd-radio-button (p. 607) elements within it.

Example Sample of Element Output

```
[
  {
    "btn1": {
      "yes": true
    },
    "btn2": {
      "no": false
    }
  }
]
```

See Also

For more information, see the following.

crowd-slider

A bar with a sliding knob that allows a worker to select a value from a range of values by moving the knob. The slider makes it a great choice for settings that reflect intensity levels, such as volume, brightness, or color saturation.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the slider as disabled.

editable

A Boolean switch that, if present, displays an up/down button that can be chosen to select the value.

Selecting the value via the up/down button is an alternative to selecting the value by moving the knob on the slider. The knob on the slider will move synchronously with the up/down button choices.

max

A number that specifies the maximum value on the slider.

min

A number that specifies the minimum value on the slider.

name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

pin

A Boolean switch that, if present, displays the current value above the knob as the knob is moved.

required

A Boolean switch that, if present, requires the worker to provide input.

secondary-progress

When used with a `crowd-slider-secondary-color` CSS attribute, the progress bar is colored to the point represented by the `secondary-progress`. For example, if this was representing the progress on a streaming video, the `value` would represent where the viewer was in the video timeline. The `secondary-progress` value would represent the point on the timeline to which the video had buffered.

step

A number that specifies the difference between selectable values on the slider.

value

A preset that becomes the default if the worker does not provide input.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-tab

A component styled to look like a tab with information below.

Attributes

The following attributes are supported by this element.

header

The text appearing on the tab. This is usually some short descriptive name indicative of the information contained below the tab.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-tabs (p. 610)
- **Child elements**: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-tabs

A container for tabbed information.

Attributes

This element has no attributes.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: crowd-tab (p. 610)

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-text-area

A field for text input.

Attributes

The following attributes are supported by this element.

auto-focus

A Boolean switch that, if present, puts the cursor in this element on-load so that users can immediately begin typing without having to click inside the element.

auto-validate

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the *error-message* and *allowed-pattern* attributes.

char-counter

A Boolean switch that, if present, puts a small text field beneath the lower-right corner of the element, displaying the number of characters inside the element.

disabled

A Boolean switch that, if present, displays the input area as disabled.

error-message

The text to be displayed below the input field, on the left side, if validation fails.

label

A string that is displayed inside a text field.

This text shrinks and rises up above a text field when the worker starts typing in the field or when the *value* attribute is set.

max-length

An integer that specifies the maximum number of characters allowed by the element. Characters typed or pasted beyond the maximum are ignored.

max-rows

An integer that specifies the maximum number of rows of text that are allowed within a crowd-text-area. Normally the element expands to accommodate new rows. If this is set, after the number of rows exceeds it, content scrolls upward out of view and a scrollbar control appears.

name

A string used to represent the element's data in the output.

placeholder

A string presented to the user as placeholder text. It disappears after the user puts something in the input area.

rows

An integer that specifies the height of the element in rows of text.

value

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

Output

This element outputs the `name` as a property name and the element's text contents as the value. Carriage returns in the text are represented as `\n`.

Example Sample output for this element

```
[
  {
    "textInput1": "This is the text; the text that\nmakes the crowd go wild."
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-toast

A subtle notification that temporarily appears on the display. Only one crowd-toast is visible.

Attributes

The following attributes are supported by this element.

duration

A number that specifies the duration, in seconds, that the notification appears on the screen.

text

The text to display in the notification.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

crowd-toggle-button

A button that acts as an ON/OFF switch, toggling a state.

Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the button switched to the ON position.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents toggling.

invalid

When in an off position, a button using this attribute, will display in an alert color. The standard is red, but may be changed in CSS. When toggled on, the button will display in the same color as other buttons in the on position.

name

A string that is used to identify the answer submitted by the worker. This value matches a key in the JSON object that specifies the answer.

required

A Boolean switch that, if present, requires the worker to provide input.

value

A value used in the output as the property name for the element's Boolean state. Defaults to "on" if not provided.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 595)
- **Child elements**: none

Output

This element outputs the `name` as the name of an object, containing the `value` as a property name and the element's state as Boolean value for the property. If no value for the element is specified, the property name defaults to "on."

Example Sample output for this element

```
[
  {
    "theToggler": {
      "on": true
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 532)
- HTML Elements Reference (p. 577)

Limits and Supported Regions

For Amazon SageMaker service limits, see Amazon SageMaker Limits.

For information about requesting limit increases for AWS resources, see AWS Service Limits.

For a list of the AWS Regions supporting Amazon SageMaker, see Amazon SageMaker Regions.

API Reference

This section contains the API Reference documentation.

Topics

Actions

The following actions are supported by Amazon SageMaker Service:

The following actions are supported by Amazon SageMaker Runtime:

Amazon SageMaker Service

The following actions are supported by Amazon SageMaker Service:

AddTags

Service: Amazon SageMaker Service

Adds or overwrites one or more tags for the specified Amazon SageMaker resource. You can add tags to notebook instances, training jobs, hyperparameter tuning jobs, batch transform jobs, models, labeling jobs, work teams, endpoint configurations, and endpoints.

Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.

> **Note**
> Tags that you add to a hyperparameter tuning job by calling this API are also added to any training jobs that the hyperparameter tuning job launches after you call this API, but not to training jobs that the hyperparameter tuning job launched before you called this API. To make sure that the tags associated with a hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches, add the tags when you first create the tuning job by specifying them in the `Tags` parameter of CreateHyperParameterTuningJob (p. 638)

Request Syntax

```
{
    "ResourceArn": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

ResourceArn (p. 620)

The Amazon Resource Name (ARN) of the resource that you want to tag.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:.*`

Required: Yes

Tags (p. 620)

An array of `Tag` objects. Each tag is a key-value pair. Only the `key` parameter is required. If you don't specify a value, Amazon SageMaker sets the value to an empty string.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: Yes

Response Syntax

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Tags (p. 621)

A list of tags associated with the Amazon SageMaker resource.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateAlgorithm

Service: Amazon SageMaker Service

Create a machine learning algorithm that you can use in Amazon SageMaker and list in the AWS Marketplace.

Request Syntax

```
{
    "AlgorithmDescription": "string",
    "AlgorithmName": "string",
    "CertifyForMarketplace": boolean,
    "InferenceSpecification": {
        "Containers": [
            {
                "ContainerHostname": "string",
                "Image": "string",
                "ImageDigest": "string",
                "ModelDataUrl": "string",
                "ProductId": "string"
            }
        ],
        "SupportedContentTypes": [ "string" ],
        "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
        "SupportedResponseMIMETypes": [ "string" ],
        "SupportedTransformInstanceTypes": [ "string" ]
    },
    "TrainingSpecification": {
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "SupportedHyperParameters": [
            {
                "DefaultValue": "string",
                "Description": "string",
                "IsRequired": boolean,
                "IsTunable": boolean,
                "Name": "string",
                "Range": {
                    "CategoricalParameterRangeSpecification": {
                        "Values": [ "string" ]
                    },
                    "ContinuousParameterRangeSpecification": {
                        "MaxValue": "string",
                        "MinValue": "string"
                    },
                    "IntegerParameterRangeSpecification": {
                        "MaxValue": "string",
                        "MinValue": "string"
                    }
                },
                "Type": "string"
            }
        ],
        "SupportedTrainingInstanceTypes": [ "string" ],
        "SupportedTuningJobObjectiveMetrics": [
            {
                "MetricName": "string",
                "Type": "string"
            }
        ],
```

```
            "SupportsDistributedTraining": boolean,
            "TrainingChannels": [
                {
                    "Description": "string",
                    "IsRequired": boolean,
                    "Name": "string",
                    "SupportedCompressionTypes": [ "string" ],
                    "SupportedContentTypes": [ "string" ],
                    "SupportedInputModes": [ "string" ]
                }
            ],
            "TrainingImage": "string",
            "TrainingImageDigest": "string"
        },
        "ValidationSpecification": {
            "ValidationProfiles": [
                {
                    "ProfileName": "string",
                    "TrainingJobDefinition": {
                        "HyperParameters": {
                            "string" : "string"
                        },
                        "InputDataConfig": [
                            {
                                "ChannelName": "string",
                                "CompressionType": "string",
                                "ContentType": "string",
                                "DataSource": {
                                    "FileSystemDataSource": {
                                        "DirectoryPath": "string",
                                        "FileSystemAccessMode": "string",
                                        "FileSystemId": "string",
                                        "FileSystemType": "string"
                                    },
                                    "S3DataSource": {
                                        "AttributeNames": [ "string" ],
                                        "S3DataDistributionType": "string",
                                        "S3DataType": "string",
                                        "S3Uri": "string"
                                    }
                                },
                                "InputMode": "string",
                                "RecordWrapperType": "string",
                                "ShuffleConfig": {
                                    "Seed": number
                                }
                            }
                        ],
                        "OutputDataConfig": {
                            "KmsKeyId": "string",
                            "S3OutputPath": "string"
                        },
                        "ResourceConfig": {
                            "InstanceCount": number,
                            "InstanceType": "string",
                            "VolumeKmsKeyId": "string",
                            "VolumeSizeInGB": number
                        },
                        "StoppingCondition": {
                            "MaxRuntimeInSeconds": number,
                            "MaxWaitTimeInSeconds": number
                        },
                        "TrainingInputMode": "string"
                    },
                    "TransformJobDefinition": {
                        "BatchStrategy": "string",
```

```
            "Environment": {
                "string" : "string"
            },
            "MaxConcurrentTransforms": number,
            "MaxPayloadInMB": number,
            "TransformInput": {
                "CompressionType": "string",
                "ContentType": "string",
                "DataSource": {
                    "S3DataSource": {
                        "S3DataType": "string",
                        "S3Uri": "string"
                    }
                },
                "SplitType": "string"
            },
            "TransformOutput": {
                "Accept": "string",
                "AssembleWith": "string",
                "KmsKeyId": "string",
                "S3OutputPath": "string"
            },
            "TransformResources": {
                "InstanceCount": number,
                "InstanceType": "string",
                "VolumeKmsKeyId": "string"
            }
        }
    }
    ],
    "ValidationRole": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmDescription (p. 622)

A description of the algorithm.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

Required: No

AlgorithmName (p. 622)

The name of the algorithm.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

CertifyForMarketplace (p. 622)

Whether to certify the algorithm so that it can be listed in AWS Marketplace.

Type: Boolean

Required: No

InferenceSpecification (p. 622)

Specifies details about inference jobs that the algorithm runs, including the following:

- The Amazon ECR paths of containers that contain the inference code and model artifacts.
- The instance types that the algorithm supports for transform jobs and real-time endpoints used for inference.
- The input and output content formats that the algorithm supports for inference.

Type: InferenceSpecification (p. 929) object

Required: No

TrainingSpecification (p. 622)

Specifies details about training jobs run by this algorithm, including the following:

- The Amazon ECR path of the container and the version digest of the algorithm.
- The hyperparameters that the algorithm supports.
- The instance types that the algorithm supports for training.
- Whether the algorithm supports distributed training.
- The metrics that the algorithm emits to Amazon CloudWatch.
- Which metrics that the algorithm emits can be used as the objective metric for hyperparameter tuning jobs.
- The input channels that the algorithm supports for training data. For example, an algorithm might support `train`, `validation`, and `test` channels.

Type: TrainingSpecification (p. 1021) object

Required: Yes

ValidationSpecification (p. 622)

Specifies configurations for one or more training jobs and that Amazon SageMaker runs to test the algorithm's training code and, optionally, one or more batch transform jobs that Amazon SageMaker runs to test the algorithm's inference code.

Type: AlgorithmValidationSpecification (p. 870) object

Required: No

Response Syntax

```
{
    "AlgorithmArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AlgorithmArn (p. 625)

The Amazon Resource Name (ARN) of the new algorithm.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:algorithm/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateCodeRepository

Service: Amazon SageMaker Service

Creates a Git repository as a resource in your Amazon SageMaker account. You can associate the repository with notebook instances so that you can use Git source control for the notebooks you create. The Git repository is a resource in your Amazon SageMaker account, so it can be associated with more than one notebook instance, and it persists independently from the lifecycle of any notebook instances it is associated with.

The repository can be hosted either in AWS CodeCommit or in any other Git repository.

Request Syntax

```
{
    "CodeRepositoryName": "string",
    "GitConfig": {
        "Branch": "string",
        "RepositoryUrl": "string",
        "SecretArn": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CodeRepositoryName (p. 627)

The name of the Git repository. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

GitConfig (p. 627)

Specifies details about the repository, including the URL where the repository is located, the default branch, and credentials to use to access the repository.

Type: GitConfig (p. 905) object

Required: Yes

Response Syntax

```
{
    "CodeRepositoryArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeRepositoryArn (p. 627)

The Amazon Resource Name (ARN) of the new repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateCompilationJob

Service: Amazon SageMaker Service

Starts a model compilation job. After the model has been compiled, Amazon SageMaker saves the resulting model artifacts to an Amazon Simple Storage Service (Amazon S3) bucket that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts with AWS IoT Greengrass. In that case, deploy them as an ML resource.

In the request body, you provide the following:

- A name for the compilation job
- Information about the input model artifacts
- The output location for the compiled model and the device (target) that the model runs on
- `The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker assumes to perform the model compilation job`

You can also provide a `Tag` to track the model compilation job's resource use and costs. The response body contains the `CompilationJobArn` for the compiled job.

To stop a model compilation job, use . To get information about a particular model compilation job, use . To get information about multiple model compilation jobs, use .

Request Syntax

```
{
    "CompilationJobName": "string",
    "InputConfig": {
        "DataInputConfig": "string",
        "Framework": "string",
        "S3Uri": "string"
    },
    "OutputConfig": {
        "S3OutputLocation": "string",
        "TargetDevice": "string"
    },
    "RoleArn": "string",
    "StoppingCondition": {
        "MaxRuntimeInSeconds": number,
        "MaxWaitTimeInSeconds": number
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see .

The request accepts the following data in JSON format.

CompilationJobName (p. 629)

A name for the model compilation job. The name must be unique within the AWS Region and within your AWS account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

InputConfig (p. 629)

Provides information about the location of input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Type: InputConfig (p. 931) object

Required: Yes

OutputConfig (p. 629)

Provides information about the output location for the compiled model and the target device the model runs on.

Type: OutputConfig (p. 975) object

Required: Yes

RoleArn (p. 629)

The Amazon Resource Name (ARN) of an IAM role that enables Amazon SageMaker to perform tasks on your behalf.

During model compilation, Amazon SageMaker needs your permission to:

- Read input data from an S3 bucket
- Write model artifacts to an S3 bucket
- Write logs to Amazon CloudWatch Logs
- Publish metrics to Amazon CloudWatch

You grant permissions for all of these tasks to an IAM role. To pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission. For more information, see Amazon SageMaker Roles.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StoppingCondition (p. 629)

Specifies a limit to how long a model compilation job can run. When the job reaches the time limit, Amazon SageMaker ends the compilation job. Use this API to cap model training costs.

Type: StoppingCondition (p. 1004) object

Required: Yes

Response Syntax

```
{
    "CompilationJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CompilationJobArn (p. 630)

If the action is successful, the service sends back an HTTP 200 response. Amazon SageMaker returns the following data in JSON format:

- `CompilationJobArn`: The Amazon Resource Name (ARN) of the compiled job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateEndpoint

Service: Amazon SageMaker Service

Creates an endpoint using the endpoint configuration specified in the request. Amazon SageMaker uses the endpoint to provision resources and deploy models. You create the endpoint configuration with the CreateEndpointConfig API.

> **Note**
> Use this API only for hosting models using Amazon SageMaker hosting services.
> You must not delete an `EndpointConfig` in use by an endpoint that is live or while the `UpdateEndpoint` or `CreateEndpoint` operations are being performed on the endpoint. To update an endpoint, you must create a new `EndpointConfig`.

The endpoint name must be unique within an AWS Region in your AWS account.

When it receives the request, Amazon SageMaker creates the endpoint, launches the resources (ML compute instances), and deploys the model(s) on them.

When Amazon SageMaker receives the request, it sets the endpoint status to `Creating`. After it creates the endpoint, it sets the status to `InService`. Amazon SageMaker can then process incoming requests for inferences. To check the status of an endpoint, use the DescribeEndpoint API.

For an example, see Exercise 1: Using the K-Means Algorithm Provided by Amazon SageMaker.

If any of the models hosted at this endpoint get model data from an Amazon S3 location, Amazon SageMaker uses AWS Security Token Service to download model artifacts from the S3 path you provided. AWS STS is activated in your IAM user account by default. If you previously deactivated AWS STS for a region, you need to reactivate AWS STS for that region. For more information, see Activating and Deactivating AWS STS i an AWS Region in the *AWS Identity and Access Management User Guide*.

Request Syntax

```
{
    "EndpointConfigName": "string",
    "EndpointName": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

EndpointConfigName (p. 632)

The name of an endpoint configuration. For more information, see CreateEndpointConfig.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

EndpointName (p. 632)

The name of the endpoint. The name must be unique within an AWS Region in your AWS account.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Tags (p. 632)

An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

Response Syntax

```
{
    "EndpointArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EndpointArn (p. 633)

The Amazon Resource Name (ARN) of the endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateEndpointConfig

Service: Amazon SageMaker Service

Creates an endpoint configuration that Amazon SageMaker hosting services uses to deploy models. In the configuration, you identify one or more models, created using the `CreateModel` API, to deploy and the resources that you want Amazon SageMaker to provision. Then you call the CreateEndpoint API.

> **Note**
> Use this API only if you want to use Amazon SageMaker hosting services to deploy models into production.

In the request, you define one or more `ProductionVariants`, each of which identifies a model. Each `ProductionVariant` parameter also describes the resources that you want Amazon SageMaker to provision. This includes the number and type of ML compute instances to deploy.

If you are hosting multiple models, you also assign a `VariantWeight` to specify how much traffic you want to allocate to each model. For example, suppose that you want to host two models, A and B, and you assign traffic weight 2 for model A and 1 for model B. Amazon SageMaker distributes two-thirds of the traffic to Model A, and one-third to model B.

Request Syntax

```
{
    "EndpointConfigName": "string",
    "KmsKeyId": "string",
    "ProductionVariants": [
        {
            "AcceleratorType": "string",
            "InitialInstanceCount": number,
            "InitialVariantWeight": number,
            "InstanceType": "string",
            "ModelName": "string",
            "VariantName": "string"
        }
    ],
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

EndpointConfigName (p. 635)

The name of the endpoint configuration. You specify this name in a CreateEndpoint request.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

KmsKeyId (p. 635)

The Amazon Resource Name (ARN) of a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance that hosts the endpoint.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

ProductionVariants (p. 635)

An list of `ProductionVariant` objects, one for each model that you want to host at this endpoint.

Type: Array of ProductionVariant (p. 981) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: Yes

Tags (p. 635)

A list of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

Response Syntax

```
{
    "EndpointConfigArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EndpointConfigArn (p. 636)

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint-config/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateHyperParameterTuningJob

Service: Amazon SageMaker Service

Starts a hyperparameter tuning job. A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm you choose and values for hyperparameters within ranges that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by an objective metric that you choose.

Request Syntax

```
{
    "HyperParameterTuningJobConfig": {
        "HyperParameterTuningJobObjective": {
            "MetricName": "string",
            "Type": "string"
        },
        "ParameterRanges": {
            "CategoricalParameterRanges": [
                {
                    "Name": "string",
                    "Values": [ "string" ]
                }
            ],
            "ContinuousParameterRanges": [
                {
                    "MaxValue": "string",
                    "MinValue": "string",
                    "Name": "string",
                    "ScalingType": "string"
                }
            ],
            "IntegerParameterRanges": [
                {
                    "MaxValue": "string",
                    "MinValue": "string",
                    "Name": "string",
                    "ScalingType": "string"
                }
            ]
        },
        "ResourceLimits": {
            "MaxNumberOfTrainingJobs": number,
            "MaxParallelTrainingJobs": number
        },
        "Strategy": "string",
        "TrainingJobEarlyStoppingType": "string"
    },
    "HyperParameterTuningJobName": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "TrainingJobDefinition": {
        "AlgorithmSpecification": {
            "AlgorithmName": "string",
            "MetricDefinitions": [
                {
                    "Name": "string",
                    "Regex": "string"
                }
            ],
```

```
            "TrainingImage": "string",
            "TrainingInputMode": "string"
        },
        "CheckpointConfig": {
            "LocalPath": "string",
            "S3Uri": "string"
        },
        "EnableInterContainerTrafficEncryption": boolean,
        "EnableManagedSpotTraining": boolean,
        "EnableNetworkIsolation": boolean,
        "InputDataConfig": [
            {
                "ChannelName": "string",
                "CompressionType": "string",
                "ContentType": "string",
                "DataSource": {
                    "FileSystemDataSource": {
                        "DirectoryPath": "string",
                        "FileSystemAccessMode": "string",
                        "FileSystemId": "string",
                        "FileSystemType": "string"
                    },
                    "S3DataSource": {
                        "AttributeNames": [ "string" ],
                        "S3DataDistributionType": "string",
                        "S3DataType": "string",
                        "S3Uri": "string"
                    }
                },
                "InputMode": "string",
                "RecordWrapperType": "string",
                "ShuffleConfig": {
                    "Seed": number
                }
            }
        ],
        "OutputDataConfig": {
            "KmsKeyId": "string",
            "S3OutputPath": "string"
        },
        "ResourceConfig": {
            "InstanceCount": number,
            "InstanceType": "string",
            "VolumeKmsKeyId": "string",
            "VolumeSizeInGB": number
        },
        "RoleArn": "string",
        "StaticHyperParameters": {
            "string" : "string"
        },
        "StoppingCondition": {
            "MaxRuntimeInSeconds": number,
            "MaxWaitTimeInSeconds": number
        },
        "VpcConfig": {
            "SecurityGroupIds": [ "string" ],
            "Subnets": [ "string" ]
        }
    },
    "WarmStartConfig": {
        "ParentHyperParameterTuningJobs": [
            {
                "HyperParameterTuningJobName": "string"
            }
        ],
        "WarmStartType": "string"
```

```
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

HyperParameterTuningJobConfig (p. 638)

> The HyperParameterTuningJobConfig (p. 922) object that describes the tuning job, including the
> search strategy, the objective metric used to evaluate training jobs, ranges of parameters to search,
> and resource limits for the tuning job. For more information, see Automatic Model Tuning (p. 288)
>
> Type: HyperParameterTuningJobConfig (p. 922) object
>
> Required: Yes

HyperParameterTuningJobName (p. 638)

> The name of the tuning job. This name is the prefix for the names of all training jobs that this tuning
> job launches. The name must be unique within the same AWS account and AWS Region. The name
> must have { } to { } characters. Valid characters are a-z, A-Z, 0-9, and : + = @ _ % - (hyphen). The
> name is not case sensitive.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 32.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

Tags (p. 638)

> An array of key-value pairs. You can use tags to categorize your AWS resources in different ways, for
> example, by purpose, owner, or environment. For more information, see AWS Tagging Strategies.
>
> Tags that you specify for the tuning job are also added to all training jobs that the tuning job
> launches.
>
> Type: Array of Tag (p. 1008) objects
>
> Array Members: Minimum number of 0 items. Maximum number of 50 items.
>
> Required: No

TrainingJobDefinition (p. 638)

> The HyperParameterTrainingJobDefinition (p. 916) object that describes the training jobs that
> this tuning job launches, including static hyperparameters, input data configuration, output data
> configuration, resource configuration, and stopping condition.
>
> Type: HyperParameterTrainingJobDefinition (p. 916) object
>
> Required: No

WarmStartConfig (p. 638)

> Specifies the configuration for starting the hyperparameter tuning job using one or more previous
> tuning jobs as a starting point. The results of previous tuning jobs are used to inform which
> combinations of hyperparameters to search over in the new tuning job.

All training jobs launched by the new hyperparameter tuning job are evaluated by using the objective metric. If you specify `IDENTICAL_DATA_AND_ALGORITHM` as the `WarmStartType` value for the warm start configuration, the training job that performs the best in the new tuning job is compared to the best training jobs from the parent tuning jobs. From these, the training job that performs the best as measured by the objective metric is returned as the overall best training job.

> **Note**
> All training jobs launched by parent hyperparameter tuning jobs and the new hyperparameter tuning jobs count against the limit of training jobs for the tuning job.

Type: HyperParameterTuningJobWarmStartConfig (p. 927) object

Required: No

Response Syntax

```
{
    "HyperParameterTuningJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

HyperParameterTuningJobArn (p. 641)

The Amazon Resource Name (ARN) of the tuning job. Amazon SageMaker assigns an ARN to a hyperparameter tuning job when you create it.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateLabelingJob

Service: Amazon SageMaker Service

Creates a job that uses workers to label the data objects in your input dataset. You can use the labeled data to train machine learning models.

You can select your workforce from one of three providers:

- A private workforce that you create. It can include employees, contractors, and outside experts. Use a private workforce when want the data to stay within your organization or when a specific set of skills is required.
- One or more vendors that you select from the AWS Marketplace. Vendors provide expertise in specific areas.
- The Amazon Mechanical Turk workforce. This is the largest workforce, but it should only be used for public data or data that has been stripped of any personally identifiable information.

You can also use *automated data labeling* to reduce the number of data objects that need to be labeled by a human. Automated data labeling uses *active learning* to determine if a data object can be labeled by machine or if it needs to be sent to a human worker. For more information, see Using Automated Data Labeling.

The data objects to be labeled are contained in an Amazon S3 bucket. You create a *manifest file* that describes the location of each object. For more information, see Using Input and Output Data.

The output can be used as the manifest file for another labeling job or as training data for your machine learning models.

Request Syntax

```
{
    "HumanTaskConfig": {
        "AnnotationConsolidationConfig": {
            "AnnotationConsolidationLambdaArn": "string"
        },
        "MaxConcurrentTaskCount": number,
        "NumberOfHumanWorkersPerDataObject": number,
        "PreHumanTaskLambdaArn": "string",
        "PublicWorkforceTaskPrice": {
            "AmountInUsd": {
                "Cents": number,
                "Dollars": number,
                "TenthFractionsOfACent": number
            }
        },
        "TaskAvailabilityLifetimeInSeconds": number,
        "TaskDescription": "string",
        "TaskKeywords": [ "string" ],
        "TaskTimeLimitInSeconds": number,
        "TaskTitle": "string",
        "UiConfig": {
            "UiTemplateS3Uri": "string"
        },
        "WorkteamArn": "string"
    },
    "InputConfig": {
        "DataAttributes": {
            "ContentClassifiers": [ "string" ]
        },
        "DataSource": {
            "S3DataSource": {
```

```
            "ManifestS3Uri": "string"
        }
    }
},
"LabelAttributeName": "string",
"LabelCategoryConfigS3Uri": "string",
"LabelingJobAlgorithmsConfig": {
    "InitialActiveLearningModelArn": "string",
    "LabelingJobAlgorithmSpecificationArn": "string",
    "LabelingJobResourceConfig": {
        "VolumeKmsKeyId": "string"
    }
},
"LabelingJobName": "string",
"OutputConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
},
"RoleArn": "string",
"StoppingConditions": {
    "MaxHumanLabeledObjectCount": number,
    "MaxPercentageOfInputDatasetLabeled": number
},
"Tags": [
    {
        "Key": "string",
        "Value": "string"
    }
]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

HumanTaskConfig (p. 643)

Configures the information required for human workers to complete a labeling task.

Type: HumanTaskConfig (p. 907) object

Required: Yes

InputConfig (p. 643)

Input data for the labeling job, such as the Amazon S3 location of the data objects and the location
of the manifest file that describes the data objects.

Type: LabelingJobInputConfig (p. 945) object

Required: Yes

LabelAttributeName (p. 643)

The attribute name to use for the label in the output manifest file. This is the key for the key/value
pair formed with the label that a worker assigns to the object. The name can't end with "-metadata".
If you are running a semantic segmentation labeling job, the attribute name must end with "-ref". If
you are running any other kind of labeling job, the attribute name must not end with "-ref".

Type: String

Length Constraints: Minimum length of 1. Maximum length of 127.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

LabelCategoryConfigS3Uri (p. 643)

The S3 URL of the file that defines the categories used to label the data objects.

The file is a JSON structure in the following format:

```
{

"document-version": "2018-11-28"

"labels": [

{

"label": "label 1"

},

{

"label": "label 2"

},

...

{

"label": "label n"

}

]

}
```

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: No

LabelingJobAlgorithmsConfig (p. 643)

Configures the information required to perform automated data labeling.

Type: LabelingJobAlgorithmsConfig (p. 939) object

Required: No

LabelingJobName (p. 643)

The name of the labeling job. This name is used to identify the job in a list of labeling jobs.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

OutputConfig (p. 643)

The location of the output data and the AWS Key Management Service key ID for the key used to encrypt the output data, if any.

Type: LabelingJobOutputConfig (p. 947) object

Required: Yes

RoleArn (p. 643)

The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during data labeling. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete data labeling.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StoppingConditions (p. 643)

A set of conditions for stopping the labeling job. If any of the conditions are met, the job is automatically stopped. You can use these conditions to control the cost of data labeling.

Type: LabelingJobStoppingConditions (p. 950) object

Required: No

Tags (p. 643)

An array of key/value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

Response Syntax

```
{
    "LabelingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

LabelingJobArn (p. 646)

The Amazon Resource Name (ARN) of the labeling job. You use this ARN to identify the labeling job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateModel

Service: Amazon SageMaker Service

Creates a model in Amazon SageMaker. In the request, you name the model and describe a primary container. For the primary container, you specify the docker image containing inference code, artifacts (from prior training), and custom environment map that the inference code uses when you deploy the model for predictions.

Use this API to create a model if you want to use Amazon SageMaker hosting services or run a batch transform job.

To host your model, you create an endpoint configuration with the `CreateEndpointConfig` API, and then create an endpoint with the `CreateEndpoint` API. Amazon SageMaker then deploys all of the containers that you defined for the model in the hosting environment.

To run a batch transform using your model, you start a job with the `CreateTransformJob` API. Amazon SageMaker uses your model and your dataset to get inferences which are then saved to a specified S3 location.

In the `CreateModel` request, you must define a container with the `PrimaryContainer` parameter.

In the request, you also provide an IAM role that Amazon SageMaker can assume to access model artifacts and docker image for deployment on ML compute hosting instances or for batch transform jobs. In addition, you also use the IAM role to manage permissions the inference code needs. For example, if the inference code access any other AWS resources, you grant necessary permissions via this role.

Request Syntax

```
{
    "Containers": [
        {
            "ContainerHostname": "string",
            "Environment": {
                "string" : "string"
            },
            "Image": "string",
            "ModelDataUrl": "string",
            "ModelPackageName": "string"
        }
    ],
    "EnableNetworkIsolation": boolean,
    "ExecutionRoleArn": "string",
    "ModelName": "string",
    "PrimaryContainer": {
        "ContainerHostname": "string",
        "Environment": {
            "string" : "string"
        },
        "Image": "string",
        "ModelDataUrl": "string",
        "ModelPackageName": "string"
    },
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "Subnets": [ "string" ]
    }
```

```
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

Containers (p. 648)

> Specifies the containers in the inference pipeline.
>
> Type: Array of ContainerDefinition (p. 886) objects
>
> Array Members: Maximum number of 5 items.
>
> Required: No

EnableNetworkIsolation (p. 648)

> Isolates the model container. No inbound or outbound network calls can be made to or from the
> model container.
>
> > **Note**
> > The Semantic Segmentation built-in algorithm does not support network isolation.
>
> Type: Boolean
>
> Required: No

ExecutionRoleArn (p. 648)

> The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker can assume to access
> model artifacts and docker image for deployment on ML compute instances or for batch transform
> jobs. Deploying on ML compute instances is part of model hosting. For more information, see
> Amazon SageMaker Roles.
>
> > **Note**
> > To be able to pass this role to Amazon SageMaker, the caller of this API must have the
> > `iam:PassRole` permission.
>
> Type: String
>
> Length Constraints: Minimum length of 20. Maximum length of 2048.
>
> Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`
>
> Required: Yes

ModelName (p. 648)

> The name of the new model.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

PrimaryContainer (p. 648)

> The location of the primary docker image containing inference code, associated artifacts, and
> custom environment map that the inference code uses when the model is deployed for predictions.

Type: ContainerDefinition (p. 886) object

Required: No

Tags (p. 648)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

VpcConfig (p. 648)

A VpcConfig object that specifies the VPC that you want your model to connect to. Control access to and from your model container by configuring the VPC. `VpcConfig` is used in hosting services and in batch transform. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud and Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

Response Syntax

```
{
    "ModelArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelArn (p. 650)

The ARN of the model created in Amazon SageMaker.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateModelPackage

Service: Amazon SageMaker Service

Creates a model package that you can use to create Amazon SageMaker models or list on AWS Marketplace. Buyers can subscribe to model packages listed on AWS Marketplace to create models in Amazon SageMaker.

To create a model package by specifying a Docker container that contains your inference code and the Amazon S3 location of your model artifacts, provide values for `InferenceSpecification`. To create a model from an algorithm resource that you created or subscribed to in AWS Marketplace, provide a value for `SourceAlgorithmSpecification`.

Request Syntax

```
{
    "CertifyForMarketplace": boolean,
    "InferenceSpecification": {
        "Containers": [
            {
                "ContainerHostname": "string",
                "Image": "string",
                "ImageDigest": "string",
                "ModelDataUrl": "string",
                "ProductId": "string"
            }
        ],
        "SupportedContentTypes": [ "string" ],
        "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
        "SupportedResponseMIMETypes": [ "string" ],
        "SupportedTransformInstanceTypes": [ "string" ]
    },
    "ModelPackageDescription": "string",
    "ModelPackageName": "string",
    "SourceAlgorithmSpecification": {
        "SourceAlgorithms": [
            {
                "AlgorithmName": "string",
                "ModelDataUrl": "string"
            }
        ]
    },
    "ValidationSpecification": {
        "ValidationProfiles": [
            {
                "ProfileName": "string",
                "TransformJobDefinition": {
                    "BatchStrategy": "string",
                    "Environment": {
                        "string" : "string"
                    },
                    "MaxConcurrentTransforms": number,
                    "MaxPayloadInMB": number,
                    "TransformInput": {
                        "CompressionType": "string",
                        "ContentType": "string",
                        "DataSource": {
                            "S3DataSource": {
                                "S3DataType": "string",
                                "S3Uri": "string"
                            }
                        },
                        "SplitType": "string"
                    },
```

```
                "TransformOutput": {
                    "Accept": "string",
                    "AssembleWith": "string",
                    "KmsKeyId": "string",
                    "S3OutputPath": "string"
                },
                "TransformResources": {
                    "InstanceCount": number,
                    "InstanceType": "string",
                    "VolumeKmsKeyId": "string"
                }
            }
        }
    ],
    "ValidationRole": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CertifyForMarketplace (p. 652)

Whether to certify the model package for listing on AWS Marketplace.

Type: Boolean

Required: No

InferenceSpecification (p. 652)

Specifies details about inference jobs that can be run with models based on this model package, including the following:

- The Amazon ECR paths of containers that contain the inference code and model artifacts.
- The instance types that the model package supports for transform jobs and real-time endpoints used for inference.
- The input and output content formats that the model package supports for inference.

Type: InferenceSpecification (p. 929) object

Required: No

ModelPackageDescription (p. 652)

A description of the model package.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

ModelPackageName (p. 652)

The name of the model package. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

SourceAlgorithmSpecification (p. 652)

Details about the algorithm that was used to create the model package.

Type: SourceAlgorithmSpecification (p. 1003) object

Required: No

ValidationSpecification (p. 652)

Specifies configurations for one or more transform jobs that Amazon SageMaker runs to test the model package.

Type: ModelPackageValidationSpecification (p. 965) object

Required: No

Response Syntax

```
{
    "ModelPackageArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelPackageArn (p. 654)

The Amazon Resource Name (ARN) of the new model package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model-package/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateNotebookInstance

Service: Amazon SageMaker Service

Creates an Amazon SageMaker notebook instance. A notebook instance is a machine learning (ML) compute instance running on a Jupyter notebook.

In a `CreateNotebookInstance` request, specify the type of ML compute instance that you want to run. Amazon SageMaker launches the instance, installs common libraries that you can use to explore datasets for model training, and attaches an ML storage volume to the notebook instance.

Amazon SageMaker also provides a set of example notebooks. Each notebook demonstrates how to use Amazon SageMaker with a specific algorithm or with a machine learning framework.

After receiving the request, Amazon SageMaker does the following:

1. Creates a network interface in the Amazon SageMaker VPC.
2. (Option) If you specified `SubnetId`, Amazon SageMaker creates a network interface in your own VPC, which is inferred from the subnet ID that you provide in the input. When creating this network interface, Amazon SageMaker attaches the security group that you specified in the request to the network interface that it creates in your VPC.
3. Launches an EC2 instance of the type specified in the request in the Amazon SageMaker VPC. If you specified `SubnetId` of your VPC, Amazon SageMaker specifies both network interfaces when launching this instance. This enables inbound traffic from your own VPC to the notebook instance, assuming that the security groups allow it.

After creating the notebook instance, Amazon SageMaker returns its Amazon Resource Name (ARN). You can't change the name of a notebook instance after you create it.

After Amazon SageMaker creates the notebook instance, you can connect to the Jupyter server and work in Jupyter notebooks. For example, you can write code to explore a dataset that you can use for model training, train a model, host models by creating Amazon SageMaker endpoints, and validate hosted models.

For more information, see How It Works.

Request Syntax

```
{
   "AcceleratorTypes": [ "string" ],
   "AdditionalCodeRepositories": [ "string" ],
   "DefaultCodeRepository": "string",
   "DirectInternetAccess": "string",
   "InstanceType": "string",
   "KmsKeyId": "string",
   "LifecycleConfigName": "string",
   "NotebookInstanceName": "string",
   "RoleArn": "string",
   "RootAccess": "string",
   "SecurityGroupIds": [ "string" ],
   "SubnetId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ],
   "VolumeSizeInGB": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AcceleratorTypes (p. 656)

A list of Elastic Inference (EI) instance types to associate with this notebook instance. Currently, only one instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: `ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge`

Required: No

AdditionalCodeRepositories (p. 656)

An array of up to three Git repositories to associate with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

DefaultCodeRepository (p. 656)

A Git repository to associate with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

DirectInternetAccess (p. 656)

Sets whether Amazon SageMaker provides internet access to the notebook instance. If you set this to `Disabled` this notebook instance will be able to access resources only in your VPC, and will not be able to connect to Amazon SageMaker training and endpoint services unless your configure a NAT Gateway in your VPC.

For more information, see Notebook Instances Are Internet-Enabled by Default. You can set the value of this parameter to `Disabled` only if you set a value for the `SubnetId` parameter.

Type: String

Valid Values: `Enabled | Disabled`

Required: No

InstanceType (p. 656)

The type of ML compute instance to launch for the notebook instance.

Type: String

Valid Values: `ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge | ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge`

Required: Yes

KmsKeyId (p. 656)

The Amazon Resource Name (ARN) of a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to your notebook instance. The KMS key you provide must be enabled. For information, see Enabling and Disabling Keys in the *AWS Key Management Service Developer Guide*.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

LifecycleConfigName (p. 656)

The name of a lifecycle configuration to associate with the notebook instance. For information about lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

NotebookInstanceName (p. 656)

The name of the new notebook instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

RoleArn (p. 656)

When you send any requests to AWS resources from the notebook instance, Amazon SageMaker assumes this role to perform tasks on your behalf. You must grant this role necessary permissions so Amazon SageMaker can perform these tasks. The policy must allow the Amazon SageMaker service principal (sagemaker.amazonaws.com) permissionsto to assume this role. For more information, see Amazon SageMaker Roles.

> **Note**
> To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

RootAccess (p. 656)

Whether root access is enabled or disabled for users of the notebook instance. The default value is `Enabled`.

> **Note**
> Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.

Type: String

Valid Values: `Enabled | Disabled`

Required: No

SecurityGroupIds (p. 656)

The VPC security group IDs, in the form sg-xxxxxxxx. The security groups must be for the same VPC as specified in the subnet.

Type: Array of strings

Array Members: Maximum number of 5 items.

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Required: No

SubnetId (p. 656)

The ID of the subnet in a VPC to which you would like to have a connectivity from your ML compute instance.

Type: String

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Required: No

Tags (p. 656)

A list of tags to associate with the notebook instance. You can add tags later by using the `CreateTags` API.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

VolumeSizeInGB (p. 656)

The size, in GB, of the ML storage volume to attach to the notebook instance. The default value is 5 GB.

Type: Integer

Valid Range: Minimum value of 5. Maximum value of 16384.

Required: No

Response Syntax

```
{
    "NotebookInstanceArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NotebookInstanceArn (p. 660)

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateNotebookInstanceLifecycleConfig

Service: Amazon SageMaker Service

Creates a lifecycle configuration that you can associate with a notebook instance. A *lifecycle configuration* is a collection of shell scripts that run when you create or start a notebook instance.

Each lifecycle configuration script has a limit of 16384 characters.

The value of the `$PATH` environment variable that is available to both scripts is `/sbin:bin:/usr/sbin:/usr/bin`.

View CloudWatch Logs for notebook instance lifecycle configurations in log group `/aws/sagemaker/NotebookInstances` in log stream `[notebook-instance-name]/[LifecycleConfigHook]`.

Lifecycle configuration scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Request Syntax

```
{
    "NotebookInstanceLifecycleConfigName": "string",
    "OnCreate": [
        {
            "Content": "string"
        }
    ],
    "OnStart": [
        {
            "Content": "string"
        }
    ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceLifecycleConfigName (p. 662)

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

OnCreate (p. 662)

A shell script that runs only once, when you create a notebook instance. The shell script must be a base64-encoded string.

Type: Array of NotebookInstanceLifecycleHook (p. 969) objects

Array Members: Maximum number of 1 item.

Required: No

OnStart (p. 662)

A shell script that runs every time you start a notebook instance, including when you create the notebook instance. The shell script must be a base64-encoded string.

Type: Array of NotebookInstanceLifecycleHook (p. 969) objects

Array Members: Maximum number of 1 item.

Required: No

Response Syntax

```
{
    "NotebookInstanceLifecycleConfigArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NotebookInstanceLifecycleConfigArn (p. 663)

The Amazon Resource Name (ARN) of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 256.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java

- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreatePresignedNotebookInstanceUrl

Service: Amazon SageMaker Service

Returns a URL that you can use to connect to the Jupyter server from a notebook instance. In the Amazon SageMaker console, when you choose `Open` next to a notebook instance, Amazon SageMaker opens a new tab showing the Jupyter server home page from the notebook instance. The console uses this API to get the URL and show the page.

IAM authorization policies for this API are also enforced for every HTTP request and WebSocket frame that attempts to connect to the notebook instance.For example, you can restrict access to this API and to the URL that it returns to a list of IP addresses that you specify. Use the `NotIpAddress` condition operator and the `aws:SourceIP` condition context key to specify the list of IP addresses that you want to have access to the notebook instance. For more information, see Limit Access to a Notebook Instance by IP Address.

> **Note**
> The URL that you get from a call to CreatePresignedNotebookInstanceUrl (p. 665) is valid only for 5 minutes. If you try to use the URL after the 5-minute limit expires, you are directed to the AWS console sign-in page.

Request Syntax

```
{
   "NotebookInstanceName": "string",
   "SessionExpirationDurationInSeconds": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 665)

The name of the notebook instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

SessionExpirationDurationInSeconds (p. 665)

The duration of the session, in seconds. The default is 12 hours.

Type: Integer

Valid Range: Minimum value of 1800. Maximum value of 43200.

Required: No

Response Syntax

```
{
```

```
    "AuthorizedUrl": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AuthorizedUrl (p. 665)

A JSON object that contains the URL string.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateTrainingJob

Service: Amazon SageMaker Service

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- `AlgorithmSpecification` - Identifies the training algorithm to use.
- `HyperParameters` - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- `InputDataConfig` - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- `OutputDataConfig` - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.

- `ResourceConfig` - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- `EnableManagedSpotTraining` - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.
- `RoleARN` - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- `StoppingCondition` - To help cap training costs, use `MaxRuntimeInSeconds` to set a time limit for training. Use `MaxWaitTimeInSeconds` to specify how long you are willing to to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see How It Works.

Request Syntax

```
{
   "AlgorithmSpecification": {
      "AlgorithmName": "string",
      "MetricDefinitions": [
         {
            "Name": "string",
            "Regex": "string"
         }
      ],
      "TrainingImage": "string",
      "TrainingInputMode": "string"
   },
   "CheckpointConfig": {
      "LocalPath": "string",
      "S3Uri": "string"
   },
   "EnableInterContainerTrafficEncryption": boolean,
   "EnableManagedSpotTraining": boolean,
   "EnableNetworkIsolation": boolean,
   "HyperParameters": {
```

```
        "string" : "string"
    },
    "InputDataConfig": [
        {
            "ChannelName": "string",
            "CompressionType": "string",
            "ContentType": "string",
            "DataSource": {
                "FileSystemDataSource": {
                    "DirectoryPath": "string",
                    "FileSystemAccessMode": "string",
                    "FileSystemId": "string",
                    "FileSystemType": "string"
                },
                "S3DataSource": {
                    "AttributeNames": [ "string" ],
                    "S3DataDistributionType": "string",
                    "S3DataType": "string",
                    "S3Uri": "string"
                }
            },
            "InputMode": "string",
            "RecordWrapperType": "string",
            "ShuffleConfig": {
                "Seed": number
            }
        }
    ],
    "OutputDataConfig": {
        "KmsKeyId": "string",
        "S3OutputPath": "string"
    },
    "ResourceConfig": {
        "InstanceCount": number,
        "InstanceType": "string",
        "VolumeKmsKeyId": "string",
        "VolumeSizeInGB": number
    },
    "RoleArn": "string",
    "StoppingCondition": {
        "MaxRuntimeInSeconds": number,
        "MaxWaitTimeInSeconds": number
    },
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "TrainingJobName": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "Subnets": [ "string" ]
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 667)

The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 863) object

Required: Yes

CheckpointConfig (p. 667)

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 880) object

Required: No

EnableInterContainerTrafficEncryption (p. 667)

To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.

Type: Boolean

Required: No

EnableManagedSpotTraining (p. 667)

To train models using managed spot training, choose `True`. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. this option is useful when training jobs can be interrupted and when there is flexibility when the training job is run.

The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No

EnableNetworkIsolation (p. 667)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

> **Note**
> The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 667)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the `Length Constraint`.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

InputDataConfig (p. 667)

An array of `Channel` objects. Each channel is a named input source. `InputDataConfig` describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, `training_data` and `validation_data`. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

OutputDataConfig (p. 667)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 976) object

Required: Yes

ResourceConfig (p. 667)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose `File` as the `TrainingInputMode` in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 991) object

Required: Yes

RoleArn (p. 667)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

> **Note**
> To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StoppingCondition (p. 667)

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1004) object

Required: Yes

Tags (p. 667)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobName (p. 667)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

VpcConfig (p. 667)

A VpcConfig (p. 1039) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

Response Syntax

```
{
    "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 672)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateTransformJob

Service: Amazon SageMaker Service

Starts a transform job. A transform job uses a trained model to get inferences on a dataset and saves these results to an Amazon S3 location that you specify.

To perform batch transformations, you create a transform job and use the data that you have readily available.

In the request body, you provide the following:

- `TransformJobName` - Identifies the transform job. The name must be unique within an AWS Region in an AWS account.
- `ModelName` - Identifies the model to use. `ModelName` must be the name of an existing Amazon SageMaker model in the same AWS Region and AWS account. For information on creating a model, see CreateModel (p. 648).
- `TransformInput` - Describes the dataset to be transformed and the Amazon S3 location where it is stored.
- `TransformOutput` - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.
- `TransformResources` - Identifies the ML compute instances for the transform job.

For more information about how batch transformation works Amazon SageMaker, see How It Works.

Request Syntax

```
{
    "BatchStrategy": "string",
    "DataProcessing": {
        "InputFilter": "string",
        "JoinSource": "string",
        "OutputFilter": "string"
    },
    "Environment": {
        "string" : "string"
    },
    "MaxConcurrentTransforms": number,
    "MaxPayloadInMB": number,
    "ModelName": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "TransformInput": {
        "CompressionType": "string",
        "ContentType": "string",
        "DataSource": {
            "S3DataSource": {
                "S3DataType": "string",
                "S3Uri": "string"
            }
        },
        "SplitType": "string"
    },
    "TransformJobName": "string",
    "TransformOutput": {
        "Accept": "string",
        "AssembleWith": "string",
```

```
        "KmsKeyId": "string",
        "S3OutputPath": "string"
    },
    "TransformResources": {
        "InstanceCount": number,
        "InstanceType": "string",
        "VolumeKmsKeyId": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

BatchStrategy (p. 673)

Specifies the number of records to include in a mini-batch for an HTTP inference request. A *record* is a single unit of input data that inference can be made on. For example, a single line in a CSV file is a record.

To enable the batch strategy, you must set `SplitType` to `Line`, `RecordIO`, or `TFRecord`.

To use only one record when making an HTTP invocation request to a container, set `BatchStrategy` to `SingleRecord` and `SplitType` to `Line`.

To fit as many records in a mini-batch as can fit within the `MaxPayloadInMB` limit, set `BatchStrategy` to `MultiRecord` and `SplitType` to `Line`.

Type: String

Valid Values: `MultiRecord | SingleRecord`

Required: No

DataProcessing (p. 673)

The data structure used to specify the data to be used for inference in a batch transform job and to associate the data that is relevant to the prediction results in the output. The input filter provided allows you to exclude input data that is not needed for inference in a batch transform job. The output filter provided allows you to include input data relevant to interpreting the predictions in the output from the job. For more information, see Associate Prediction Results with their Corresponding Input Records.

Type: DataProcessing (p. 891) object

Required: No

Environment (p. 673)

The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: `[a-zA-Z_][a-zA-Z0-9_]*`

Value Length Constraints: Maximum length of 10240.

Value Pattern: `[\S\s]*`

Required: No

MaxConcurrentTransforms (p. 673)

The maximum number of parallel requests that can be sent to each instance in a transform job. If `MaxConcurrentTransforms` is set to `0` or left unset, Amazon SageMaker checks the optional execution-parameters to determine the optimal settings for your chosen algorithm. If the execution-parameters endpoint is not enabled, the default value is `1`. For more information on execution-parameters, see How Containers Serve Requests. For built-in algorithms, you don't need to set a value for `MaxConcurrentTransforms`.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

MaxPayloadInMB (p. 673)

The maximum allowed size of the payload, in MB. A *payload* is the data portion of a record (without metadata). The value in `MaxPayloadInMB` must be greater than, or equal to, the size of a single record. To estimate the size of a record in MB, divide the size of your dataset by the number of records. To ensure that the records fit within the maximum payload size, we recommend using a slightly larger value. The default value is `6` MB.

For cases where the payload might be arbitrarily large and is transmitted using HTTP chunked encoding, set the value to `0`. This feature works only in supported algorithms. Currently, Amazon SageMaker built-in algorithms do not support HTTP chunked encoding.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

ModelName (p. 673)

The name of the model that you want to use for the transform job. `ModelName` must be the name of an existing Amazon SageMaker model within an AWS Region in an AWS account.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Tags (p. 673)

(Optional) An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TransformInput (p. 673)

Describes the input source and the way the transform job consumes it.

Type: TransformInput (p. 1024) object

Required: Yes

TransformJobName (p. 673)

The name of the transform job. The name must be unique within an AWS Region in an AWS account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

TransformOutput (p. 673)

Describes the results of the transform job.

Type: TransformOutput (p. 1030) object

Required: Yes

TransformResources (p. 673)

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: TransformResources (p. 1032) object

Required: Yes

Response Syntax

```
{
    "TransformJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TransformJobArn (p. 676)

The Amazon Resource Name (ARN) of the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

CreateWorkteam

Service: Amazon SageMaker Service

Creates a new work team for labeling your data. A work team is defined by one or more Amazon Cognito user pools. You must first create the user pools before you can create a work team.

You cannot create more than 25 work teams in an account and region.

Request Syntax

```
{
    "Description": "string",
    "MemberDefinitions": [
        {
            "CognitoMemberDefinition": {
                "ClientId": "string",
                "UserGroup": "string",
                "UserPool": "string"
            }
        }
    ],
    "NotificationConfiguration": {
        "NotificationTopicArn": "string"
    },
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

Description (p. 678)

A description of the work team.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `.+`

Required: Yes

MemberDefinitions (p. 678)

A list of `MemberDefinition` objects that contains objects that identify the Amazon Cognito user pool that makes up the work team. For more information, see Amazon Cognito User Pools.

All of the `CognitoMemberDefinition` objects that make up the member definition must have the same `ClientId` and `UserPool` values.

Type: Array of MemberDefinition (p. 954) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: Yes

NotificationConfiguration (p. 678)

Configures notification of workers regarding available or expiring work items.

Type: NotificationConfiguration (p. 973) object

Required: No

Tags (p. 678)

An array of key-value pairs.

For more information, see Resource Tag and Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

WorkteamName (p. 678)

The name of the work team. Use this name to identify the work team.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```
{
    "WorkteamArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

WorkteamArn (p. 679)

The Amazon Resource Name (ARN) of the work team. You can use this ARN to identify the work team.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteAlgorithm

Service: Amazon SageMaker Service

Removes the specified algorithm from your account.

Request Syntax

```
{
    "AlgorithmName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmName (p. 681)

The name of the algorithm to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteCodeRepository

Service: Amazon SageMaker Service

Deletes the specified Git repository from your account.

Request Syntax

```
{
    "CodeRepositoryName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CodeRepositoryName (p. 682)

>   The name of the Git repository to delete.

>   Type: String

>   Length Constraints: Minimum length of 1. Maximum length of 63.

>   Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

>   Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteEndpoint

Service: Amazon SageMaker Service

Deletes an endpoint. Amazon SageMaker frees up all of the resources that were deployed when the endpoint was created.

Amazon SageMaker retires any custom KMS key grants associated with the endpoint, meaning you don't need to use the RevokeGrant API call.

Request Syntax

```
{
    "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

EndpointName (p. 683)

> The name of the endpoint that you want to delete.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python

- [AWS SDK for Ruby V2](#)

DeleteEndpointConfig

Service: Amazon SageMaker Service

Deletes an endpoint configuration. The `DeleteEndpointConfig` API deletes only the specified configuration. It does not delete endpoints created using the configuration.

Request Syntax

```
{
    "EndpointConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

EndpointConfigName (p. 685)

The name of the endpoint configuration that you want to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteModel

Service: Amazon SageMaker Service

Deletes a model. The `DeleteModel` API deletes only the model entry that was created in Amazon SageMaker when you called the CreateModel API. It does not delete model artifacts, inference code, or the IAM role that you specified when creating the model.

Request Syntax

```
{
    "ModelName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

ModelName (p. 686)

> The name of the model to delete.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
> Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteModelPackage

Service: Amazon SageMaker Service

Deletes a model package.

A model package is used to create Amazon SageMaker models or list on AWS Marketplace. Buyers can subscribe to model packages listed on AWS Marketplace to create models in Amazon SageMaker.

Request Syntax

```
{
    "ModelPackageName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

ModelPackageName (p. 688)

The name of the model package. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python

- [AWS SDK for Ruby V2](#)

DeleteNotebookInstance

Service: Amazon SageMaker Service

Deletes an Amazon SageMaker notebook instance. Before you can delete a notebook instance, you must call the `StopNotebookInstance` API.

> **Important**
> When you delete a notebook instance, you lose all of your data. Amazon SageMaker removes the ML compute instance, and deletes the ML storage volume and the network interface associated with the notebook instance.

Request Syntax

```
{
    "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 690)

The name of the Amazon SageMaker notebook instance to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteNotebookInstanceLifecycleConfig

Service: Amazon SageMaker Service

Deletes a notebook instance lifecycle configuration.

Request Syntax

```
{
    "NotebookInstanceLifecycleConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceLifecycleConfigName (p. 692)

> The name of the lifecycle configuration to delete.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteTags

Service: Amazon SageMaker Service

Deletes the specified tags from an Amazon SageMaker resource.

To list a resource's tags, use the `ListTags` API.

> **Note**
> When you call this API to delete tags from a hyperparameter tuning job, the deleted tags are not removed from training jobs that the hyperparameter tuning job launched before you called this API.

Request Syntax

```
{
   "ResourceArn": "string",
   "TagKeys": [ "string" ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

ResourceArn (p. 693)

The Amazon Resource Name (ARN) of the resource whose tags you want to delete.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:.*`

Required: Yes

TagKeys (p. 693)

An array or one or more tag keys to delete.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 50 items.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-@]*)$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteWorkteam

Service: Amazon SageMaker Service

Deletes an existing work team. This operation can't be undone.

Request Syntax

```
{
    "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

WorkteamName (p. 695)

>   The name of the work team to delete.
>
>   Type: String
>
>   Length Constraints: Minimum length of 1. Maximum length of 63.
>
>   Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
>   Required: Yes

Response Syntax

```
{
    "Success": boolean
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Success (p. 695)

>   Returns `true` if the work team was successfully deleted; otherwise, returns `false`.
>
>   Type: Boolean

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

>   You have exceeded an Amazon SageMaker resource limit. For example, you might have too many
>   training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeAlgorithm

Service: Amazon SageMaker Service

Returns a description of the specified algorithm that is in your account.

Request Syntax

```
{
    "AlgorithmName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AlgorithmName (p. 697)

> The name of the algorithm to describe.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 170.
>
> Pattern: `(arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\/)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?<!-)$`
>
> Required: Yes

Response Syntax

```
{
    "AlgorithmArn": "string",
    "AlgorithmDescription": "string",
    "AlgorithmName": "string",
    "AlgorithmStatus": "string",
    "AlgorithmStatusDetails": {
        "ImageScanStatuses": [
            {
                "FailureReason": "string",
                "Name": "string",
                "Status": "string"
            }
        ],
        "ValidationStatuses": [
            {
                "FailureReason": "string",
                "Name": "string",
                "Status": "string"
            }
        ]
    },
    "CertifyForMarketplace": boolean,
    "CreationTime": number,
    "InferenceSpecification": {
        "Containers": [
            {
                "ContainerHostname": "string",
```

```
                    "Image": "string",
                    "ImageDigest": "string",
                    "ModelDataUrl": "string",
                    "ProductId": "string"
                }
            ],
            "SupportedContentTypes": [ "string" ],
            "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
            "SupportedResponseMIMETypes": [ "string" ],
            "SupportedTransformInstanceTypes": [ "string" ]
        },
        "ProductId": "string",
        "TrainingSpecification": {
            "MetricDefinitions": [
                {
                    "Name": "string",
                    "Regex": "string"
                }
            ],
            "SupportedHyperParameters": [
                {
                    "DefaultValue": "string",
                    "Description": "string",
                    "IsRequired": boolean,
                    "IsTunable": boolean,
                    "Name": "string",
                    "Range": {
                        "CategoricalParameterRangeSpecification": {
                            "Values": [ "string" ]
                        },
                        "ContinuousParameterRangeSpecification": {
                            "MaxValue": "string",
                            "MinValue": "string"
                        },
                        "IntegerParameterRangeSpecification": {
                            "MaxValue": "string",
                            "MinValue": "string"
                        }
                    },
                    "Type": "string"
                }
            ],
            "SupportedTrainingInstanceTypes": [ "string" ],
            "SupportedTuningJobObjectiveMetrics": [
                {
                    "MetricName": "string",
                    "Type": "string"
                }
            ],
            "SupportsDistributedTraining": boolean,
            "TrainingChannels": [
                {
                    "Description": "string",
                    "IsRequired": boolean,
                    "Name": "string",
                    "SupportedCompressionTypes": [ "string" ],
                    "SupportedContentTypes": [ "string" ],
                    "SupportedInputModes": [ "string" ]
                }
            ],
            "TrainingImage": "string",
            "TrainingImageDigest": "string"
        },
        "ValidationSpecification": {
            "ValidationProfiles": [
                {
```

```
            "ProfileName": "string",
            "TrainingJobDefinition": {
                "HyperParameters": {
                    "string" : "string"
                },
                "InputDataConfig": [
                    {
                        "ChannelName": "string",
                        "CompressionType": "string",
                        "ContentType": "string",
                        "DataSource": {
                            "FileSystemDataSource": {
                                "DirectoryPath": "string",
                                "FileSystemAccessMode": "string",
                                "FileSystemId": "string",
                                "FileSystemType": "string"
                            },
                            "S3DataSource": {
                                "AttributeNames": [ "string" ],
                                "S3DataDistributionType": "string",
                                "S3DataType": "string",
                                "S3Uri": "string"
                            }
                        },
                        "InputMode": "string",
                        "RecordWrapperType": "string",
                        "ShuffleConfig": {
                            "Seed": number
                        }
                    }
                ],
                "OutputDataConfig": {
                    "KmsKeyId": "string",
                    "S3OutputPath": "string"
                },
                "ResourceConfig": {
                    "InstanceCount": number,
                    "InstanceType": "string",
                    "VolumeKmsKeyId": "string",
                    "VolumeSizeInGB": number
                },
                "StoppingCondition": {
                    "MaxRuntimeInSeconds": number,
                    "MaxWaitTimeInSeconds": number
                },
                "TrainingInputMode": "string"
            },
            "TransformJobDefinition": {
                "BatchStrategy": "string",
                "Environment": {
                    "string" : "string"
                },
                "MaxConcurrentTransforms": number,
                "MaxPayloadInMB": number,
                "TransformInput": {
                    "CompressionType": "string",
                    "ContentType": "string",
                    "DataSource": {
                        "S3DataSource": {
                            "S3DataType": "string",
                            "S3Uri": "string"
                        }
                    },
                    "SplitType": "string"
                },
                "TransformOutput": {
```

```
                "Accept": "string",
                "AssembleWith": "string",
                "KmsKeyId": "string",
                "S3OutputPath": "string"
            },
            "TransformResources": {
                "InstanceCount": number,
                "InstanceType": "string",
                "VolumeKmsKeyId": "string"
            }
        }
    }
    ],
    "ValidationRole": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AlgorithmArn (p. 697)

The Amazon Resource Name (ARN) of the algorithm.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:algorithm/.*`

AlgorithmDescription (p. 697)

A brief summary about the algorithm.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

AlgorithmName (p. 697)

The name of the algorithm being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

AlgorithmStatus (p. 697)

The current status of the algorithm.

Type: String

Valid Values: `Pending | InProgress | Completed | Failed | Deleting`

AlgorithmStatusDetails (p. 697)

Details about the current status of the algorithm.

Type: AlgorithmStatusDetails (p. 865) object

CertifyForMarketplace (p. 697)

Whether the algorithm is certified to be listed in AWS Marketplace.

Type: Boolean

CreationTime (p. 697)

A timestamp specifying when the algorithm was created.

Type: Timestamp

InferenceSpecification (p. 697)

Details about inference jobs that the algorithm runs.

Type: InferenceSpecification (p. 929) object

ProductId (p. 697)

The product identifier of the algorithm.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

TrainingSpecification (p. 697)

Details about training jobs run by this algorithm.

Type: TrainingSpecification (p. 1021) object

ValidationSpecification (p. 697)

Details about configurations for one or more training jobs that Amazon SageMaker runs to test the algorithm.

Type: AlgorithmValidationSpecification (p. 870) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeCodeRepository

Service: Amazon SageMaker Service

Gets details about the specified Git repository.

Request Syntax

```
{
    "CodeRepositoryName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CodeRepositoryName (p. 703)

>   The name of the Git repository to describe.
>
>   Type: String
>
>   Length Constraints: Minimum length of 1. Maximum length of 63.
>
>   Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
>
>   Required: Yes

Response Syntax

```
{
    "CodeRepositoryArn": "string",
    "CodeRepositoryName": "string",
    "CreationTime": number,
    "GitConfig": {
        "Branch": "string",
        "RepositoryUrl": "string",
        "SecretArn": "string"
    },
    "LastModifiedTime": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeRepositoryArn (p. 703)

>   The Amazon Resource Name (ARN) of the Git repository.
>
>   Type: String
>
>   Length Constraints: Minimum length of 1. Maximum length of 2048.
>
>   Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*

CodeRepositoryName (p. 703)

> The name of the Git repository.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

CreationTime (p. 703)

> The date and time that the repository was created.
>
> Type: Timestamp

GitConfig (p. 703)

> Configuration details about the repository, including the URL where the repository is located, the default branch, and the Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the repository.
>
> Type: GitConfig (p. 905) object

LastModifiedTime (p. 703)

> The date and time that the repository was last changed.
>
> Type: Timestamp

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeCompilationJob

Service: Amazon SageMaker Service

Returns information about a model compilation job.

To create a model compilation job, use CreateCompilationJob (p. 629). To get information about multiple model compilation jobs, use ListCompilationJobs (p. 767).

Request Syntax

```
{
    "CompilationJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CompilationJobName (p. 705)

The name of the model compilation job that you want information about.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

Response Syntax

```
{
    "CompilationEndTime": number,
    "CompilationJobArn": "string",
    "CompilationJobName": "string",
    "CompilationJobStatus": "string",
    "CompilationStartTime": number,
    "CreationTime": number,
    "FailureReason": "string",
    "InputConfig": {
        "DataInputConfig": "string",
        "Framework": "string",
        "S3Uri": "string"
    },
    "LastModifiedTime": number,
    "ModelArtifacts": {
        "S3ModelArtifacts": "string"
    },
    "OutputConfig": {
        "S3OutputLocation": "string",
        "TargetDevice": "string"
    },
    "RoleArn": "string",
    "StoppingCondition": {
        "MaxRuntimeInSeconds": number,
        "MaxWaitTimeInSeconds": number
```

```
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CompilationEndTime (p. 705)

The time when the model compilation job on a compilation job instance ended. For a successful or stopped job, this is when the job's model artifacts have finished uploading. For a failed job, this is when Amazon SageMaker detected that the job failed.

Type: Timestamp

CompilationJobArn (p. 705)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker assumes to perform the model compilation job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*`

CompilationJobName (p. 705)

The name of the model compilation job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

CompilationJobStatus (p. 705)

The status of the model compilation job.

Type: String

Valid Values: `INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED`

CompilationStartTime (p. 705)

The time when the model compilation job started the `CompilationJob` instances.

You are billed for the time between this timestamp and the timestamp in the DescribeCompilationJob:CompilationEndTime (p. 706) field. In Amazon CloudWatch Logs, the start time might be later than this time. That's because it takes time to download the compilation job, which depends on the size of the compilation job container.

Type: Timestamp

CreationTime (p. 705)

The time that the model compilation job was created.

Type: Timestamp

FailureReason (p. 705)

If a model compilation job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

InputConfig (p. 705)

Information about the location in Amazon S3 of the input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Type: InputConfig (p. 931) object

LastModifiedTime (p. 705)

The time that the status of the model compilation job was last modified.

Type: Timestamp

ModelArtifacts (p. 705)

Information about the location in Amazon S3 that has been configured for storing the model artifacts used in the compilation job.

Type: ModelArtifacts (p. 957) object

OutputConfig (p. 705)

Information about the output location for the compiled model and the target device that the model runs on.

Type: OutputConfig (p. 975) object

RoleArn (p. 705)

The Amazon Resource Name (ARN) of the model compilation job.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

StoppingCondition (p. 705)

Specifies a limit to how long a model compilation job can run. When the job reaches the time limit, Amazon SageMaker ends the compilation job. Use this API to cap model training costs.

Type: StoppingCondition (p. 1004) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeEndpoint

Service: Amazon SageMaker Service

Returns the description of an endpoint.

Request Syntax

```
{
    "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

EndpointName (p. 709)

>   The name of the endpoint.
>
>   Type: String
>
>   Length Constraints: Maximum length of 63.
>
>   Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
>   Required: Yes

Response Syntax

```
{
    "CreationTime": number,
    "EndpointArn": "string",
    "EndpointConfigName": "string",
    "EndpointName": "string",
    "EndpointStatus": "string",
    "FailureReason": "string",
    "LastModifiedTime": number,
    "ProductionVariants": [
        {
            "CurrentInstanceCount": number,
            "CurrentWeight": number,
            "DeployedImages": [
                {
                    "ResolutionTime": number,
                    "ResolvedImage": "string",
                    "SpecifiedImage": "string"
                }
            ],
            "DesiredInstanceCount": number,
            "DesiredWeight": number,
            "VariantName": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreationTime (p. 709)

A timestamp that shows when the endpoint was created.

Type: Timestamp

EndpointArn (p. 709)

The Amazon Resource Name (ARN) of the endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*`

EndpointConfigName (p. 709)

The name of the endpoint configuration associated with this endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

EndpointName (p. 709)

Name of the endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

EndpointStatus (p. 709)

The status of the endpoint.

- `OutOfService`: Endpoint is not available to take incoming requests.
- `Creating`: CreateEndpoint (p. 632) is executing.
- `Updating`: UpdateEndpoint (p. 840) or UpdateEndpointWeightsAndCapacities (p. 842) is executing.
- `SystemUpdating`: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This maintenance operation does not change any customer-specified values such as VPC config, KMS encryption, model, instance type, or instance count.
- `RollingBack`: Endpoint fails to scale up or down or change its variant weight and is in the process of rolling back to its previous configuration. Once the rollback completes, endpoint returns to an `InService` status. This transitional status only applies to an endpoint that has autoscaling enabled and is undergoing variant weight or capacity changes as part of an UpdateEndpointWeightsAndCapacities (p. 842) call or when the UpdateEndpointWeightsAndCapacities (p. 842) operation is called explicitly.
- `InService`: Endpoint is available to process incoming requests.
- `Deleting`: DeleteEndpoint (p. 683) is executing.
- `Failed`: Endpoint could not be created, updated, or re-scaled. Use DescribeEndpoint:FailureReason (p. 711) for information about the failure. DeleteEndpoint (p. 683) is the only operation that can be performed on a failed endpoint.

Type: String

Valid Values: `OutOfService` | `Creating` | `Updating` | `SystemUpdating` | `RollingBack`
| `InService` | `Deleting` | `Failed`

FailureReason (p. 709)

If the status of the endpoint is `Failed`, the reason why it failed.

Type: String

Length Constraints: Maximum length of 1024.

LastModifiedTime (p. 709)

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

ProductionVariants (p. 709)

An array of ProductionVariantSummary (p. 983) objects, one for each model hosted behind this
endpoint.

Type: Array of ProductionVariantSummary (p. 983) objects

Array Members: Minimum number of 1 item.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeEndpointConfig

Service: Amazon SageMaker Service

Returns the description of an endpoint configuration created using the `CreateEndpointConfig` API.

Request Syntax

```
{
    "EndpointConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

EndpointConfigName (p. 712)

> The name of the endpoint configuration.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

Response Syntax

```
{
    "CreationTime": number,
    "EndpointConfigArn": "string",
    "EndpointConfigName": "string",
    "KmsKeyId": "string",
    "ProductionVariants": [
        {
            "AcceleratorType": "string",
            "InitialInstanceCount": number,
            "InitialVariantWeight": number,
            "InstanceType": "string",
            "ModelName": "string",
            "VariantName": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreationTime (p. 712)

> A timestamp that shows when the endpoint configuration was created.

Type: Timestamp

EndpointConfigArn (p. 712)

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint-config/.*`

EndpointConfigName (p. 712)

Name of the Amazon SageMaker endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

KmsKeyId (p. 712)

AWS KMS key ID Amazon SageMaker uses to encrypt data when storing it on the ML storage volume attached to the instance.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

ProductionVariants (p. 712)

An array of `ProductionVariant` objects, one for each model that you want to host at this endpoint.

Type: Array of ProductionVariant (p. 981) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeHyperParameterTuningJob

Service: Amazon SageMaker Service

Gets a description of a hyperparameter tuning job.

Request Syntax

```
{
    "HyperParameterTuningJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 715)

> The name of the tuning job to describe.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 32.
>
> Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
> Required: Yes

Response Syntax

```
{
    "BestTrainingJob": {
        "CreationTime": number,
        "FailureReason": "string",
        "FinalHyperParameterTuningJobObjectiveMetric": {
            "MetricName": "string",
            "Type": "string",
            "Value": number
        },
        "ObjectiveStatus": "string",
        "TrainingEndTime": number,
        "TrainingJobArn": "string",
        "TrainingJobName": "string",
        "TrainingJobStatus": "string",
        "TrainingStartTime": number,
        "TunedHyperParameters": {
            "string" : "string"
        },
        "TuningJobName": "string"
    },
    "CreationTime": number,
    "FailureReason": "string",
    "HyperParameterTuningEndTime": number,
    "HyperParameterTuningJobArn": "string",
    "HyperParameterTuningJobConfig": {
        "HyperParameterTuningJobObjective": {
            "MetricName": "string",
            "Type": "string"
```

```
        },
        "ParameterRanges": {
            "CategoricalParameterRanges": [
                {
                    "Name": "string",
                    "Values": [ "string" ]
                }
            ],
            "ContinuousParameterRanges": [
                {
                    "MaxValue": "string",
                    "MinValue": "string",
                    "Name": "string",
                    "ScalingType": "string"
                }
            ],
            "IntegerParameterRanges": [
                {
                    "MaxValue": "string",
                    "MinValue": "string",
                    "Name": "string",
                    "ScalingType": "string"
                }
            ]
        },
        "ResourceLimits": {
            "MaxNumberOfTrainingJobs": number,
            "MaxParallelTrainingJobs": number
        },
        "Strategy": "string",
        "TrainingJobEarlyStoppingType": "string"
    },
    "HyperParameterTuningJobName": "string",
    "HyperParameterTuningJobStatus": "string",
    "LastModifiedTime": number,
    "ObjectiveStatusCounters": {
        "Failed": number,
        "Pending": number,
        "Succeeded": number
    },
    "OverallBestTrainingJob": {
        "CreationTime": number,
        "FailureReason": "string",
        "FinalHyperParameterTuningJobObjectiveMetric": {
            "MetricName": "string",
            "Type": "string",
            "Value": number
        },
        "ObjectiveStatus": "string",
        "TrainingEndTime": number,
        "TrainingJobArn": "string",
        "TrainingJobName": "string",
        "TrainingJobStatus": "string",
        "TrainingStartTime": number,
        "TunedHyperParameters": {
            "string" : "string"
        },
        "TuningJobName": "string"
    },
    "TrainingJobDefinition": {
        "AlgorithmSpecification": {
            "AlgorithmName": "string",
            "MetricDefinitions": [
                {
                    "Name": "string",
                    "Regex": "string"
```

```
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
    },
    "EnableInterContainerTrafficEncryption": boolean,
    "EnableManagedSpotTraining": boolean,
    "EnableNetworkIsolation": boolean,
    "InputDataConfig": [
        {
            "ChannelName": "string",
            "CompressionType": "string",
            "ContentType": "string",
            "DataSource": {
                "FileSystemDataSource": {
                    "DirectoryPath": "string",
                    "FileSystemAccessMode": "string",
                    "FileSystemId": "string",
                    "FileSystemType": "string"
                },
                "S3DataSource": {
                    "AttributeNames": [ "string" ],
                    "S3DataDistributionType": "string",
                    "S3DataType": "string",
                    "S3Uri": "string"
                }
            },
            "InputMode": "string",
            "RecordWrapperType": "string",
            "ShuffleConfig": {
                "Seed": number
            }
        }
    ],
    "OutputDataConfig": {
        "KmsKeyId": "string",
        "S3OutputPath": "string"
    },
    "ResourceConfig": {
        "InstanceCount": number,
        "InstanceType": "string",
        "VolumeKmsKeyId": "string",
        "VolumeSizeInGB": number
    },
    "RoleArn": "string",
    "StaticHyperParameters": {
        "string" : "string"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": number,
        "MaxWaitTimeInSeconds": number
    },
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "Subnets": [ "string" ]
    }
},
"TrainingJobStatusCounters": {
    "Completed": number,
    "InProgress": number,
    "NonRetryableError": number,
    "RetryableError": number,
```

```
        "Stopped": number
    },
    "WarmStartConfig": {
        "ParentHyperParameterTuningJobs": [
            {
                "HyperParameterTuningJobName": "string"
            }
        ],
        "WarmStartType": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

BestTrainingJob (p. 715)

A TrainingJobSummary (p. 1019) object that describes the training job that completed with the best current HyperParameterTuningJobObjective (p. 924).

Type: HyperParameterTrainingJobSummary (p. 919) object

CreationTime (p. 715)

The date and time that the tuning job started.

Type: Timestamp

FailureReason (p. 715)

If the tuning job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

HyperParameterTuningEndTime (p. 715)

The date and time that the tuning job ended.

Type: Timestamp

HyperParameterTuningJobArn (p. 715)

The Amazon Resource Name (ARN) of the tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

HyperParameterTuningJobConfig (p. 715)

The HyperParameterTuningJobConfig (p. 922) object that specifies the configuration of the tuning job.

Type: HyperParameterTuningJobConfig (p. 922) object

HyperParameterTuningJobName (p. 715)

The name of the tuning job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

HyperParameterTuningJobStatus (p. 715)

The status of the tuning job: InProgress, Completed, Failed, Stopping, or Stopped.

Type: String

Valid Values: `Completed | InProgress | Failed | Stopped | Stopping`

LastModifiedTime (p. 715)

The date and time that the status of the tuning job was modified.

Type: Timestamp

ObjectiveStatusCounters (p. 715)

The ObjectiveStatusCounters (p. 974) object that specifies the number of training jobs, categorized by the status of their final objective metric, that this tuning job launched.

Type: ObjectiveStatusCounters (p. 974) object

OverallBestTrainingJob (p. 715)

If the hyperparameter tuning job is an warm start tuning job with a `WarmStartType` of `IDENTICAL_DATA_AND_ALGORITHM`, this is the TrainingJobSummary (p. 1019) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.

Type: HyperParameterTrainingJobSummary (p. 919) object

TrainingJobDefinition (p. 715)

The HyperParameterTrainingJobDefinition (p. 916) object that specifies the definition of the training jobs that this tuning job launches.

Type: HyperParameterTrainingJobDefinition (p. 916) object

TrainingJobStatusCounters (p. 715)

The TrainingJobStatusCounters (p. 1017) object that specifies the number of training jobs, categorized by status, that this tuning job launched.

Type: TrainingJobStatusCounters (p. 1017) object

WarmStartConfig (p. 715)

The configuration for starting the hyperparameter parameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.

Type: HyperParameterTuningJobWarmStartConfig (p. 927) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeLabelingJob

Service: Amazon SageMaker Service

Gets information about a labeling job.

Request Syntax

```
{
    "LabelingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

LabelingJobName (p. 721)

>   The name of the labeling job to return information for.
>
>   Type: String
>
>   Length Constraints: Minimum length of 1. Maximum length of 63.
>
>   Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
>   Required: Yes

Response Syntax

```
{
    "CreationTime": number,
    "FailureReason": "string",
    "HumanTaskConfig": {
        "AnnotationConsolidationConfig": {
            "AnnotationConsolidationLambdaArn": "string"
        },
        "MaxConcurrentTaskCount": number,
        "NumberOfHumanWorkersPerDataObject": number,
        "PreHumanTaskLambdaArn": "string",
        "PublicWorkforceTaskPrice": {
            "AmountInUsd": {
                "Cents": number,
                "Dollars": number,
                "TenthFractionsOfACent": number
            }
        },
        "TaskAvailabilityLifetimeInSeconds": number,
        "TaskDescription": "string",
        "TaskKeywords": [ "string" ],
        "TaskTimeLimitInSeconds": number,
        "TaskTitle": "string",
        "UiConfig": {
            "UiTemplateS3Uri": "string"
        },
        "WorkteamArn": "string"
    },
    "InputConfig": {
        "DataAttributes": {
```

```
            "ContentClassifiers": [ "string" ]
        },
        "DataSource": {
            "S3DataSource": {
                "ManifestS3Uri": "string"
            }
        }
    },
    "JobReferenceCode": "string",
    "LabelAttributeName": "string",
    "LabelCategoryConfigS3Uri": "string",
    "LabelCounters": {
        "FailedNonRetryableError": number,
        "HumanLabeled": number,
        "MachineLabeled": number,
        "TotalLabeled": number,
        "Unlabeled": number
    },
    "LabelingJobAlgorithmsConfig": {
        "InitialActiveLearningModelArn": "string",
        "LabelingJobAlgorithmSpecificationArn": "string",
        "LabelingJobResourceConfig": {
            "VolumeKmsKeyId": "string"
        }
    },
    "LabelingJobArn": "string",
    "LabelingJobName": "string",
    "LabelingJobOutput": {
        "FinalActiveLearningModelArn": "string",
        "OutputDatasetS3Uri": "string"
    },
    "LabelingJobStatus": "string",
    "LastModifiedTime": number,
    "OutputConfig": {
        "KmsKeyId": "string",
        "S3OutputPath": "string"
    },
    "RoleArn": "string",
    "StoppingConditions": {
        "MaxHumanLabeledObjectCount": number,
        "MaxPercentageOfInputDatasetLabeled": number
    },
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreationTime (p. 721)

> The date and time that the labeling job was created.

> Type: Timestamp

FailureReason (p. 721)

> If the job failed, the reason that it failed.

Type: String

Length Constraints: Maximum length of 1024.

HumanTaskConfig (p. 721)

Configuration information required for human workers to complete a labeling task.

Type: HumanTaskConfig (p. 907) object

InputConfig (p. 721)

Input configuration information for the labeling job, such as the Amazon S3 location of the data objects and the location of the manifest file that describes the data objects.

Type: LabelingJobInputConfig (p. 945) object

JobReferenceCode (p. 721)

A unique identifier for work done as part of a labeling job.

Type: String

Length Constraints: Minimum length of 1.

Pattern: `.+`

LabelAttributeName (p. 721)

The attribute used as the label in the output manifest file.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 127.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

LabelCategoryConfigS3Uri (p. 721)

The S3 location of the JSON file that defines the categories used to label data objects.

The file is a JSON structure in the following format:

```
{

"document-version": "2018-11-28"

"labels": [

{

"label": "label 1"

},

{

"label": "label 2"

},

...

{
```

```
"label": "label n"

}

]

}
```

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

LabelCounters (p. 721)

Provides a breakdown of the number of data objects labeled by humans, the number of objects labeled by machine, the number of objects than couldn't be labeled, and the total number of objects labeled.

Type: LabelCounters (p. 936) object

LabelingJobAlgorithmsConfig (p. 721)

Configuration information for automated data labeling.

Type: LabelingJobAlgorithmsConfig (p. 939) object

LabelingJobArn (p. 721)

The Amazon Resource Name (ARN) of the labeling job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`

LabelingJobName (p. 721)

The name assigned to the labeling job when it was created.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

LabelingJobOutput (p. 721)

The location of the output produced by the labeling job.

Type: LabelingJobOutput (p. 946) object

LabelingJobStatus (p. 721)

The processing status of the labeling job.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

LastModifiedTime (p. 721)

The date and time that the labeling job was last updated.

Type: Timestamp

OutputConfig (p. 721)

The location of the job's output data and the AWS Key Management Service key ID for the key used to encrypt the output data, if any.

Type: LabelingJobOutputConfig (p. 947) object

RoleArn (p. 721)

The Amazon Resource Name (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during data labeling.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

StoppingConditions (p. 721)

A set of conditions for stopping a labeling job. If any of the conditions are met, the job is automatically stopped.

Type: LabelingJobStoppingConditions (p. 950) object

Tags (p. 721)

An array of key/value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeModel

Service: Amazon SageMaker Service

Describes a model that you created using the `CreateModel` API.

Request Syntax

```
{
    "ModelName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

ModelName (p. 727)

 The name of the model.

 Type: String

 Length Constraints: Maximum length of 63.

 Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

 Required: Yes

Response Syntax

```
{
    "Containers": [
        {
            "ContainerHostname": "string",
            "Environment": {
                "string" : "string"
            },
            "Image": "string",
            "ModelDataUrl": "string",
            "ModelPackageName": "string"
        }
    ],
    "CreationTime": number,
    "EnableNetworkIsolation": boolean,
    "ExecutionRoleArn": "string",
    "ModelArn": "string",
    "ModelName": "string",
    "PrimaryContainer": {
        "ContainerHostname": "string",
        "Environment": {
            "string" : "string"
        },
        "Image": "string",
        "ModelDataUrl": "string",
        "ModelPackageName": "string"
    },
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
```

```
        "Subnets": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Containers (p. 727)

> The containers in the inference pipeline.
>
> Type: Array of ContainerDefinition (p. 886) objects
>
> Array Members: Maximum number of 5 items.

CreationTime (p. 727)

> A timestamp that shows when the model was created.
>
> Type: Timestamp

EnableNetworkIsolation (p. 727)

> If `True`, no inbound or outbound network calls can be made to or from the model container.
>
> **Note**
> The Semantic Segmentation built-in algorithm does not support network isolation.
>
> Type: Boolean

ExecutionRoleArn (p. 727)

> The Amazon Resource Name (ARN) of the IAM role that you specified for the model.
>
> Type: String
>
> Length Constraints: Minimum length of 20. Maximum length of 2048.
>
> Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

ModelArn (p. 727)

> The Amazon Resource Name (ARN) of the model.
>
> Type: String
>
> Length Constraints: Minimum length of 20. Maximum length of 2048.
>
> Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*`

ModelName (p. 727)

> Name of the Amazon SageMaker model.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

PrimaryContainer (p. 727)

> The location of the primary inference code, associated artifacts, and custom environment map that the inference code uses when it is deployed in production.

Type: ContainerDefinition (p. 886) object

VpcConfig (p. 727)

A VpcConfig (p. 1039) object that specifies the VPC that this model has access to. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud

Type: VpcConfig (p. 1039) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeModelPackage

Service: Amazon SageMaker Service

Returns a description of the specified model package, which is used to create Amazon SageMaker models or list them on AWS Marketplace.

To create models in Amazon SageMaker, buyers can subscribe to model packages listed on AWS Marketplace.

Request Syntax

```
{
    "ModelPackageName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

ModelPackageName (p. 730)

> The name of the model package to describe.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 170.
>
> Pattern: `(arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\/)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?<!-)$`
>
> Required: Yes

Response Syntax

```
{
    "CertifyForMarketplace": boolean,
    "CreationTime": number,
    "InferenceSpecification": {
        "Containers": [
            {
                "ContainerHostname": "string",
                "Image": "string",
                "ImageDigest": "string",
                "ModelDataUrl": "string",
                "ProductId": "string"
            }
        ],
        "SupportedContentTypes": [ "string" ],
        "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
        "SupportedResponseMIMETypes": [ "string" ],
        "SupportedTransformInstanceTypes": [ "string" ]
    },
    "ModelPackageArn": "string",
    "ModelPackageDescription": "string",
    "ModelPackageName": "string",
    "ModelPackageStatus": "string",
    "ModelPackageStatusDetails": {
        "ImageScanStatuses": [
```

```
        {
            "FailureReason": "string",
            "Name": "string",
            "Status": "string"
        }
    ],
    "ValidationStatuses": [
        {
            "FailureReason": "string",
            "Name": "string",
            "Status": "string"
        }
    ]
},
"SourceAlgorithmSpecification": {
    "SourceAlgorithms": [
        {
            "AlgorithmName": "string",
            "ModelDataUrl": "string"
        }
    ]
},
"ValidationSpecification": {
    "ValidationProfiles": [
        {
            "ProfileName": "string",
            "TransformJobDefinition": {
                "BatchStrategy": "string",
                "Environment": {
                    "string" : "string"
                },
                "MaxConcurrentTransforms": number,
                "MaxPayloadInMB": number,
                "TransformInput": {
                    "CompressionType": "string",
                    "ContentType": "string",
                    "DataSource": {
                        "S3DataSource": {
                            "S3DataType": "string",
                            "S3Uri": "string"
                        }
                    },
                    "SplitType": "string"
                },
                "TransformOutput": {
                    "Accept": "string",
                    "AssembleWith": "string",
                    "KmsKeyId": "string",
                    "S3OutputPath": "string"
                },
                "TransformResources": {
                    "InstanceCount": number,
                    "InstanceType": "string",
                    "VolumeKmsKeyId": "string"
                }
            }
        }
    ],
    "ValidationRole": "string"
}
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CertifyForMarketplace (p. 730)

Whether the model package is certified for listing on AWS Marketplace.

Type: Boolean

CreationTime (p. 730)

A timestamp specifying when the model package was created.

Type: Timestamp

InferenceSpecification (p. 730)

Details about inference jobs that can be run with models based on this model package.

Type: InferenceSpecification (p. 929) object

ModelPackageArn (p. 730)

The Amazon Resource Name (ARN) of the model package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model-package/.*`

ModelPackageDescription (p. 730)

A brief summary of the model package.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

ModelPackageName (p. 730)

The name of the model package being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

ModelPackageStatus (p. 730)

The current status of the model package.

Type: String

Valid Values: `Pending | InProgress | Completed | Failed | Deleting`

ModelPackageStatusDetails (p. 730)

Details about the current status of the model package.

Type: ModelPackageStatusDetails (p. 960) object

SourceAlgorithmSpecification (p. 730)

Details about the algorithm that was used to create the model package.

Type: SourceAlgorithmSpecification (p. 1003) object

ValidationSpecification (p. 730)

Configurations for one or more transform jobs that Amazon SageMaker runs to test the model package.

Type: ModelPackageValidationSpecification (p. 965) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeNotebookInstance

Service: Amazon SageMaker Service

Returns information about a notebook instance.

Request Syntax

```
{
    "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 734)

> The name of the notebook instance that you want information about.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
> Required: Yes

Response Syntax

```
{
    "AcceleratorTypes": [ "string" ],
    "AdditionalCodeRepositories": [ "string" ],
    "CreationTime": number,
    "DefaultCodeRepository": "string",
    "DirectInternetAccess": "string",
    "FailureReason": "string",
    "InstanceType": "string",
    "KmsKeyId": "string",
    "LastModifiedTime": number,
    "NetworkInterfaceId": "string",
    "NotebookInstanceArn": "string",
    "NotebookInstanceLifecycleConfigName": "string",
    "NotebookInstanceName": "string",
    "NotebookInstanceStatus": "string",
    "RoleArn": "string",
    "RootAccess": "string",
    "SecurityGroups": [ "string" ],
    "SubnetId": "string",
    "Url": "string",
    "VolumeSizeInGB": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AcceleratorTypes (p. 734)

A list of the Elastic Inference (EI) instance types associated with this notebook instance. Currently only one EI instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: `ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge`

AdditionalCodeRepositories (p. 734)

An array of up to three Git repositories associated with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

CreationTime (p. 734)

A timestamp. Use this parameter to return the time when the notebook instance was created

Type: Timestamp

DefaultCodeRepository (p. 734)

The Git repository associated with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

DirectInternetAccess (p. 734)

Describes whether Amazon SageMaker provides internet access to the notebook instance. If this value is set to *Disabled*, the notebook instance does not have internet access, and cannot connect to Amazon SageMaker training and endpoint services.

For more information, see Notebook Instances Are Internet-Enabled by Default.

Type: String

Valid Values: `Enabled | Disabled`

FailureReason (p. 734)

If status is `Failed`, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

InstanceType (p. 734)

The type of ML compute instance running on the notebook instance.

Type: String

Valid Values: `ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge | ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge`

KmsKeyId (p. 734)

The AWS KMS key ID Amazon SageMaker uses to encrypt data when storing it on the ML storage volume attached to the instance.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

LastModifiedTime (p. 734)

A timestamp. Use this parameter to retrieve the time when the notebook instance was last modified.

Type: Timestamp

NetworkInterfaceId (p. 734)

The network interface IDs that Amazon SageMaker created at the time of creating the instance.

Type: String

NotebookInstanceArn (p. 734)

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

NotebookInstanceLifecycleConfigName (p. 734)

Returns the name of a notebook instance lifecycle configuration.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

NotebookInstanceName (p. 734)

The name of the Amazon SageMaker notebook instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

NotebookInstanceStatus (p. 734)

The status of the notebook instance.

Type: String

Valid Values: `Pending | InService | Stopping | Stopped | Failed | Deleting | Updating`

RoleArn (p. 734)

The Amazon Resource Name (ARN) of the IAM role associated with the instance.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

RootAccess (p. 734)

Whether root access is enabled or disabled for users of the notebook instance.

> **Note**
> Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.

Type: String

Valid Values: `Enabled | Disabled`

SecurityGroups (p. 734)

The IDs of the VPC security groups.

Type: Array of strings

Array Members: Maximum number of 5 items.

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

SubnetId (p. 734)

The ID of the VPC subnet.

Type: String

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Url (p. 734)

The URL that you use to connect to the Jupyter notebook that is running in your notebook instance.

Type: String

VolumeSizeInGB (p. 734)

The size, in GB, of the ML storage volume attached to the notebook instance.

Type: Integer

Valid Range: Minimum value of 5. Maximum value of 16384.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeNotebookInstanceLifecycleConfig

Service: Amazon SageMaker Service

Returns a description of a notebook instance lifecycle configuration.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Request Syntax

```
{
    "NotebookInstanceLifecycleConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceLifecycleConfigName (p. 739)

    The name of the lifecycle configuration to describe.

    Type: String

    Length Constraints: Maximum length of 63.

    Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

    Required: Yes

Response Syntax

```
{
    "CreationTime": number,
    "LastModifiedTime": number,
    "NotebookInstanceLifecycleConfigArn": "string",
    "NotebookInstanceLifecycleConfigName": "string",
    "OnCreate": [
        {
            "Content": "string"
        }
    ],
    "OnStart": [
        {
            "Content": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreationTime (p. 739)

A timestamp that tells when the lifecycle configuration was created.

Type: Timestamp

LastModifiedTime (p. 739)

A timestamp that tells when the lifecycle configuration was last modified.

Type: Timestamp

NotebookInstanceLifecycleConfigArn (p. 739)

The Amazon Resource Name (ARN) of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 256.

NotebookInstanceLifecycleConfigName (p. 739)

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

OnCreate (p. 739)

The shell script that runs only once, when you create a notebook instance.

Type: Array of NotebookInstanceLifecycleHook (p. 969) objects

Array Members: Maximum number of 1 item.

OnStart (p. 739)

The shell script that runs every time you start a notebook instance, including when you create the notebook instance.

Type: Array of NotebookInstanceLifecycleHook (p. 969) objects

Array Members: Maximum number of 1 item.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeSubscribedWorkteam

Service: Amazon SageMaker Service

Gets information about a work team provided by a vendor. It returns details about the subscription with a vendor in the AWS Marketplace.

Request Syntax

```
{
    "WorkteamArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

WorkteamArn (p. 742)

The Amazon Resource Name (ARN) of the subscribed work team to describe.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

Required: Yes

Response Syntax

```
{
    "SubscribedWorkteam": {
        "ListingId": "string",
        "MarketplaceDescription": "string",
        "MarketplaceTitle": "string",
        "SellerName": "string",
        "WorkteamArn": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

SubscribedWorkteam (p. 742)

A `Workteam` instance that contains information about the work team.

Type: SubscribedWorkteam (p. 1005) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeTrainingJob

Service: Amazon SageMaker Service

Returns information about a training job.

Request Syntax

```
{
    "TrainingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

TrainingJobName (p. 744)

>   The name of the training job.
>
>   Type: String
>
>   Length Constraints: Minimum length of 1. Maximum length of 63.
>
>   Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
>   Required: Yes

Response Syntax

```
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "BillableTimeInSeconds": number,
    "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
    },
    "CreationTime": number,
    "EnableInterContainerTrafficEncryption": boolean,
    "EnableManagedSpotTraining": boolean,
    "EnableNetworkIsolation": boolean,
    "FailureReason": "string",
    "FinalMetricDataList": [
        {
            "MetricName": "string",
            "Timestamp": number,
            "Value": number
        }
```

```
        ],
        "HyperParameters": {
            "string" : "string"
        },
        "InputDataConfig": [
            {
                "ChannelName": "string",
                "CompressionType": "string",
                "ContentType": "string",
                "DataSource": {
                    "FileSystemDataSource": {
                        "DirectoryPath": "string",
                        "FileSystemAccessMode": "string",
                        "FileSystemId": "string",
                        "FileSystemType": "string"
                    },
                    "S3DataSource": {
                        "AttributeNames": [ "string" ],
                        "S3DataDistributionType": "string",
                        "S3DataType": "string",
                        "S3Uri": "string"
                    }
                },
                "InputMode": "string",
                "RecordWrapperType": "string",
                "ShuffleConfig": {
                    "Seed": number
                }
            }
        ],
        "LabelingJobArn": "string",
        "LastModifiedTime": number,
        "ModelArtifacts": {
            "S3ModelArtifacts": "string"
        },
        "OutputDataConfig": {
            "KmsKeyId": "string",
            "S3OutputPath": "string"
        },
        "ResourceConfig": {
            "InstanceCount": number,
            "InstanceType": "string",
            "VolumeKmsKeyId": "string",
            "VolumeSizeInGB": number
        },
        "RoleArn": "string",
        "SecondaryStatus": "string",
        "SecondaryStatusTransitions": [
            {
                "EndTime": number,
                "StartTime": number,
                "Status": "string",
                "StatusMessage": "string"
            }
        ],
        "StoppingCondition": {
            "MaxRuntimeInSeconds": number,
            "MaxWaitTimeInSeconds": number
        },
        "TrainingEndTime": number,
        "TrainingJobArn": "string",
        "TrainingJobName": "string",
        "TrainingJobStatus": "string",
        "TrainingStartTime": number,
        "TrainingTimeInSeconds": number,
        "TuningJobArn": "string",
```

```
    "VpcConfig": {
      "SecurityGroupIds": [ "string" ],
      "Subnets": [ "string" ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AlgorithmSpecification (p. 744)

Information about the algorithm used for training, and algorithm metadata.

Type: AlgorithmSpecification (p. 863) object

BillableTimeInSeconds (p. 744)

The billable time in seconds.

You can calculate the savings from using managed spot training using the formula `(1
– BillableTimeInSeconds / TrainingTimeInSeconds) * 100`. For example, if
`BillableTimeInSeconds` is 100 and `TrainingTimeInSeconds` is 500, the savings is 80%.

Type: Integer

Valid Range: Minimum value of 1.

CheckpointConfig (p. 744)

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 880) object

CreationTime (p. 744)

A timestamp that indicates when the training job was created.

Type: Timestamp

EnableInterContainerTrafficEncryption (p. 744)

To encrypt all communications between ML compute instances in distributed training, choose `True`.
Encryption provides greater security for distributed training, but training might take longer. How
long it takes depends on the amount of communication between compute instances, especially if
you use a deep learning algorithms in distributed training.

Type: Boolean

EnableManagedSpotTraining (p. 744)

A Boolean indicating whether managed spot training is enabled (`True`) or not (`False`).

Type: Boolean

EnableNetworkIsolation (p. 744)

If you want to allow inbound or outbound network calls, except for calls between peers within a
training cluster for distributed training, choose `True`. If you enable network isolation for training
jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data
and model artifacts through the specified VPC, but the training container does not have network
access.

Note

The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

FailureReason (p. 744)

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

FinalMetricDataList (p. 744)

A collection of `MetricData` objects that specify the names, values, and dates and times that the training algorithm emitted to Amazon CloudWatch.

Type: Array of MetricData (p. 955) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

HyperParameters (p. 744)

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

InputDataConfig (p. 744)

An array of `Channel` objects that describes each data input channel.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

LabelingJobArn (p. 744)

The Amazon Resource Name (ARN) of the Amazon SageMaker Ground Truth labeling job that created the transform or training job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`

LastModifiedTime (p. 744)

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

ModelArtifacts (p. 744)

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: ModelArtifacts (p. 957) object

OutputDataConfig (p. 744)

The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.

Type: OutputDataConfig (p. 976) object

ResourceConfig (p. 744)

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: ResourceConfig (p. 991) object

RoleArn (p. 744)

The AWS Identity and Access Management (IAM) role configured for the training job.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

SecondaryStatus (p. 744)

Provides detailed information about the state of the training job. For detailed information on the secondary status of the training job, see `StatusMessage` under SecondaryStatusTransition (p. 999).

Amazon SageMaker provides primary statuses and secondary statuses that apply to each of them:

InProgress

- `Starting` - Starting the training job.
- `Downloading` - An optional stage for algorithms that support `File` training input mode. It indicates that data is being downloaded to the ML storage volumes.
- `Training` - Training is in progress.
- `Uploading` - Training is complete and the model artifacts are being uploaded to the S3 location.

Completed

- `Completed` - The training job has completed.

Failed

- `Failed` - The training job has failed. The reason for the failure is returned in the `FailureReason` field of `DescribeTrainingJobResponse`.

Stopped

- `MaxRuntimeExceeded` - The job stopped because it exceeded the maximum allowed runtime.
- `MaxWaitTmeExceeded` - The job stopped because it exceeded the maximum allowed wait time.
- `Interrupted` - The job stopped because the managed spot training instances were interrupted.
- `Stopped` - The training job has stopped.

Stopping

- `Stopping` - Stopping the training job.

  Important
  Valid values for `SecondaryStatus` are subject to change.

We no longer support the following secondary statuses:

- `LaunchingMLInstances`
- `PreparingTrainingStack`
- `DownloadingTrainingImage`

Type: String

Valid Values: `Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed | Interrupted | MaxWaitTimeExceeded`

SecondaryStatusTransitions (p. 744)

A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of SecondaryStatusTransition (p. 999) objects

StoppingCondition (p. 744)

Specifies a limit to how long a model training job can run. It also specifies the maximum time to wait for a spot instance. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1004) object

TrainingEndTime (p. 744)

Indicates the time when the training job ends on training instances. You are billed for the time interval between the value of `TrainingStartTime` and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: Timestamp

TrainingJobArn (p. 744)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

TrainingJobName (p. 744)

Name of the model training job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

TrainingJobStatus (p. 744)

The status of the training job.

Amazon SageMaker provides the following training job statuses:

- `InProgress` - The training is in progress.

- `Completed` - The training job has completed.
- `Failed` - The training job has failed. To see the reason for the failure, see the `FailureReason` field in the response to a `DescribeTrainingJobResponse` call.
- `Stopping` - The training job is stopping.
- `Stopped` - The training job has stopped.

For more detailed information, see `SecondaryStatus`.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

TrainingStartTime (p. 744)

Indicates the time when the training job starts on training instances. You are billed for the time interval between this time and the value of `TrainingEndTime`. The start time in CloudWatch Logs might be later than this time. The difference is due to the time it takes to download the training data and to the size of the training container.

Type: Timestamp

TrainingTimeInSeconds (p. 744)

The training time in seconds.

Type: Integer

Valid Range: Minimum value of 1.

TuningJobArn (p. 744)

The Amazon Resource Name (ARN) of the associated hyperparameter tuning job if the training job was launched by a hyperparameter tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

VpcConfig (p. 744)

A VpcConfig (p. 1039) object that specifies the VPC that this training job has access to. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeTransformJob

Service: Amazon SageMaker Service

Returns information about a transform job.

Request Syntax

```
{
    "TransformJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

TransformJobName (p. 752)

> The name of the transform job that you want to view details of.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 63.
>
> Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
> Required: Yes

Response Syntax

```
{
    "BatchStrategy": "string",
    "CreationTime": number,
    "DataProcessing": {
        "InputFilter": "string",
        "JoinSource": "string",
        "OutputFilter": "string"
    },
    "Environment": {
        "string" : "string"
    },
    "FailureReason": "string",
    "LabelingJobArn": "string",
    "MaxConcurrentTransforms": number,
    "MaxPayloadInMB": number,
    "ModelName": "string",
    "TransformEndTime": number,
    "TransformInput": {
        "CompressionType": "string",
        "ContentType": "string",
        "DataSource": {
            "S3DataSource": {
                "S3DataType": "string",
                "S3Uri": "string"
            }
        },
        "SplitType": "string"
    },
```

```
    "TransformJobArn": "string",
    "TransformJobName": "string",
    "TransformJobStatus": "string",
    "TransformOutput": {
        "Accept": "string",
        "AssembleWith": "string",
        "KmsKeyId": "string",
        "S3OutputPath": "string"
    },
    "TransformResources": {
        "InstanceCount": number,
        "InstanceType": "string",
        "VolumeKmsKeyId": "string"
    },
    "TransformStartTime": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

BatchStrategy (p. 752)

> Specifies the number of records to include in a mini-batch for an HTTP inference request. A *record* is a single unit of input data that inference can be made on. For example, a single line in a CSV file is a record.
>
> To enable the batch strategy, you must set `SplitType` to `Line`, `RecordIO`, or `TFRecord`.
>
> Type: String
>
> Valid Values: `MultiRecord | SingleRecord`

CreationTime (p. 752)

> A timestamp that shows when the transform Job was created.
>
> Type: Timestamp

DataProcessing (p. 752)

> The data structure used to specify the data to be used for inference in a batch transform job and to associate the data that is relevant to the prediction results in the output. The input filter provided allows you to exclude input data that is not needed for inference in a batch transform job. The output filter provided allows you to include input data relevant to interpreting the predictions in the output from the job. For more information, see Associate Prediction Results with their Corresponding Input Records.
>
> Type: DataProcessing (p. 891) object

Environment (p. 752)

> The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.
>
> Type: String to string map
>
> Key Length Constraints: Maximum length of 1024.
>
> Key Pattern: `[a-zA-Z_][a-zA-Z0-9_]*`
>
> Value Length Constraints: Maximum length of 10240.

Value Pattern: `[\S\s]*`

FailureReason (p. 752)

If the transform job failed, `FailureReason` describes why it failed. A transform job creates a log file, which includes error messages, and stores it as an Amazon S3 object. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch.

Type: String

Length Constraints: Maximum length of 1024.

LabelingJobArn (p. 752)

The Amazon Resource Name (ARN) of the Amazon SageMaker Ground Truth labeling job that created the transform or training job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`

MaxConcurrentTransforms (p. 752)

The maximum number of parallel requests on each instance node that can be launched in a transform job. The default value is 1.

Type: Integer

Valid Range: Minimum value of 0.

MaxPayloadInMB (p. 752)

The maximum payload size, in MB, used in the transform job.

Type: Integer

Valid Range: Minimum value of 0.

ModelName (p. 752)

The name of the model used in the transform job.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

TransformEndTime (p. 752)

Indicates when the transform job has been completed, or has stopped or failed. You are billed for the time interval between this time and the value of `TransformStartTime`.

Type: Timestamp

TransformInput (p. 752)

Describes the dataset to be transformed and the Amazon S3 location where it is stored.

Type: TransformInput (p. 1024) object

TransformJobArn (p. 752)

The Amazon Resource Name (ARN) of the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*`

TransformJobName (p. 752)

The name of the transform job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

TransformJobStatus (p. 752)

The status of the transform job. If the transform job failed, the reason is returned in the `FailureReason` field.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

TransformOutput (p. 752)

Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.

Type: TransformOutput (p. 1030) object

TransformResources (p. 752)

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: TransformResources (p. 1032) object

TransformStartTime (p. 752)

Indicates when the transform job starts on ML instances. You are billed for the time interval between this time and the value of `TransformEndTime`.

Type: Timestamp

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeWorkteam

Service: Amazon SageMaker Service

Gets information about a specific work team. You can see information such as the create date, the last updated date, membership information, and the work team's Amazon Resource Name (ARN).

Request Syntax

```
{
    "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

WorkteamName (p. 757)

> The name of the work team to return a description of.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 63.
>
> Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
>
> Required: Yes

Response Syntax

```
{
    "Workteam": {
        "CreateDate": number,
        "Description": "string",
        "LastUpdatedDate": number,
        "MemberDefinitions": [
            {
                "CognitoMemberDefinition": {
                    "ClientId": "string",
                    "UserGroup": "string",
                    "UserPool": "string"
                }
            }
        ],
        "NotificationConfiguration": {
            "NotificationTopicArn": "string"
        },
        "ProductListingIds": [ "string" ],
        "SubDomain": "string",
        "WorkteamArn": "string",
        "WorkteamName": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Workteam (p. 757)

A `Workteam` instance that contains information about the work team.

Type: Workteam (p. 1040) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

GetSearchSuggestions

Service: Amazon SageMaker Service

An auto-complete API for the search functionality in the Amazon SageMaker console. It returns suggestions of possible matches for the property name to use in `Search` queries. Provides suggestions for `HyperParameters`, `Tags`, and `Metrics`.

Request Syntax

```
{
    "Resource": "string",
    "SuggestionQuery": {
        "PropertyNameQuery": {
            "PropertyNameHint": "string"
        }
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

Resource (p. 759)

> The name of the Amazon SageMaker resource to Search for. The only valid `Resource` value is `TrainingJob`.
>
> Type: String
>
> Valid Values: `TrainingJob`
>
> Required: Yes

SuggestionQuery (p. 759)

> Limits the property names that are included in the response.
>
> Type: SuggestionQuery (p. 1007) object
>
> Required: No

Response Syntax

```
{
    "PropertyNameSuggestions": [
        {
            "PropertyName": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

PropertyNameSuggestions (p. 759)

A list of property names for a `Resource` that match a `SuggestionQuery`.

Type: Array of PropertyNameSuggestion (p. 986) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListAlgorithms

Service: Amazon SageMaker Service

Lists the machine learning algorithms that have been created.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 761)

A filter that returns only algorithms created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 761)

A filter that returns only algorithms created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 761)

The maximum number of algorithms to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 761)

A string in the algorithm name. This filter returns only algorithms whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

NextToken (p. 761)

> If the response to a previous `ListAlgorithms` request was truncated, the response includes a `NextToken`. To retrieve the next set of algorithms, use the token in the next request.
>
> Type: String
>
> Length Constraints: Maximum length of 8192.
>
> Pattern: `.*`
>
> Required: No

SortBy (p. 761)

> The parameter by which to sort the results. The default is `CreationTime`.
>
> Type: String
>
> Valid Values: `Name | CreationTime`
>
> Required: No

SortOrder (p. 761)

> The sort order for the results. The default is `Ascending`.
>
> Type: String
>
> Valid Values: `Ascending | Descending`
>
> Required: No

Response Syntax

```
{
   "AlgorithmSummaryList": [
      {
         "AlgorithmArn": "string",
         "AlgorithmDescription": "string",
         "AlgorithmName": "string",
         "AlgorithmStatus": "string",
         "CreationTime": number
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AlgorithmSummaryList (p. 762)

> >An array of `AlgorithmSummary` objects, each of which lists an algorithm.
>
> Type: Array of AlgorithmSummary (p. 867) objects

NextToken (p. 762)

> If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of algorithms, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListCodeRepositories

Service: Amazon SageMaker Service

Gets a list of the Git repositories in your account.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 764)

A filter that returns only Git repositories that were created after the specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 764)

A filter that returns only Git repositories that were created before the specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 764)

A filter that returns only Git repositories that were last modified after the specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 764)

A filter that returns only Git repositories that were last modified before the specified time.

Type: Timestamp

Required: No

MaxResults (p. 764)

The maximum number of Git repositories to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 764)

A string in the Git repositories name. This filter returns only repositories whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9-]+`

Required: No

NextToken (p. 764)

If the result of a `ListCodeRepositoriesOutput` request was truncated, the response includes a `NextToken`. To get the next set of Git repositories, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 764)

The field to sort results by. The default is `Name`.

Type: String

Valid Values: `Name | CreationTime | LastModifiedTime`

Required: No

SortOrder (p. 764)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

Response Syntax

```
{
   "CodeRepositorySummaryList": [
      {
         "CodeRepositoryArn": "string",
         "CodeRepositoryName": "string",
         "CreationTime": number,
         "GitConfig": {
            "Branch": "string",
            "RepositoryUrl": "string",
            "SecretArn": "string"
         },
         "LastModifiedTime": number
```

```
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeRepositorySummaryList (p. 765)

Gets a list of summaries of the Git repositories. Each summary specifies the following values for the repository:

- Name
- Amazon Resource Name (ARN)
- Creation time
- Last modified time
- Configuration information, including the URL location of the repository and the ARN of the AWS Secrets Manager secret that contains the credentials used to access the repository.

Type: Array of CodeRepositorySummary (p. 881) objects

NextToken (p. 765)

If the result of a `ListCodeRepositoriesOutput` request was truncated, the response includes a `NextToken`. To get the next set of Git repositories, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListCompilationJobs

Service: Amazon SageMaker Service

Lists model compilation jobs that satisfy various filters.

To create a model compilation job, use CreateCompilationJob (p. 629). To get information about a particular model compilation job you have created, use DescribeCompilationJob (p. 705).

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 767)

A filter that returns the model compilation jobs that were created after a specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 767)

A filter that returns the model compilation jobs that were created before a specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 767)

A filter that returns the model compilation jobs that were modified after a specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 767)

A filter that returns the model compilation jobs that were modified before a specified time.

Type: Timestamp

Required: No

MaxResults (p. 767)

The maximum number of model compilation jobs to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 767)

A filter that returns the model compilation jobs whose name contains a specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

NextToken (p. 767)

If the result of the previous `ListCompilationJobs` request was truncated, the response includes a `NextToken`. To retrieve the next set of model compilation jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 767)

The field by which to sort results. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime | Status`

Required: No

SortOrder (p. 767)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 767)

A filter that retrieves model compilation jobs with a specific DescribeCompilationJob:CompilationJobStatus (p. 706) status.

Type: String

Valid Values: `INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED`

Required: No

Response Syntax

```
{
    "CompilationJobSummaries": [
        {
            "CompilationEndTime": number,
            "CompilationJobArn": "string",
            "CompilationJobName": "string",
            "CompilationJobStatus": "string",
            "CompilationStartTime": number,
            "CompilationTargetDevice": "string",
            "CreationTime": number,
            "LastModifiedTime": number
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CompilationJobSummaries (p. 769)

An array of CompilationJobSummary (p. 884) objects, each describing a model compilation job.

Type: Array of CompilationJobSummary (p. 884) objects

NextToken (p. 769)

If the response is truncated, Amazon SageMaker returns this `NextToken`. To retrieve the next set of model compilation jobs, use this token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V2

ListEndpointConfigs

Service: Amazon SageMaker Service

Lists endpoint configurations.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 771)

A filter that returns only endpoint configurations with a creation time greater than or equal to the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 771)

A filter that returns only endpoint configurations created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 771)

The maximum number of training jobs to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 771)

A string in the endpoint configuration name. This filter returns only endpoint configurations whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9-]+`

Required: No

If the result of the previous `ListEndpointConfig` request was truncated, the response includes a `NextToken`. To retrieve the next set of endpoint configurations, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime`

Required: No

The sort order for results. The default is `Descending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

Response Syntax

```
{
   "EndpointConfigs": [
      {
         "CreationTime": number,
         "EndpointConfigArn": "string",
         "EndpointConfigName": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

An array of endpoint configurations.

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of endpoint configurations, use it in the subsequent request

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListEndpoints

Service: Amazon SageMaker Service

Lists endpoints.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 774)

A filter that returns only endpoints with a creation time greater than or equal to the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 774)

A filter that returns only endpoints that were created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 774)

A filter that returns only endpoints that were modified after the specified timestamp.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 774)

A filter that returns only endpoints that were modified before the specified timestamp.

Type: Timestamp

Required: No

MaxResults (p. 774)

The maximum number of endpoints to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 774)

A string in endpoint names. This filter returns only endpoints whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9-]+`

Required: No

NextToken (p. 774)

If the result of a `ListEndpoints` request was truncated, the response includes a `NextToken`. To retrieve the next set of endpoints, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 774)

Sorts the list of results. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime | Status`

Required: No

SortOrder (p. 774)

The sort order for results. The default is `Descending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 774)

A filter that returns only endpoints with the specified status.

Type: String

Valid Values: `OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed`

Required: No

Response Syntax

```
{
```

```
    "Endpoints": [
        {
            "CreationTime": number,
            "EndpointArn": "string",
            "EndpointName": "string",
            "EndpointStatus": "string",
            "LastModifiedTime": number
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Endpoints (p. 775)

> An array or endpoint objects.

> Type: Array of EndpointSummary (p. 897) objects

NextToken (p. 775)

> If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of training jobs, use it in the subsequent request.

> Type: String

> Length Constraints: Maximum length of 8192.

> Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListHyperParameterTuningJobs

Service: Amazon SageMaker Service

Gets a list of HyperParameterTuningJobSummary (p. 925) objects that describe the hyperparameter tuning jobs launched in your account.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 777)

A filter that returns only tuning jobs that were created after the specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 777)

A filter that returns only tuning jobs that were created before the specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 777)

A filter that returns only tuning jobs that were modified after the specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 777)

A filter that returns only tuning jobs that were modified before the specified time.

Type: Timestamp

Required: No

MaxResults (p. 777)

The maximum number of tuning jobs to return. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 777)

A string in the tuning job name. This filter returns only tuning jobs whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

NextToken (p. 777)

If the result of the previous `ListHyperParameterTuningJobs` request was truncated, the response includes a `NextToken`. To retrieve the next set of tuning jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 777)

The field to sort results by. The default is `Name`.

Type: String

Valid Values: `Name | Status | CreationTime`

Required: No

SortOrder (p. 777)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 777)

A filter that returns only tuning jobs with the specified status.

Type: String

Valid Values: `Completed | InProgress | Failed | Stopped | Stopping`

Required: No

Response Syntax

```
{
```

```
    "HyperParameterTuningJobSummaries": [
        {
            "CreationTime": number,
            "HyperParameterTuningEndTime": number,
            "HyperParameterTuningJobArn": "string",
            "HyperParameterTuningJobName": "string",
            "HyperParameterTuningJobStatus": "string",
            "LastModifiedTime": number,
            "ObjectiveStatusCounters": {
                "Failed": number,
                "Pending": number,
                "Succeeded": number
            },
            "ResourceLimits": {
                "MaxNumberOfTrainingJobs": number,
                "MaxParallelTrainingJobs": number
            },
            "Strategy": "string",
            "TrainingJobStatusCounters": {
                "Completed": number,
                "InProgress": number,
                "NonRetryableError": number,
                "RetryableError": number,
                "Stopped": number
            }
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

HyperParameterTuningJobSummaries (p. 778)

A list of HyperParameterTuningJobSummary (p. 925) objects that describe the tuning jobs that the `ListHyperParameterTuningJobs` request returned.

Type: Array of HyperParameterTuningJobSummary (p. 925) objects

NextToken (p. 778)

If the result of this `ListHyperParameterTuningJobs` request was truncated, the response includes a `NextToken`. To retrieve the next set of tuning jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListLabelingJobs

Service: Amazon SageMaker Service

Gets a list of labeling jobs.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 781)

A filter that returns only labeling jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 781)

A filter that returns only labeling jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 781)

A filter that returns only labeling jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 781)

A filter that returns only labeling jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 781)

The maximum number of labeling jobs to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 781)

A string in the labeling job name. This filter returns only labeling jobs whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

NextToken (p. 781)

If the result of the previous `ListLabelingJobs` request was truncated, the response includes a `NextToken`. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 781)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime | Status`

Required: No

SortOrder (p. 781)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 781)

A filter that retrieves only labeling jobs with a specific status.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: No

Response Syntax

```
{
```

```
    "LabelingJobSummaryList": [
        {
            "AnnotationConsolidationLambdaArn": "string",
            "CreationTime": number,
            "FailureReason": "string",
            "InputConfig": {
                "DataAttributes": {
                    "ContentClassifiers": [ "string" ]
                },
                "DataSource": {
                    "S3DataSource": {
                        "ManifestS3Uri": "string"
                    }
                }
            },
            "LabelCounters": {
                "FailedNonRetryableError": number,
                "HumanLabeled": number,
                "MachineLabeled": number,
                "TotalLabeled": number,
                "Unlabeled": number
            },
            "LabelingJobArn": "string",
            "LabelingJobName": "string",
            "LabelingJobOutput": {
                "FinalActiveLearningModelArn": "string",
                "OutputDatasetS3Uri": "string"
            },
            "LabelingJobStatus": "string",
            "LastModifiedTime": number,
            "PreHumanTaskLambdaArn": "string",
            "WorkteamArn": "string"
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

LabelingJobSummaryList (p. 782)

An array of `LabelingJobSummary` objects, each describing a labeling job.

Type: Array of LabelingJobSummary (p. 951) objects

NextToken (p. 782)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of labeling jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListLabelingJobsForWorkteam

Service: Amazon SageMaker Service

Gets a list of labeling jobs assigned to a specified work team.

Request Syntax

```
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "JobReferenceCodeContains": "string",
   "MaxResults": number,
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "WorkteamArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 785)

A filter that returns only labeling jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 785)

A filter that returns only labeling jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

JobReferenceCodeContains (p. 785)

A filter the limits jobs to only the ones whose job reference code contains the specified string.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: No

MaxResults (p. 785)

The maximum number of labeling jobs to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NextToken (p. 785)

If the result of the previous `ListLabelingJobsForWorkteam` request was truncated, the response includes a `NextToken`. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 785)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `CreationTime`

Required: No

SortOrder (p. 785)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

WorkteamArn (p. 785)

The Amazon Resource Name (ARN) of the work team for which you want to see labeling jobs for.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

Required: Yes

Response Syntax

```
{
   "LabelingJobSummaryList": [
      {
         "CreationTime": number,
         "JobReferenceCode": "string",
         "LabelCounters": {
            "HumanLabeled": number,
            "PendingHuman": number,
            "Total": number
         },
         "LabelingJobName": "string",
         "NumberOfHumanWorkersPerDataObject": number,
         "WorkRequesterAccountId": "string"
      }
   ],
   "NextToken": "string"
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

LabelingJobSummaryList (p. 786)

An array of `LabelingJobSummary` objects, each describing a labeling job.

Type: Array of LabelingJobForWorkteamSummary (p. 943) objects

NextToken (p. 786)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of labeling jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListModelPackages

Service: Amazon SageMaker Service

Lists the model packages that have been created.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 788)

A filter that returns only model packages created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 788)

A filter that returns only model packages created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 788)

The maximum number of model packages to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 788)

A string in the model package name. This filter returns only model packages whose name contains
the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

NextToken (p. 788)

> If the response to a previous `ListModelPackages` request was truncated, the response includes a `NextToken`. To retrieve the next set of model packages, use the token in the next request.
>
> Type: String
>
> Length Constraints: Maximum length of 8192.
>
> Pattern: `.*`
>
> Required: No

SortBy (p. 788)

> The parameter by which to sort the results. The default is `CreationTime`.
>
> Type: String
>
> Valid Values: `Name | CreationTime`
>
> Required: No

SortOrder (p. 788)

> The sort order for the results. The default is `Ascending`.
>
> Type: String
>
> Valid Values: `Ascending | Descending`
>
> Required: No

Response Syntax

```
{
    "ModelPackageSummaryList": [
        {
            "CreationTime": number,
            "ModelPackageArn": "string",
            "ModelPackageDescription": "string",
            "ModelPackageName": "string",
            "ModelPackageStatus": "string"
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelPackageSummaryList (p. 789)

> An array of `ModelPackageSummary` objects, each of which lists a model package.
>
> Type: Array of ModelPackageSummary (p. 962) objects

NextToken (p. 789)

> If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of model packages, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListModels

Service: Amazon SageMaker Service

Lists models created with the CreateModel API.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 791)

A filter that returns only models with a creation time greater than or equal to the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 791)

A filter that returns only models created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 791)

The maximum number of models to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 791)

A string in the training job name. This filter returns only models in the training job whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9-]+`

Required: No

NextToken (p. 791)

If the response to a previous `ListModels` request was truncated, the response includes a `NextToken`. To retrieve the next set of models, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 791)

Sorts the list of results. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime`

Required: No

SortOrder (p. 791)

The sort order for results. The default is `Descending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

Response Syntax

```
{
   "Models": [
      {
         "CreationTime": number,
         "ModelArn": "string",
         "ModelName": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Models (p. 792)

An array of `ModelSummary` objects, each of which lists a model.

Type: Array of ModelSummary (p. 966) objects

NextToken (p. 792)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of models, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListNotebookInstanceLifecycleConfigs

Service: Amazon SageMaker Service

Lists notebook instance lifestyle configurations created with the
CreateNotebookInstanceLifecycleConfig (p. 662) API.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 794)

A filter that returns only lifecycle configurations that were created after the specified time
(timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 794)

A filter that returns only lifecycle configurations that were created before the specified time
(timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 794)

A filter that returns only lifecycle configurations that were modified after the specified time
(timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 794)

A filter that returns only lifecycle configurations that were modified before the specified time
(timestamp).

Type: Timestamp

Required: No

MaxResults (p. 794)

The maximum number of lifecycle configurations to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 794)

A string in the lifecycle configuration name. This filter returns only lifecycle configurations whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9-]+`

Required: No

NextToken (p. 794)

If the result of a `ListNotebookInstanceLifecycleConfigs` request was truncated, the response includes a `NextToken`. To get the next set of lifecycle configurations, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 794)

Sorts the list of results. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime | LastModifiedTime`

Required: No

SortOrder (p. 794)

The sort order for results.

Type: String

Valid Values: `Ascending | Descending`

Required: No

Response Syntax

```
{
   "NextToken": "string",
   "NotebookInstanceLifecycleConfigs": [
      {
```

```
            "CreationTime": number,
            "LastModifiedTime": number,
            "NotebookInstanceLifecycleConfigArn": "string",
            "NotebookInstanceLifecycleConfigName": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 795)

If the response is truncated, Amazon SageMaker returns this token. To get the next set of lifecycle configurations, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

NotebookInstanceLifecycleConfigs (p. 795)

An array of `NotebookInstanceLifecycleConfiguration` objects, each listing a lifecycle configuration.

Type: Array of NotebookInstanceLifecycleConfigSummary (p. 968) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListNotebookInstances

Service: Amazon SageMaker Service

Returns a list of the Amazon SageMaker notebook instances in the requester's account in an AWS Region.

Request Syntax

```
{
    "AdditionalCodeRepositoryEquals": "string",
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "DefaultCodeRepositoryContains": "string",
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "NotebookInstanceLifecycleConfigNameContains": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AdditionalCodeRepositoryEquals (p. 797)

A filter that returns only notebook instances with associated with the specified git repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

CreationTimeAfter (p. 797)

A filter that returns only notebook instances that were created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 797)

A filter that returns only notebook instances that were created before the specified time (timestamp).

Type: Timestamp

Required: No

DefaultCodeRepositoryContains (p. 797)

A string in the name or URL of a Git repository associated with this notebook instance. This filter returns only notebook instances associated with a git repository with a name that contains the specified string.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[a-zA-Z0-9-]+`

Required: No

LastModifiedTimeAfter (p. 797)

A filter that returns only notebook instances that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 797)

A filter that returns only notebook instances that were modified before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 797)

The maximum number of notebook instances to return.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 797)

A string in the notebook instances' name. This filter returns only notebook instances whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9-]+`

Required: No

NextToken (p. 797)

If the previous call to the `ListNotebookInstances` is truncated, the response includes a `NextToken`. You can use this token in your subsequent `ListNotebookInstances` request to fetch the next set of notebook instances.

> **Note**
> You might specify a filter or a sort order in your request. When response is truncated, you must use the same values for the filer and sort order in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

NotebookInstanceLifecycleConfigNameContains (p. 797)

A string in the name of a notebook instances lifecycle configuration associated with this notebook instance. This filter returns only notebook instances associated with a lifecycle configuration with a name that contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

SortBy (p. 797)

The field to sort results by. The default is `Name`.

Type: String

Valid Values: `Name | CreationTime | Status`

Required: No

SortOrder (p. 797)

The sort order for results.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 797)

A filter that returns only notebook instances with the specified status.

Type: String

Valid Values: `Pending | InService | Stopping | Stopped | Failed | Deleting | Updating`

Required: No

Response Syntax

```
{
    "NextToken": "string",
    "NotebookInstances": [
        {
            "AdditionalCodeRepositories": [ "string" ],
            "CreationTime": number,
            "DefaultCodeRepository": "string",
            "InstanceType": "string",
            "LastModifiedTime": number,
            "NotebookInstanceArn": "string",
            "NotebookInstanceLifecycleConfigName": "string",
            "NotebookInstanceName": "string",
            "NotebookInstanceStatus": "string",
            "Url": "string"
        }
    ]
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 799)

> If the response to the previous `ListNotebookInstances` request was truncated, Amazon
> SageMaker returns this token. To retrieve the next set of notebook instances, use the token in the
> next request.
>
> Type: String
>
> Length Constraints: Maximum length of 8192.
>
> Pattern: `.*`

NotebookInstances (p. 799)

> An array of `NotebookInstanceSummary` objects, one for each notebook instance.
>
> Type: Array of NotebookInstanceSummary (p. 970) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListSubscribedWorkteams

Service: Amazon SageMaker Service

Gets a list of the work teams that you are subscribed to in the AWS Marketplace. The list may be empty if no work team satisfies the filter specified in the `NameContains` parameter.

Request Syntax

```
{
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

MaxResults (p. 801)

The maximum number of work teams to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 801)

A string in the work team name. This filter returns only work teams whose name contains the specified string.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

NextToken (p. 801)

If the result of the previous `ListSubscribedWorkteams` request was truncated, the response includes a `NextToken`. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

Response Syntax

```
{
```

```
    "NextToken": "string",
    "SubscribedWorkteams": [
        {
            "ListingId": "string",
            "MarketplaceDescription": "string",
            "MarketplaceTitle": "string",
            "SellerName": "string",
            "WorkteamArn": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 801)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of work teams, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

SubscribedWorkteams (p. 801)

An array of `Workteam` objects, each describing a work team.

Type: Array of SubscribedWorkteam (p. 1005) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListTags

Service: Amazon SageMaker Service

Returns the tags for the specified Amazon SageMaker resource.

Request Syntax

```
{
   "MaxResults": number,
   "NextToken": "string",
   "ResourceArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

MaxResults (p. 803)

Maximum number of tags to return.

Type: Integer

Valid Range: Minimum value of 50.

Required: No

NextToken (p. 803)

If the response to the previous `ListTags` request is truncated, Amazon SageMaker returns this
token. To retrieve the next set of tags, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

ResourceArn (p. 803)

The Amazon Resource Name (ARN) of the resource whose tags you want to retrieve.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:.*`

Required: Yes

Response Syntax

```
{
   "NextToken": "string",
   "Tags": [
```

```
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 803)

If response is truncated, Amazon SageMaker includes a token in the response. You can use this token in your subsequent request to fetch next set of tokens.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Tags (p. 803)

An array of `Tag` objects, each with a tag key and a value.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListTrainingJobs

Service: Amazon SageMaker Service

Lists training jobs.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 805)

A filter that returns only training jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 805)

A filter that returns only training jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 805)

A filter that returns only training jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 805)

A filter that returns only training jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 805)

The maximum number of training jobs to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 805)

A string in the training job name. This filter returns only training jobs whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

NextToken (p. 805)

If the result of the previous `ListTrainingJobs` request was truncated, the response includes a `NextToken`. To retrieve the next set of training jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 805)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime | Status`

Required: No

SortOrder (p. 805)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 805)

A filter that retrieves only training jobs with a specific status.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: No

Response Syntax

```
{
    "NextToken": "string",
```

```
    "TrainingJobSummaries": [
        {
            "CreationTime": number,
            "LastModifiedTime": number,
            "TrainingEndTime": number,
            "TrainingJobArn": "string",
            "TrainingJobName": "string",
            "TrainingJobStatus": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 806)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of training jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

TrainingJobSummaries (p. 806)

An array of `TrainingJobSummary` objects, each listing a training job.

Type: Array of TrainingJobSummary (p. 1019) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListTrainingJobsForHyperParameterTuningJob

Service: Amazon SageMaker Service

Gets a list of TrainingJobSummary (p. 1019) objects that describe the training jobs that a hyperparameter tuning job launched.

Request Syntax

```
{
    "HyperParameterTuningJobName": "string",
    "MaxResults": number,
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 808)

    The name of the tuning job whose training jobs you want to list.

    Type: String

    Length Constraints: Minimum length of 1. Maximum length of 32.

    Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

    Required: Yes

MaxResults (p. 808)

    The maximum number of training jobs to return. The default value is 10.

    Type: Integer

    Valid Range: Minimum value of 1. Maximum value of 100.

    Required: No

NextToken (p. 808)

    If the result of the previous `ListTrainingJobsForHyperParameterTuningJob` request was truncated, the response includes a `NextToken`. To retrieve the next set of training jobs, use the token in the next request.

    Type: String

    Length Constraints: Maximum length of 8192.

    Pattern: `.*`

    Required: No

SortBy (p. 808)

    The field to sort results by. The default is `Name`.

If the value of this field is `FinalObjectiveMetricValue`, any training jobs that did not return an objective metric are not listed.

Type: String

Valid Values: `Name | CreationTime | Status | FinalObjectiveMetricValue`

Required: No

SortOrder (p. 808)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 808)

A filter that returns only training jobs with the specified status.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: No

Response Syntax

```
{
   "NextToken": "string",
   "TrainingJobSummaries": [
      {
         "CreationTime": number,
         "FailureReason": "string",
         "FinalHyperParameterTuningJobObjectiveMetric": {
            "MetricName": "string",
            "Type": "string",
            "Value": number
         },
         "ObjectiveStatus": "string",
         "TrainingEndTime": number,
         "TrainingJobArn": "string",
         "TrainingJobName": "string",
         "TrainingJobStatus": "string",
         "TrainingStartTime": number,
         "TunedHyperParameters": {
            "string" : "string"
         },
         "TuningJobName": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 809)

If the result of this `ListTrainingJobsForHyperParameterTuningJob` request was truncated, the response includes a `NextToken`. To retrieve the next set of training jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

TrainingJobSummaries (p. 809)

A list of TrainingJobSummary (p. 1019) objects that describe the training jobs that the `ListTrainingJobsForHyperParameterTuningJob` request returned.

Type: Array of HyperParameterTrainingJobSummary (p. 919) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListTransformJobs

Service: Amazon SageMaker Service

Lists transform jobs.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 811)

A filter that returns only transform jobs created after the specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 811)

A filter that returns only transform jobs created before the specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 811)

A filter that returns only transform jobs modified after the specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 811)

A filter that returns only transform jobs modified before the specified time.

Type: Timestamp

Required: No

MaxResults (p. 811)

The maximum number of transform jobs to return in the response. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 811)

A string in the transform job name. This filter returns only transform jobs whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

NextToken (p. 811)

If the result of the previous `ListTransformJobs` request was truncated, the response includes a `NextToken`. To retrieve the next set of transform jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 811)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreationTime | Status`

Required: No

SortOrder (p. 811)

The sort order for results. The default is `Descending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

StatusEquals (p. 811)

A filter that retrieves only transform jobs with a specific status.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: No

Response Syntax

```
{
   "NextToken": "string",
```

```
    "TransformJobSummaries": [
        {
            "CreationTime": number,
            "FailureReason": "string",
            "LastModifiedTime": number,
            "TransformEndTime": number,
            "TransformJobArn": "string",
            "TransformJobName": "string",
            "TransformJobStatus": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 812)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of transform jobs, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

TransformJobSummaries (p. 812)

An array of `TransformJobSummary` objects.

Type: Array of TransformJobSummary (p. 1028) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListWorkteams

Service: Amazon SageMaker Service

Gets a list of work teams that you have defined in a region. The list may be empty if no work team satisfies the filter specified in the `NameContains` parameter.

Request Syntax

```
{
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

MaxResults (p. 814)

The maximum number of work teams to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 814)

A string in the work team's name. This filter returns only work teams whose name contains the specified string.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

NextToken (p. 814)

If the result of the previous `ListWorkteams` request was truncated, the response includes a `NextToken`. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

SortBy (p. 814)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name | CreateDate`

Required: No

SortOrder (p. 814)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

Response Syntax

```
{
    "NextToken": "string",
    "Workteams": [
        {
            "CreateDate": number,
            "Description": "string",
            "LastUpdatedDate": number,
            "MemberDefinitions": [
                {
                    "CognitoMemberDefinition": {
                        "ClientId": "string",
                        "UserGroup": "string",
                        "UserPool": "string"
                    }
                }
            ],
            "NotificationConfiguration": {
                "NotificationTopicArn": "string"
            },
            "ProductListingIds": [ "string" ],
            "SubDomain": "string",
            "WorkteamArn": "string",
            "WorkteamName": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 815)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of work teams, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Workteams (p. 815)

An array of `Workteam` objects, each describing a work team.

Type: Array of Workteam (p. 1040) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

RenderUiTemplate

Service: Amazon SageMaker Service

Renders the UI template so that you can preview the worker's experience.

Request Syntax

```
{
   "RoleArn": "string",
   "Task": {
      "Input": "string"
   },
   "UiTemplate": {
      "Content": "string"
   }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

RoleArn (p. 817)

The Amazon Resource Name (ARN) that has access to the S3 objects that are used by the template.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

Task (p. 817)

A `RenderableTask` object containing a representative task to render.

Type: RenderableTask (p. 989) object

Required: Yes

UiTemplate (p. 817)

A `Template` object containing the worker UI template to render.

Type: UiTemplate (p. 1037) object

Required: Yes

Response Syntax

```
{
   "Errors": [
      {
         "Code": "string",
         "Message": "string"
      }
```

```
    ],
    "RenderedContent": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Errors (p. 817)

A list of one or more `RenderingError` objects if any were encountered while rendering the template. If there were no errors, the list is empty.

Type: Array of RenderingError (p. 990) objects

RenderedContent (p. 817)

A Liquid template that renders the HTML for the worker UI.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Search

Service: Amazon SageMaker Service

Finds Amazon SageMaker resources that match a search query. Matching resource objects are returned as a list of `SearchResult` objects in the response. You can sort the search results by any resource property in a ascending or descending order.

You can query against the following value types: numerical, text, Booleans, and timestamps.

Request Syntax

```
{
    "MaxResults": number,
    "NextToken": "string",
    "Resource": "string",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "string",
                "Operator": "string",
                "Value": "string"
            }
        ],
        "NestedFilters": [
            {
                "Filters": [
                    {
                        "Name": "string",
                        "Operator": "string",
                        "Value": "string"
                    }
                ],
                "NestedPropertyName": "string"
            }
        ],
        "Operator": "string",
        "SubExpressions": [
            "SearchExpression"
        ]
    },
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

MaxResults (p. 819)

> The maximum number of results to return in a `SearchResponse`.
>
> Type: Integer
>
> Valid Range: Minimum value of 1. Maximum value of 100.
>
> Required: No

NextToken (p. 819)

If more than `MaxResults` resource objects match the specified `SearchExpression`, the `SearchResponse` includes a `NextToken`. The `NextToken` can be passed to the next `SearchRequest` to continue retrieving results for the specified `SearchExpression` and `Sort` parameters.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

Resource (p. 819)

The name of the Amazon SageMaker resource to search for. Currently, the only valid `Resource` value is `TrainingJob`.

Type: String

Valid Values: `TrainingJob`

Required: Yes

SearchExpression (p. 819)

A Boolean conditional statement. Resource objects must satisfy this condition to be included in search results. You must provide at least one subexpression, filter, or nested filter. The maximum number of recursive `SubExpressions`, `NestedFilters`, and `Filters` that can be included in a `SearchExpression` object is 50.

Type: SearchExpression (p. 996) object

Required: No

SortBy (p. 819)

The name of the resource property used to sort the `SearchResults`. The default is `LastModifiedTime`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: No

SortOrder (p. 819)

How `SearchResults` are ordered. Valid values are `Ascending` or `Descending`. The default is `Descending`.

Type: String

Valid Values: `Ascending | Descending`

Required: No

Response Syntax

```
{
    "NextToken": "string",
```

```
"Results": [
   {
      "TrainingJob": {
         "AlgorithmSpecification": {
            "AlgorithmName": "string",
            "MetricDefinitions": [
               {
                  "Name": "string",
                  "Regex": "string"
               }
            ],
            "TrainingImage": "string",
            "TrainingInputMode": "string"
         },
         "CreationTime": number,
         "EnableInterContainerTrafficEncryption": boolean,
         "EnableNetworkIsolation": boolean,
         "FailureReason": "string",
         "FinalMetricDataList": [
            {
               "MetricName": "string",
               "Timestamp": number,
               "Value": number
            }
         ],
         "HyperParameters": {
            "string" : "string"
         },
         "InputDataConfig": [
            {
               "ChannelName": "string",
               "CompressionType": "string",
               "ContentType": "string",
               "DataSource": {
                  "FileSystemDataSource": {
                     "DirectoryPath": "string",
                     "FileSystemAccessMode": "string",
                     "FileSystemId": "string",
                     "FileSystemType": "string"
                  },
                  "S3DataSource": {
                     "AttributeNames": [ "string" ],
                     "S3DataDistributionType": "string",
                     "S3DataType": "string",
                     "S3Uri": "string"
                  }
               },
               "InputMode": "string",
               "RecordWrapperType": "string",
               "ShuffleConfig": {
                  "Seed": number
               }
            }
         ],
         "LabelingJobArn": "string",
         "LastModifiedTime": number,
         "ModelArtifacts": {
            "S3ModelArtifacts": "string"
         },
         "OutputDataConfig": {
            "KmsKeyId": "string",
            "S3OutputPath": "string"
         },
         "ResourceConfig": {
            "InstanceCount": number,
            "InstanceType": "string",
```

```
            "VolumeKmsKeyId": "string",
            "VolumeSizeInGB": number
        },
        "RoleArn": "string",
        "SecondaryStatus": "string",
        "SecondaryStatusTransitions": [
            {
                "EndTime": number,
                "StartTime": number,
                "Status": "string",
                "StatusMessage": "string"
            }
        ],
        "StoppingCondition": {
            "MaxRuntimeInSeconds": number,
            "MaxWaitTimeInSeconds": number
        },
        "Tags": [
            {
                "Key": "string",
                "Value": "string"
            }
        ],
        "TrainingEndTime": number,
        "TrainingJobArn": "string",
        "TrainingJobName": "string",
        "TrainingJobStatus": "string",
        "TrainingStartTime": number,
        "TuningJobArn": "string",
        "VpcConfig": {
            "SecurityGroupIds": [ "string" ],
            "Subnets": [ "string" ]
        }
      }
    }
  ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 820)

If the result of the previous `Search` request was truncated, the response includes a NextToken. To retrieve the next set of results, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Results (p. 820)

A list of `SearchResult` objects.

Type: Array of SearchRecord (p. 998) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StartNotebookInstance

Service: Amazon SageMaker Service

Launches an ML compute instance with the latest version of the libraries and attaches your ML storage volume. After configuring the notebook instance, Amazon SageMaker sets the notebook instance status to `InService`. A notebook instance's status must be `InService` before you can connect to your Jupyter notebook.

Request Syntax

```
{
    "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 824)

The name of the notebook instance to start.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StopCompilationJob

Service: Amazon SageMaker Service

Stops a model compilation job.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal. This gracefully shuts the job down. If the job hasn't stopped, it sends the SIGKILL signal.

When it receives a `StopCompilationJob` request, Amazon SageMaker changes the CompilationJobSummary:CompilationJobStatus (p. 884) of the job to `Stopping`. After Amazon SageMaker stops the job, it sets the CompilationJobSummary:CompilationJobStatus (p. 884) to `Stopped`.

Request Syntax

```
{
    "CompilationJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

CompilationJobName (p. 826)

> The name of the model compilation job to stop.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`
>
> Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

> Resource being access is not found.
>
> HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StopHyperParameterTuningJob

Service: Amazon SageMaker Service

Stops a running hyperparameter tuning job and all running training jobs that the tuning job launched.

All model artifacts output from the training jobs are stored in Amazon Simple Storage Service (Amazon S3). All data that the training jobs write to Amazon CloudWatch Logs are still available in CloudWatch. After the tuning job moves to the `Stopped` state, it releases all reserved resources for the tuning job.

Request Syntax

```
{
    "HyperParameterTuningJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 828)

The name of the tuning job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StopLabelingJob

Service: Amazon SageMaker Service

Stops a running labeling job. A job that is stopped cannot be restarted. Any results obtained before the job is stopped are placed in the Amazon S3 output bucket.

Request Syntax

```
{
    "LabelingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

LabelingJobName (p. 830)

The name of the labeling job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java

- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StopNotebookInstance

Service: Amazon SageMaker Service

Terminates the ML compute instance. Before terminating the instance, Amazon SageMaker disconnects the ML storage volume from it. Amazon SageMaker preserves the ML storage volume. Amazon SageMaker stops charging you for the ML compute instance when you call `StopNotebookInstance`.

To access data on the ML storage volume for a notebook instance that has been terminated, call the `StartNotebookInstance` API. `StartNotebookInstance` launches another ML compute instance, configures it, and attaches the preserved ML storage volume so you can continue your work.

Request Syntax

```
{
    "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 832)

> The name of the notebook instance to terminate.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StopTrainingJob

Service: Amazon SageMaker Service

Stops a training job. To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts, so the results of the training is not lost.

When it receives a `StopTrainingJob` request, Amazon SageMaker changes the status of the job to `Stopping`. After Amazon SageMaker stops the job, it sets the status to `Stopped`.

Request Syntax

```
{
   "TrainingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

TrainingJobName (p. 834)

The name of the training job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StopTransformJob

Service: Amazon SageMaker Service

Stops a transform job.

When Amazon SageMaker receives a `StopTransformJob` request, the status of the job changes to `Stopping`. After Amazon SageMaker stops the job, the status is set to `Stopped`. When you stop a transform job before it is completed, Amazon SageMaker doesn't store the job's output in Amazon S3.

Request Syntax

```
{
    "TransformJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

TransformJobName (p. 836)

The name of the transform job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

UpdateCodeRepository

Service: Amazon SageMaker Service

Updates the specified Git repository with the specified values.

Request Syntax

```
{
    "CodeRepositoryName": "string",
    "GitConfig": {
        "SecretArn": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

CodeRepositoryName (p. 838)

> The name of the Git repository to update.
>
> Type: String
>
> Length Constraints: Minimum length of 1. Maximum length of 63.
>
> Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
>
> Required: Yes

GitConfig (p. 838)

> The configuration of the git repository, including the URL and the Amazon Resource Name (ARN)
> of the AWS Secrets Manager secret that contains the credentials used to access the repository. The
> secret must have a staging label of AWSCURRENT and must be in the following format:
>
> {"username": UserName, "password": Password}
>
> Type: GitConfigForUpdate (p. 906) object
>
> Required: No

Response Syntax

```
{
    "CodeRepositoryArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeRepositoryArn (p. 838)

> The ARN of the Git repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

UpdateEndpoint

Service: Amazon SageMaker Service

Deploys the new `EndpointConfig` specified in the request, switches to using newly created endpoint, and then deletes resources provisioned for the endpoint using the previous `EndpointConfig` (there is no availability loss).

When Amazon SageMaker receives the request, it sets the endpoint status to `Updating`. After updating the endpoint, it sets the status to `InService`. To check the status of an endpoint, use the DescribeEndpoint API.

> **Note**
> You must not delete an `EndpointConfig` in use by an endpoint that is live or while the `UpdateEndpoint` or `CreateEndpoint` operations are being performed on the endpoint. To update an endpoint, you must create a new `EndpointConfig`.

Request Syntax

```
{
    "EndpointConfigName": "string",
    "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

EndpointConfigName (p. 840)

The name of the new endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

EndpointName (p. 840)

The name of the endpoint whose configuration you want to update.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```
{
    "EndpointArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EndpointArn (p. 840)

The Amazon Resource Name (ARN) of the endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

UpdateEndpointWeightsAndCapacities

Service: Amazon SageMaker Service

Updates variant weight of one or more variants associated with an existing endpoint, or capacity of one variant associated with an existing endpoint. When it receives the request, Amazon SageMaker sets the endpoint status to `Updating`. After updating the endpoint, it sets the status to `InService`. To check the status of an endpoint, use the DescribeEndpoint API.

Request Syntax

```
{
   "DesiredWeightsAndCapacities": [
      {
         "DesiredInstanceCount": number,
         "DesiredWeight": number,
         "VariantName": "string"
      }
   ],
   "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

DesiredWeightsAndCapacities (p. 842)

An object that provides new capacity and weight values for a variant.

Type: Array of DesiredWeightAndCapacity (p. 895) objects

Array Members: Minimum number of 1 item.

Required: Yes

EndpointName (p. 842)

The name of an existing Amazon SageMaker endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```
{
   "EndpointArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EndpointArn (p. 842)

The Amazon Resource Name (ARN) of the updated endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

UpdateNotebookInstance

Service: Amazon SageMaker Service

Updates a notebook instance. NotebookInstance updates include upgrading or downgrading the ML compute instance used for your notebook instance to accommodate changes in your workload requirements.

Request Syntax

```
{
   "AcceleratorTypes": [ "string" ],
   "AdditionalCodeRepositories": [ "string" ],
   "DefaultCodeRepository": "string",
   "DisassociateAcceleratorTypes": boolean,
   "DisassociateAdditionalCodeRepositories": boolean,
   "DisassociateDefaultCodeRepository": boolean,
   "DisassociateLifecycleConfig": boolean,
   "InstanceType": "string",
   "LifecycleConfigName": "string",
   "NotebookInstanceName": "string",
   "RoleArn": "string",
   "RootAccess": "string",
   "VolumeSizeInGB": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1043).

The request accepts the following data in JSON format.

AcceleratorTypes (p. 844)

A list of the Elastic Inference (EI) instance types to associate with this notebook instance. Currently only one EI instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: `ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge`

Required: No

AdditionalCodeRepositories (p. 844)

An array of up to three Git repositories to associate with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

DefaultCodeRepository (p. 844)

The Git repository to associate with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

DisassociateAcceleratorTypes (p. 844)

A list of the Elastic Inference (EI) instance types to remove from this notebook instance. This operation is idempotent. If you specify an accelerator type that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

DisassociateAdditionalCodeRepositories (p. 844)

A list of names or URLs of the default Git repositories to remove from this notebook instance. This operation is idempotent. If you specify a Git repository that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

DisassociateDefaultCodeRepository (p. 844)

The name or URL of the default Git repository to remove from this notebook instance. This operation is idempotent. If you specify a Git repository that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

DisassociateLifecycleConfig (p. 844)

Set to `true` to remove the notebook instance lifecycle configuration currently associated with the notebook instance. This operation is idempotent. If you specify a lifecycle configuration that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

InstanceType (p. 844)

The Amazon ML compute instance type.

Type: String

Valid Values: `ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge`

```
| ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge
| ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge
| ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge |
ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge
| ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge
| ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge
```

Required: No

LifecycleConfigName (p. 844)

The name of a lifecycle configuration to associate with the notebook instance. For information about lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

NotebookInstanceName (p. 844)

The name of the notebook instance to update.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

RoleArn (p. 844)

The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker can assume to access the notebook instance. For more information, see Amazon SageMaker Roles.

> **Note**
> To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: No

RootAccess (p. 844)

Whether root access is enabled or disabled for users of the notebook instance. The default value is `Enabled`.

> **Note**
> If you set this to `Disabled`, users don't have root access on the notebook instance, but lifecycle configuration scripts still run with root permissions.

Type: String

Valid Values: `Enabled | Disabled`

Required: No

VolumeSizeInGB (p. 844)

The size, in GB, of the ML storage volume to attach to the notebook instance. The default value is 5 GB. ML storage volumes are encrypted, so Amazon SageMaker can't determine the amount of available free space on the volume. Because of this, you can increase the volume size when you update a notebook instance, but you can't decrease the volume size. If you want to decrease the size of the ML storage volume in use, create a new notebook instance with the desired size.

Type: Integer

Valid Range: Minimum value of 5. Maximum value of 16384.

Required: No

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

UpdateNotebookInstanceLifecycleConfig

Service: Amazon SageMaker Service

Updates a notebook instance lifecycle configuration created with the
CreateNotebookInstanceLifecycleConfig (p. 662) API.

Request Syntax

```
{
   "NotebookInstanceLifecycleConfigName": "string",
   "OnCreate": [
      {
         "Content": "string"
      }
   ],
   "OnStart": [
      {
         "Content": "string"
      }
   ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

NotebookInstanceLifecycleConfigName (p. 848)

> The name of the lifecycle configuration.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

OnCreate (p. 848)

> The shell script that runs only once, when you create a notebook instance. The shell script must be a
> base64-encoded string.
>
> Type: Array of NotebookInstanceLifecycleHook (p. 969) objects
>
> Array Members: Maximum number of 1 item.
>
> Required: No

OnStart (p. 848)

> The shell script that runs every time you start a notebook instance, including when you create the
> notebook instance. The shell script must be a base64-encoded string.
>
> Type: Array of NotebookInstanceLifecycleHook (p. 969) objects
>
> Array Members: Maximum number of 1 item.
>
> Required: No

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

UpdateWorkteam

Service: Amazon SageMaker Service

Updates an existing work team with new member definitions or description.

Request Syntax

```
{
   "Description": "string",
   "MemberDefinitions": [
      {
         "CognitoMemberDefinition": {
            "ClientId": "string",
            "UserGroup": "string",
            "UserPool": "string"
         }
      }
   ],
   "NotificationConfiguration": {
      "NotificationTopicArn": "string"
   },
   "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 1043).

The request accepts the following data in JSON format.

Description (p. 850)

An updated description for the work team.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `.+`

Required: No

MemberDefinitions (p. 850)

A list of `MemberDefinition` objects that contain the updated work team members.

Type: Array of MemberDefinition (p. 954) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: No

NotificationConfiguration (p. 850)

Configures SNS topic notifications for available or expiring work items

Type: NotificationConfiguration (p. 973) object

Required: No

The name of the work team to update.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```
{
   "Workteam": {
      "CreateDate": number,
      "Description": "string",
      "LastUpdatedDate": number,
      "MemberDefinitions": [
         {
            "CognitoMemberDefinition": {
               "ClientId": "string",
               "UserGroup": "string",
               "UserPool": "string"
            }
         }
      ],
      "NotificationConfiguration": {
         "NotificationTopicArn": "string"
      },
      "ProductListingIds": [ "string" ],
      "SubDomain": "string",
      "WorkteamArn": "string",
      "WorkteamName": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

A `Workteam` object that describes the updated work team.

Type: Workteam (p. 1040) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Amazon SageMaker Runtime

The following actions are supported by Amazon SageMaker Runtime:

- InvokeEndpoint (p. 853)

InvokeEndpoint

Service: Amazon SageMaker Runtime

After you deploy a model into production using Amazon SageMaker hosting services, your client applications use this API to get inferences from the model hosted at the specified endpoint.

For an overview of Amazon SageMaker, see How It Works.

Amazon SageMaker strips all POST headers except those supported by the API. Amazon SageMaker might add additional headers. You should not rely on the behavior of headers outside those enumerated in the request syntax.

Calls to `InvokeEndpoint` are authenticated by using AWS Signature Version 4. For information, see Authenticating Requests (AWS Signature Version 4) in the *Amazon S3 API Reference*.

A customer's model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the /invocations. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

> **Note**
> Endpoints are scoped to an individual account, and are not public. The URL does not contain the account ID, but Amazon SageMaker determines the account ID from the authentication token that is supplied by the caller.

Request Syntax

```
POST /endpoints/EndpointName/invocations HTTP/1.1
Content-Type: ContentType
Accept: Accept
X-Amzn-SageMaker-Custom-Attributes: CustomAttributes

Body
```

URI Request Parameters

The request requires the following URI parameters.

Accept (p. 853)

The desired MIME type of the inference in the response.

Length Constraints: Maximum length of 1024.

Pattern: `\p{ASCII}*`

ContentType (p. 853)

The MIME type of the input data in the request body.

Length Constraints: Maximum length of 1024.

Pattern: `\p{ASCII}*`

CustomAttributes (p. 853)

Provides additional information about a request for an inference submitted to a model hosted at an Amazon SageMaker endpoint. The information is an opaque value that is forwarded verbatim. You could use this value, for example, to provide an ID that you can use to track a request or to provide other metadata that a service endpoint was programmed to process. The value must consist of no more than 1024 visible US-ASCII characters as specified in Section 3.3.6. Field Value Components of

the Hypertext Transfer Protocol (HTTP/1.1). This feature is currently supported in the AWS SDKs but not in the Amazon SageMaker Python SDK.

Length Constraints: Maximum length of 1024.

Pattern: `\p{ASCII}*`

EndpointName (p. 853)

The name of the endpoint that you specified when you created the endpoint using the CreateEndpoint API.

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Request Body

The request accepts the following binary data.

Body (p. 853)

Provides input data, in the format specified in the `ContentType` request header. Amazon SageMaker passes all of the data in the body to the model.

For information about the format of the request body, see Common Data Formats—Inference.

Length Constraints: Maximum length of 5242880.

Response Syntax

```
HTTP/1.1 200
Content-Type: ContentType
x-Amzn-Invoked-Production-Variant: InvokedProductionVariant
X-Amzn-SageMaker-Custom-Attributes: CustomAttributes

Body
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The response returns the following HTTP headers.

ContentType (p. 854)

The MIME type of the inference returned in the response body.

Length Constraints: Maximum length of 1024.

Pattern: `\p{ASCII}*`

CustomAttributes (p. 854)

Provides additional information in the response about the inference returned by a model hosted at an Amazon SageMaker endpoint. The information is an opaque value that is forwarded verbatim. You could use this value, for example, to return an ID received in the `CustomAttributes` header of a request or other metadata that a service endpoint was programmed to produce. The value must consist of no more than 1024 visible US-ASCII characters as specified in Section 3.3.6. Field

Value Components of the Hypertext Transfer Protocol (HTTP/1.1). If the customer wants the custom attribute returned, the model must set the custom attribute to be included on the way back.

This feature is currently supported in the AWS SDKs but not in the Amazon SageMaker Python SDK.

Length Constraints: Maximum length of 1024.

Pattern: `\p{ASCII}*`

InvokedProductionVariant (p. 854)

Identifies the production variant that was invoked.

Length Constraints: Maximum length of 1024.

Pattern: `\p{ASCII}*`

The response returns the following as the HTTP body.

Body (p. 854)

Includes the inference provided by the model.

For information about the format of the response body, see Common Data Formats—Inference.

Length Constraints: Maximum length of 5242880.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1041).

InternalFailure

An internal failure occurred.

HTTP Status Code: 500

ModelError

Model (owned by the customer in the container) returned 4xx or 5xx error code.

HTTP Status Code: 424

ServiceUnavailable

The service is unavailable. Try your call again.

HTTP Status Code: 503

ValidationError

Inspect your request and try again.

HTTP Status Code: 400

Example

Pass a trace ID in the CustomAttribute of a request and return it in the CustomAttribute of the response.

In this example a trace ID is passed to the service endpoint in the `CustomAttributes` header of the request and then retrieved and returned in the `CustomAttributes` header of the response.

Sample Request

```
import boto3

client = boto3.client('sagemaker-runtime')

custom_attributes = "c000b4f9-df62-4c85-a0bf-7c525f9104a4"   # An example of a trace ID.
endpoint_name = "..."                                         # Your endpoint name.
content_type = "..."                                          # The MIME type of the input
 data in the request body.
accept = "..."                                                # The desired MIME type of the
 inference in the response.
payload = "..."                                               # Payload for inference.
```

Sample Response

```
response = client.invoke_endpoint(
    EndpointName=endpoint_name,
    CustomAttributes=custom_attributes,
    ContentType=content_type,
    Accept=accept,
    Body=payload
    )

print(response['CustomAttributes'])                          # If model receives and updates
 the custom_attributes header

                                                             # by adding "Trace id: " in

 front of custom_attributes in the request,

                                                             # custom_attributes in response

 becomes

                                                             # "Trace ID: c000b4f9-
df62-4c85-a0bf-7c525f9104a4"
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Data Types

The following data types are supported by Amazon SageMaker Service:

- AlgorithmSpecification (p. 863)

The following data types are supported by Amazon SageMaker Runtime:

Amazon SageMaker Service

The following data types are supported by Amazon SageMaker Service:

- UiTemplate (p. 1037)
- USD (p. 1038)
- VpcConfig (p. 1039)
- Workteam (p. 1040)

AlgorithmSpecification

Service: Amazon SageMaker Service

Specifies the training algorithm to use in a CreateTrainingJob request.

For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about using your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Contents

AlgorithmName

The name of the algorithm resource to use for the training job. This must be an algorithm resource that you created or subscribe to on AWS Marketplace. If you specify a value for this parameter, you can't specify a value for TrainingImage.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\/)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?<!-)$`

Required: No

MetricDefinitions

A list of metric definition objects. Each object specifies the metric name and regular expressions used to parse algorithm logs. Amazon SageMaker publishes each metric to Amazon CloudWatch.

Type: Array of MetricDefinition (p. 956) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

Required: No

TrainingImage

The registry path of the Docker image that contains the training algorithm. For information about docker registry paths for built-in algorithms, see Algorithms Provided by Amazon SageMaker: Common Parameters. Amazon SageMaker supports both `registry/repository[:tag]` and `registry/repository[@digest]` image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `.*`

Required: No

TrainingInputMode

The input mode that the algorithm supports. For the input modes that Amazon SageMaker algorithms support, see Algorithms. If an algorithm supports the File input mode, Amazon SageMaker downloads the training data from S3 to the provisioned ML storage Volume, and mounts the directory to docker volume for training container. If an algorithm supports the Pipe input mode, Amazon SageMaker streams data directly from S3 to the container.

In File mode, make sure you provision ML storage volume with sufficient capacity to accommodate the data download from S3. In addition to the training data, the ML storage volume also stores

the output model. The algorithm container use ML storage volume to also store intermediate information, if any.

For distributed algorithms using File mode, training data is distributed uniformly, and your training duration is predictable if the input data objects size is approximately same. Amazon SageMaker does not split the files any further for model training. If the object sizes are skewed, training won't be optimal as the data distribution is also skewed where one host in a training cluster is overloaded, thus becoming bottleneck in training.

Type: String

Valid Values: `Pipe | File`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

AlgorithmStatusDetails

Service: Amazon SageMaker Service

Specifies the validation and image scan statuses of the algorithm.

Contents

ImageScanStatuses

The status of the scan of the algorithm's Docker image container.

Type: Array of AlgorithmStatusItem (p. 866) objects

Required: No

ValidationStatuses

The status of algorithm validation.

Type: Array of AlgorithmStatusItem (p. 866) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

AlgorithmStatusItem

Service: Amazon SageMaker Service

Represents the overall status of an algorithm.

Contents

FailureReason

if the overall status is `Failed`, the reason for the failure.

Type: String

Required: No

Name

The name of the algorithm for which the overall status is being reported.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Status

The current status.

Type: String

Valid Values: `NotStarted | InProgress | Completed | Failed`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

AlgorithmSummary

Service: Amazon SageMaker Service

Provides summary information about an algorithm.

Contents

AlgorithmArn

The Amazon Resource Name (ARN) of the algorithm.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:algorithm/.*`

Required: Yes

AlgorithmDescription

A brief description of the algorithm.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

Required: No

AlgorithmName

The name of the algorithm that is described by the summary.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

AlgorithmStatus

The overall status of the algorithm.

Type: String

Valid Values: `Pending | InProgress | Completed | Failed | Deleting`

Required: Yes

CreationTime

A timestamp that shows when the algorithm was created.

Type: Timestamp

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

AlgorithmValidationProfile

Service: Amazon SageMaker Service

Defines a training job and a batch transform job that Amazon SageMaker runs to validate your algorithm.

The data provided in the validation profile is made available to your buyers on AWS Marketplace.

Contents

ProfileName

The name of the profile for the algorithm. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

TrainingJobDefinition

The `TrainingJobDefinition` object that describes the training job that Amazon SageMaker runs to validate your algorithm.

Type: TrainingJobDefinition (p. 1015) object

Required: Yes

TransformJobDefinition

The `TransformJobDefinition` object that describes the transform job that Amazon SageMaker runs to validate your algorithm.

Type: TransformJobDefinition (p. 1026) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

AlgorithmValidationSpecification

Service: Amazon SageMaker Service

Specifies configurations for one or more training jobs that Amazon SageMaker runs to test the algorithm.

Contents

ValidationProfiles

An array of `AlgorithmValidationProfile` objects, each of which specifies a training job and batch transform job that Amazon SageMaker runs to validate your algorithm.

Type: Array of AlgorithmValidationProfile (p. 869) objects

Array Members: Fixed number of 1 item.

Required: Yes

ValidationRole

The IAM roles that Amazon SageMaker uses to run the training jobs.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

AnnotationConsolidationConfig

Service: Amazon SageMaker Service

Configures how labels are consolidated across human workers.

Contents

AnnotationConsolidationLambdaArn

The Amazon Resource Name (ARN) of a Lambda function implements the logic for annotation consolidation.

For the built-in bounding box, image classification, semantic segmentation, and text classification task types, Amazon SageMaker Ground Truth provides the following Lambda functions:

- *Bounding box* - Finds the most similar boxes from different workers based on the Jaccard index of the boxes.

  ```
  arn:aws:lambda:us-east-1:432418664414:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:us-east-2:266458841044:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:us-west-2:081040173940:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:eu-west-1:568282634449:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:ap-southeast-2:454466003867:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:ap-south-1:565803892007:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:eu-central-1:203001061592:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:ap-northeast-2:845288260483:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:eu-west-2:487402164563:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:ap-southeast-1:377565633583:function:ACS-BoundingBox
  ```

  ```
  arn:aws:lambda:ca-central-1:918755190332:function:ACS-BoundingBox
  ```

- *Image classification* - Uses a variant of the Expectation Maximization approach to estimate the true class of an image based on annotations from individual workers.

  ```
  arn:aws:lambda:us-east-1:432418664414:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:us-east-2:266458841044:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:us-west-2:081040173940:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:eu-west-1:568282634449:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:ap-southeast-2:454466003867:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:ap-south-1:565803892007:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:eu-central-1:203001061592:function:ACS-ImageMultiClass
  ```

  ```
  arn:aws:lambda:ap-northeast-2:845288260483:function:ACS-ImageMultiClass
  ```

```
arn:aws:lambda:eu-west-2:487402164563:function:ACS-ImageMultiClass
```

```
arn:aws:lambda:ap-southeast-1:377565633583:function:ACS-ImageMultiClass
```

```
arn:aws:lambda:ca-central-1:918755190332:function:ACS-ImageMultiClass
```

- *Semantic segmentation* - Treats each pixel in an image as a multi-class classification and treats pixel annotations from workers as "votes" for the correct label.

```
arn:aws:lambda:us-east-1:432418664414:function:ACS-SemanticSegmentation
```

```
arn:aws:lambda:us-east-2:266458841044:function:ACS-SemanticSegmentation
```

```
arn:aws:lambda:us-west-2:081040173940:function:ACS-SemanticSegmentation
```

```
arn:aws:lambda:eu-west-1:568282634449:function:ACS-SemanticSegmentation
```

```
arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-
SemanticSegmentation
```

```
arn:aws:lambda:ap-southeast-2:454466003867:function:ACS-
SemanticSegmentation
```

```
arn:aws:lambda:ap-south-1:565803892007:function:ACS-SemanticSegmentation
```

```
arn:aws:lambda:eu-central-1:203001061592:function:ACS-SemanticSegmentation
```

```
arn:aws:lambda:ap-northeast-2:845288260483:function:ACS-
SemanticSegmentation
```

```
arn:aws:lambda:eu-west-2:487402164563:function:ACS-SemanticSegmentation
```

```
arn:aws:lambda:ap-southeast-1:377565633583:function:ACS-
SemanticSegmentation
```

```
arn:aws:lambda:ca-central-1:918755190332:function:ACS-SemanticSegmentation
```

- *Text classification* - Uses a variant of the Expectation Maximization approach to estimate the true class of text based on annotations from individual workers.

```
arn:aws:lambda:us-east-1:432418664414:function:ACS-TextMultiClass
```

```
arn:aws:lambda:us-east-2:266458841044:function:ACS-TextMultiClass
```

```
arn:aws:lambda:us-west-2:081040173940:function:ACS-TextMultiClass
```

```
arn:aws:lambda:eu-west-1:568282634449:function:ACS-TextMultiClass
```

```
arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-TextMultiClass
```

```
arn:aws:lambda:ap-southeast-2:454466003867:function:ACS-TextMultiClass
```

```
arn:aws:lambda:ap-south-1:565803892007:function:ACS-TextMultiClass
```

```
arn:aws:lambda:eu-central-1:203001061592:function:ACS-TextMultiClass
```

```
arn:aws:lambda:ap-northeast-2:845288260483:function:ACS-TextMultiClass
```

```
arn:aws:lambda:eu-west-2:487402164563:function:ACS-TextMultiClass
```

```
arn:aws:lambda:ap-southeast-1:377565633583:function:ACS-TextMultiClass
```

```
arn:aws:lambda:ca-central-1:918755190332:function:ACS-TextMultiClass
```

- *Named entity eecognition* - Groups similar selections and calculates aggregate boundaries, resolving to most-assigned label.

```
arn:aws:lambda:us-east-1:432418664414:function:ACS-NamedEntityRecognition
```

```
arn:aws:lambda:us-east-2:266458841044:function:ACS-NamedEntityRecognition
```

```
arn:aws:lambda:us-west-2:081040173940:function:ACS-NamedEntityRecognition
```

```
arn:aws:lambda:eu-west-1:568282634449:function:ACS-NamedEntityRecognition
```

```
arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-
NamedEntityRecognition
```

```
arn:aws:lambda:ap-southeast-2:454466003867:function:ACS-
NamedEntityRecognition
```

```
arn:aws:lambda:ap-south-1:565803892007:function:ACS-NamedEntityRecognition
```

```
arn:aws:lambda:eu-central-1:203001061592:function:ACS-
NamedEntityRecognition
```

```
arn:aws:lambda:ap-northeast-2:845288260483:function:ACS-
NamedEntityRecognition
```

```
arn:aws:lambda:eu-west-2:487402164563:function:ACS-NamedEntityRecognition
```

```
arn:aws:lambda:ap-southeast-1:377565633583:function:ACS-
NamedEntityRecognition
```

```
arn:aws:lambda:ca-central-1:918755190332:function:ACS-
NamedEntityRecognition
```

For more information, see Annotation Consolidation.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:lambda:[a-z]{2}-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

CategoricalParameterRange

Service: Amazon SageMaker Service

A list of categorical hyperparameters to tune.

Contents

Name

The name of the categorical hyperparameter to tune.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

Values

A list of the categories for the hyperparameter.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

CategoricalParameterRangeSpecification

Service: Amazon SageMaker Service

Defines the possible values for a categorical hyperparameter.

Contents

Values

The allowed categories for the hyperparameter.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

Channel

Service: Amazon SageMaker Service

A channel is a named input source that training algorithms can consume.

Contents

ChannelName

The name of the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[A-Za-z0-9\.\-_]+`

Required: Yes

CompressionType

If training data is compressed, the compression type. The default value is `None`. `CompressionType` is used only in Pipe input mode. In File mode, leave this field unset or set it to None.

Type: String

Valid Values: `None | Gzip`

Required: No

ContentType

The MIME type of the data.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: No

DataSource

The location of the channel data.

Type: DataSource (p. 893) object

Required: Yes

InputMode

(Optional) The input mode to use for the data channel in a training job. If you don't set a value for `InputMode`, Amazon SageMaker uses the value set for `TrainingInputMode`. Use this parameter to override the `TrainingInputMode` setting in a AlgorithmSpecification (p. 863) request when you have a channel that needs a different input mode from the training job's general setting. To download the data from Amazon Simple Storage Service (Amazon S3) to the provisioned ML storage volume, and mount the directory to a Docker volume, use `File` input mode. To stream data directly from Amazon S3 to the container, choose `Pipe` input mode.

To use a model for incremental training, choose `File` input model.

Type: String

Valid Values: `Pipe | File`

Required: No

RecordWrapperType

Specify RecordIO as the value when input data is in raw format but the training algorithm requires the RecordIO format. In this case, Amazon SageMaker wraps each individual S3 object in a RecordIO record. If the input data is already in RecordIO format, you don't need to set this attribute. For more information, see Create a Dataset Using RecordIO.

In File mode, leave this field unset or set it to None.

Type: String

Valid Values: `None | RecordIO`

Required: No

ShuffleConfig

A configuration for a shuffle option for input data in a channel. If you use `S3Prefix` for `S3DataType`, this shuffles the results of the S3 key prefix matches. If you use `ManifestFile`, the order of the S3 object references in the `ManifestFile` is shuffled. If you use `AugmentedManifestFile`, the order of the JSON lines in the `AugmentedManifestFile` is shuffled. The shuffling order is determined using the `Seed` value.

For Pipe input mode, shuffling is done at the start of every epoch. With large datasets this ensures that the order of the training data is different for each epoch, it helps reduce bias and possible overfitting. In a multi-node training job when ShuffleConfig is combined with `S3DataDistributionType` of `ShardedByS3Key`, the data is shuffled across nodes so that the content sent to a particular node on the first epoch might be sent to a different node on the second epoch.

Type: ShuffleConfig (p. 1001) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ChannelSpecification

Service: Amazon SageMaker Service

Defines a named input source, called a channel, to be used by an algorithm.

Contents

Description

A brief description of the channel.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

Required: No

IsRequired

Indicates whether the channel is required by the algorithm.

Type: Boolean

Required: No

Name

The name of the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[A-Za-z0-9\.\-_]+`

Required: Yes

SupportedCompressionTypes

The allowed compression types, if data compression is used.

Type: Array of strings

Valid Values: `None | Gzip`

Required: No

SupportedContentTypes

The supported MIME types for the data.

Type: Array of strings

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

SupportedInputModes

The allowed input mode, either FILE or PIPE.

In FILE mode, Amazon SageMaker copies the data from the input source onto the local Amazon Elastic Block Store (Amazon EBS) volumes before starting your training algorithm. This is the most commonly used input mode.

In PIPE mode, Amazon SageMaker streams input data from the source directly to your algorithm without using the EBS volume.

Type: Array of strings

Array Members: Minimum number of 1 item.

Valid Values: `Pipe | File`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

CheckpointConfig

Service: Amazon SageMaker Service

Contains information about the output location for managed spot training checkpoint data.

Contents

LocalPath

(Optional) The local directory where checkpoints are written. The default directory is `/opt/ml/checkpoints/`.

Type: String

Length Constraints: Maximum length of 4096.

Pattern: `.*`

Required: No

S3Uri

Identifies the S3 path where you want Amazon SageMaker to store checkpoints. For example, `s3://bucket-name/key-name-prefix`.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

CodeRepositorySummary

Service: Amazon SageMaker Service

Specifies summary information about a Git repository.

Contents

CodeRepositoryArn

The Amazon Resource Name (ARN) of the Git repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*`

Required: Yes

CodeRepositoryName

The name of the Git repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

CreationTime

The date and time that the Git repository was created.

Type: Timestamp

Required: Yes

GitConfig

Configuration details for the Git repository, including the URL where it is located and the ARN of the AWS Secrets Manager secret that contains the credentials used to access the repository.

Type: GitConfig (p. 905) object

Required: No

LastModifiedTime

The date and time that the Git repository was last modified.

Type: Timestamp

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

CognitoMemberDefinition

Service: Amazon SageMaker Service

Identifies a Amazon Cognito user group. A user group can be used in on or more work teams.

Contents

ClientId

An identifier for an application client. You must create the app client ID using Amazon Cognito.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[\w+]+`

Required: Yes

UserGroup

An identifier for a user group.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[\p{L}\p{M}\p{S}\p{N}\p{P}]+`

Required: Yes

UserPool

An identifier for a user pool. The user pool must be in the same region as the service that you are calling.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 55.

Pattern: `[\w-]+_[0-9a-zA-Z]+`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

CompilationJobSummary

Service: Amazon SageMaker Service

A summary of a model compilation job.

Contents

CompilationEndTime

The time when the model compilation job completed.

Type: Timestamp

Required: No

CompilationJobArn

The Amazon Resource Name (ARN) of the model compilation job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*`

Required: Yes

CompilationJobName

The name of the model compilation job that you want a summary for.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

CompilationJobStatus

The status of the model compilation job.

Type: String

Valid Values: `INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED`

Required: Yes

CompilationStartTime

The time when the model compilation job started.

Type: Timestamp

Required: No

CompilationTargetDevice

The type of device that the model will run on after compilation has completed.

Type: String

Valid Values: `lambda | ml_m4 | ml_m5 | ml_c4 | ml_c5 | ml_p2 | ml_p3 | jetson_tx1 | jetson_tx2 | jetson_nano | rasp3b | deeplens | rk3399 | rk3288 | aisage | sbe_c | qcs605 | qcs603`

Required: Yes

CreationTime

The time when the model compilation job was created.

Type: Timestamp

Required: Yes

LastModifiedTime

The time when the model compilation job was last modified.

Type: Timestamp

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ContainerDefinition

Service: Amazon SageMaker Service

Describes the container, as part of model definition.

Contents

ContainerHostname

This parameter is ignored for models that contain only a `PrimaryContainer`.

When a `ContainerDefinition` is part of an inference pipeline, the value of ths parameter uniquely identifies the container for the purposes of logging and metrics. For information, see Use Logs and Metrics to Monitor an Inference Pipeline. If you don't specify a value for this parameter for a `ContainerDefinition` that is part of an inference pipeline, a unique name is automatically assigned based on the position of the `ContainerDefinition` in the pipeline. If you specify a value for the `ContainerHostName` for any `ContainerDefinition` that is part of an inference pipeline, you must specify a value for the `ContainerHostName` parameter of every `ContainerDefinition` in that pipeline.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

Environment

The environment variables to set in the Docker container. Each key and value in the `Environment` string to string map can have length of up to 1024. We support up to 16 entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: `[a-zA-Z_][a-zA-Z0-9_]*`

Value Length Constraints: Maximum length of 1024.

Value Pattern: `[\S\s]*`

Required: No

Image

The Amazon EC2 Container Registry (Amazon ECR) path where inference code is stored. If you are using your own custom algorithm instead of an algorithm provided by Amazon SageMaker, the inference code must meet Amazon SageMaker requirements. Amazon SageMaker supports both `registry/repository[:tag]` and `registry/repository[@digest]` image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker

Type: String

Length Constraints: Maximum length of 255.

Pattern: `[\S]+`

Required: No

ModelDataUrl

The S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix). The S3 path is required for Amazon SageMaker built-in algorithms, but not if you use your own algorithms. For more information on built-in algorithms, see Common Parameters.

If you provide a value for this parameter, Amazon SageMaker uses AWS Security Token Service to download model artifacts from the S3 path you provide. AWS STS is activated in your IAM user account by default. If you previously deactivated AWS STS for a region, you need to reactivate AWS STS for that region. For more information, see Activating and Deactivating AWS STS in an AWS Region in the *AWS Identity and Access Management User Guide*.

> **Important**
> If you use a built-in algorithm to create a model, Amazon SageMaker requires that you provide a S3 path to the model artifacts in `ModelDataUrl`.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: No

ModelPackageName

The name or Amazon Resource Name (ARN) of the model package to use to create the model.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\/)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?<!-)$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ContinuousParameterRange

Service: Amazon SageMaker Service

A list of continuous hyperparameters to tune.

Contents

MaxValue

The maximum value for the hyperparameter. The tuning job uses floating-point values between `MinValue` value and this value for tuning.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

MinValue

The minimum value for the hyperparameter. The tuning job uses floating-point values between this value and `MaxValue`for tuning.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

Name

The name of the continuous hyperparameter to tune.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

ScalingType

The scale that hyperparameter tuning uses to search the hyperparameter range. For information about choosing a hyperparameter scale, see Hyperparameter Scaling. One of the following values:

Auto

Amazon SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.

Linear

Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale.

Logarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a logarithmic scale.

Logarithmic scaling works only for ranges that have only values greater than 0.

ReverseLogarithmic

Hyperparemeter tuning searches the values in the hyperparameter range by using a reverse logarithmic scale.

Reverse logarithmic scaling works only for ranges that are entirely within the range 0<=x<1.0.

Type: String

Valid Values: `Auto | Linear | Logarithmic | ReverseLogarithmic`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ContinuousParameterRangeSpecification

Service: Amazon SageMaker Service

Defines the possible values for a continuous hyperparameter.

Contents

MaxValue

The maximum floating-point value allowed.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

MinValue

The minimum floating-point value allowed.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

DataProcessing

Service: Amazon SageMaker Service

The data structure used to specify the data to be used for inference in a batch transform job and to associate the data that is relevant to the prediction results in the output. The input filter provided allows you to exclude input data that is not needed for inference in a batch transform job. The output filter provided allows you to include input data relevant to interpreting the predictions in the output from the job. For more information, see Associate Prediction Results with their Corresponding Input Records.

Contents

InputFilter

A JSONPath expression used to select a portion of the input data to pass to the algorithm. Use the `InputFilter` parameter to exclude fields, such as an ID column, from the input. If you want Amazon SageMaker to pass the entire input dataset to the algorithm, accept the default value `$`.

Examples: `"$"`, `"$[1:]"`, `"$.features"`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 63.

Required: No

JoinSource

Specifies the source of the data to join with the transformed data. The valid values are `None` and `Input` The default value is `None` which specifies not to join the input with the transformed data. If you want the batch transform job to join the original input data with the transformed data, set `JoinSource` to `Input`.

For JSON or JSONLines objects, such as a JSON array, Amazon SageMaker adds the transformed data to the input JSON object in an attribute called `SageMakerOutput`. The joined result for JSON must be a key-value pair object. If the input is not a key-value pair object, Amazon SageMaker creates a new JSON file. In the new JSON file, and the input data is stored under the `SageMakerInput` key and the results are stored in `SageMakerOutput`.

For CSV files, Amazon SageMaker combines the transformed data with the input data at the end of the input data and stores it in the output file. The joined data has the joined input data followed by the transformed data and the output is a CSV file.

Type: String

Valid Values: `Input | None`

Required: No

OutputFilter

A JSONPath expression used to select a portion of the joined dataset to save in the output file for a batch transform job. If you want Amazon SageMaker to store the entire input dataset in the output file, leave the default value, `$`. If you specify indexes that aren't within the dimension size of the joined dataset, you get an error.

Examples: `"$"`, `"$[0,5:]"`, `"$['id','SageMakerOutput']"`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 63.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

DataSource

Service: Amazon SageMaker Service

Describes the location of the channel data.

Contents

FileSystemDataSource

The file system that is associated with a channel.

Type: FileSystemDataSource (p. 899) object

Required: No

S3DataSource

The S3 location of the data source that is associated with a channel.

Type: S3DataSource (p. 994) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

DeployedImage

Service: Amazon SageMaker Service

Gets the Amazon EC2 Container Registry path of the docker image of the model that is hosted in this ProductionVariant (p. 981).

If you used the `registry/repository[:tag]` form to specify the image path of the primary container when you created the model hosted in this `ProductionVariant`, the path resolves to a path of the form `registry/repository[@digest]`. A digest is a hash value that identifies a specific version of an image. For information about Amazon ECR paths, see Pulling an Image in the *Amazon ECR User Guide*.

Contents

ResolutionTime

The date and time when the image path for the model resolved to the `ResolvedImage`

Type: Timestamp

Required: No

ResolvedImage

The specific digest path of the image hosted in this `ProductionVariant`.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `[\S]+`

Required: No

SpecifiedImage

The image path you specified when you created the model.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `[\S]+`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

DesiredWeightAndCapacity

Service: Amazon SageMaker Service

Specifies weight and capacity values for a production variant.

Contents

DesiredInstanceCount

> The variant's capacity.
>
> Type: Integer
>
> Valid Range: Minimum value of 1.
>
> Required: No

DesiredWeight

> The variant's weight.
>
> Type: Float
>
> Valid Range: Minimum value of 0.
>
> Required: No

VariantName

> The name of the variant to update.
>
> Type: String
>
> Length Constraints: Maximum length of 63.
>
> Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
>
> Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

EndpointConfigSummary

Service: Amazon SageMaker Service

Provides summary information for an endpoint configuration.

Contents

CreationTime

A timestamp that shows when the endpoint configuration was created.

Type: Timestamp

Required: Yes

EndpointConfigArn

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint-config/.*`

Required: Yes

EndpointConfigName

The name of the endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

EndpointSummary

Service: Amazon SageMaker Service

Provides summary information for an endpoint.

Contents

CreationTime

A timestamp that shows when the endpoint was created.

Type: Timestamp

Required: Yes

EndpointArn

The Amazon Resource Name (ARN) of the endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*`

Required: Yes

EndpointName

The name of the endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

EndpointStatus

The status of the endpoint.

- `OutOfService`: Endpoint is not available to take incoming requests.
- `Creating`: CreateEndpoint (p. 632) is executing.
- `Updating`: UpdateEndpoint (p. 840) or UpdateEndpointWeightsAndCapacities (p. 842) is executing.
- `SystemUpdating`: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This maintenance operation does not change any customer-specified values such as VPC config, KMS encryption, model, instance type, or instance count.
- `RollingBack`: Endpoint fails to scale up or down or change its variant weight and is in the process of rolling back to its previous configuration. Once the rollback completes, endpoint returns to an `InService` status. This transitional status only applies to an endpoint that has autoscaling enabled and is undergoing variant weight or capacity changes as part of an UpdateEndpointWeightsAndCapacities (p. 842) call or when the UpdateEndpointWeightsAndCapacities (p. 842) operation is called explicitly.
- `InService`: Endpoint is available to process incoming requests.
- `Deleting`: DeleteEndpoint (p. 683) is executing.
- `Failed`: Endpoint could not be created, updated, or re-scaled. Use DescribeEndpoint:FailureReason (p. 711) for information about the failure. DeleteEndpoint (p. 683) is the only operation that can be performed on a failed endpoint.

To get a list of endpoints with a specified status, use the filter.

Type: String

Valid Values: `OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed`

Required: Yes

LastModifiedTime

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

FileSystemDataSource

Service: Amazon SageMaker Service

Specifies a file system data source for a channel.

Contents

DirectoryPath

The full path to the directory to associate with the channel.

Type: String

Length Constraints: Maximum length of 4096.

Pattern: `.*`

Required: Yes

FileSystemAccessMode

The access mode of the mount of the directory associated with the channel. A directory can be mounted either in `ro` (read-only) or `rw` (read-write) mode.

Type: String

Valid Values: `rw | ro`

Required: Yes

FileSystemId

The file system id.

Type: String

Length Constraints: Minimum length of 11.

Pattern: `.*`

Required: Yes

FileSystemType

The file system type.

Type: String

Valid Values: `EFS | FSxLustre`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

Filter

Service: Amazon SageMaker Service

A conditional statement for a search expression that includes a Boolean operator, a resource property, and a value.

If you don't specify an `Operator` and a `Value`, the filter searches for only the specified property. For example, defining a `Filter` for the `FailureReason` for the `TrainingJob Resource` searches for training job objects that have a value in the `FailureReason` field.

If you specify a `Value`, but not an `Operator`, Amazon SageMaker uses the equals operator as the default.

In search, there are several property types:

Metrics

> To define a metric filter, enter a value using the form `"Metrics.<name>"`, where <name> is a metric name. For example, the following filter searches for training jobs with an `"accuracy"` metric greater than `"0.9"`:
>
> {
>
> "Name": "Metrics.accuracy",
>
> "Operator": "GREATER_THAN",
>
> "Value": "0.9"
>
> }

HyperParameters

> To define a hyperparameter filter, enter a value with the form `"HyperParameters.<name>"`. Decimal hyperparameter values are treated as a decimal in a comparison if the specified `Value` is also a decimal value. If the specified `Value` is an integer, the decimal hyperparameter values are treated as integers. For example, the following filter is satisfied by training jobs with a `"learning_rate"` hyperparameter that is less than `"0.5"`:
>
> {
>
> "Name": "HyperParameters.learning_rate",
>
> "Operator": "LESS_THAN",
>
> "Value": "0.5"
>
> }

Tags

> To define a tag filter, enter a value with the form `"Tags.<key>"`.

Contents

Name

> A property name. For example, `TrainingJobName`. For the list of valid property names returned in a search result for each supported resource, see TrainingJob (p. 1009) properties. You must specify a valid property name for the resource.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: Yes

Operator

A Boolean binary operator that is used to evaluate the filter. The operator field contains one of the following values:

Equals

The specified resource in `Name` equals the specified `Value`.

NotEquals

The specified resource in `Name` does not equal the specified `Value`.

GreaterThan

The specified resource in `Name` is greater than the specified `Value`. Not supported for text-based properties.

GreaterThanOrEqualTo

The specified resource in `Name` is greater than or equal to the specified `Value`. Not supported for text-based properties.

LessThan

The specified resource in `Name` is less than the specified `Value`. Not supported for text-based properties.

LessThanOrEqualTo

The specified resource in `Name` is less than or equal to the specified `Value`. Not supported for text-based properties.

Contains

Only supported for text-based properties. The word-list of the property contains the specified `Value`.

If you have specified a filter `Value`, the default is `Equals`.

Type: String

Valid Values: `Equals | NotEquals | GreaterThan | GreaterThanOrEqualTo | LessThan | LessThanOrEqualTo | Contains`

Required: No

Value

A value used with `Resource` and `Operator` to determine if objects satisfy the filter's condition. For numerical properties, `Value` must be an integer or floating-point decimal. For timestamp properties, `Value` must be an ISO 8601 date-time string of the following format: `YYYY-mm-dd'T'HH:MM:SS`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `.+`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

FinalHyperParameterTuningJobObjectiveMetric

Service: Amazon SageMaker Service

Shows the final value for the objective metric for a training job that was launched by a hyperparameter tuning job. You define the objective metric in the `HyperParameterTuningJobObjective` parameter of HyperParameterTuningJobConfig (p. 922).

Contents

MetricName

The name of the objective metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: Yes

Type

Whether to minimize or maximize the objective metric. Valid values are Minimize and Maximize.

Type: String

Valid Values: `Maximize | Minimize`

Required: No

Value

The value of the objective metric.

Type: Float

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

GitConfig

Service: Amazon SageMaker Service

Specifies configuration details for a Git repository in your AWS account.

Contents

Branch

The default branch for the Git repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `[^ ~^:?*\[ ]+`

Required: No

RepositoryUrl

The URL where the Git repository is located.

Type: String

Pattern: `^https://([^/]+)/?(.*)$`

Required: Yes

SecretArn

The Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the git repository. The secret must have a staging label of `AWSCURRENT` and must be in the following format:

`{"username": UserName, "password": Password}`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:secretsmanager:[a-z0-9\-]*:[0-9]{12}:secret:.*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

GitConfigForUpdate

Service: Amazon SageMaker Service

Specifies configuration details for a Git repository when the repository is updated.

Contents

SecretArn

The Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the git repository. The secret must have a staging label of `AWSCURRENT` and must be in the following format:

`{"username": `*`UserName`*`, "password": `*`Password`*`}`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:secretsmanager:[a-z0-9\-]*:[0-9]{12}:secret:.*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HumanTaskConfig

Service: Amazon SageMaker Service

Information required for human workers to complete a labeling task.

Contents

AnnotationConsolidationConfig

Configures how labels are consolidated across human workers.

Type: object

Required: Yes

MaxConcurrentTaskCount

Defines the maximum number of data objects that can be labeled by human workers at the same time. Each object may have more than one worker at one time.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

NumberOfHumanWorkersPerDataObject

The number of human workers that will label an object.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 9.

Required: Yes

PreHumanTaskLambdaArn

The Amazon Resource Name (ARN) of a Lambda function that is run before a data object is sent to a human worker. Use this function to provide input to a custom labeling job.

For the built-in bounding box, image classification, semantic segmentation, and text classification task types, Amazon SageMaker Ground Truth provides the following Lambda functions:

US East (Northern Virginia) (us-east-1):

- `arn:aws:lambda:us-east-1:432418664414:function:PRE-BoundingBox`
- `arn:aws:lambda:us-east-1:432418664414:function:PRE-ImageMultiClass`
- `arn:aws:lambda:us-east-1:432418664414:function:PRE-SemanticSegmentation`
- `arn:aws:lambda:us-east-1:432418664414:function:PRE-TextMultiClass`
- `arn:aws:lambda:us-east-1:432418664414:function:PRE-NamedEntityRecognition`

US East (Ohio) (us-east-2):

- `arn:aws:lambda:us-east-2:266458841044:function:PRE-BoundingBox`
- `arn:aws:lambda:us-east-2:266458841044:function:PRE-ImageMultiClass`
- `arn:aws:lambda:us-east-2:266458841044:function:PRE-SemanticSegmentation`
- `arn:aws:lambda:us-east-2:266458841044:function:PRE-TextMultiClass`
- `arn:aws:lambda:us-east-2:266458841044:function:PRE-NamedEntityRecognition`

US West (Oregon) (us-west-2):

- `arn:aws:lambda:us-west-2:081040173940:function:PRE-BoundingBox`
- `arn:aws:lambda:us-west-2:081040173940:function:PRE-ImageMultiClass`
- `arn:aws:lambda:us-west-2:081040173940:function:PRE-SemanticSegmentation`
- `arn:aws:lambda:us-west-2:081040173940:function:PRE-TextMultiClass`
- `arn:aws:lambda:us-west-2:081040173940:function:PRE-NamedEntityRecognition`

Canada (Central) (ca-central-1):

- `arn:awslambda:ca-central-1:918755190332:function:PRE-BoundingBox`
- `arn:awslambda:ca-central-1:918755190332:function:PRE-ImageMultiClass`
- `arn:awslambda:ca-central-1:918755190332:function:PRE-SemanticSegmentation`
- `arn:awslambda:ca-central-1:918755190332:function:PRE-TextMultiClass`
- `arn:awslambda:ca-central-1:918755190332:function:PRE-NamedEntityRecognition`

EU (Ireland) (eu-west-1):

- `arn:aws:lambda:eu-west-1:568282634449:function:PRE-BoundingBox`
- `arn:aws:lambda:eu-west-1:568282634449:function:PRE-ImageMultiClass`
- `arn:aws:lambda:eu-west-1:568282634449:function:PRE-SemanticSegmentation`
- `arn:aws:lambda:eu-west-1:568282634449:function:PRE-TextMultiClass`
- `arn:aws:lambda:eu-west-1:568282634449:function:PRE-NamedEntityRecognition`

EU (London) (eu-west-2):

- `arn:awslambda:eu-west-2:487402164563:function:PRE-BoundingBox`
- `arn:awslambda:eu-west-2:487402164563:function:PRE-ImageMultiClass`
- `arn:awslambda:eu-west-2:487402164563:function:PRE-SemanticSegmentation`
- `arn:awslambda:eu-west-2:487402164563:function:PRE-TextMultiClass`
- `arn:awslambda:eu-west-2:487402164563:function:PRE-NamedEntityRecognition`

EU Frankfurt (eu-central-1):

- `arn:awslambda:eu-central-1:203001061592:function:PRE-BoundingBox`
- `arn:awslambda:eu-central-1:203001061592:function:PRE-ImageMultiClass`
- `arn:awslambda:eu-central-1:203001061592:function:PRE-SemanticSegmentation`
- `arn:awslambda:eu-central-1:203001061592:function:PRE-TextMultiClass`
- `arn:awslambda:eu-central-1:203001061592:function:PRE-NamedEntityRecognition`

Asia Pacific (Tokyo) (ap-northeast-1):

- `arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-BoundingBox`
- `arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-ImageMultiClass`
- `arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-SemanticSegmentation`
- `arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-TextMultiClass`
- `arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-NamedEntityRecognition`

Asia Pacific (Seoul) (ap-northeast-2):

- `arn:awslambda:ap-northeast-2:845288260483:function:PRE-BoundingBox`
- `arn:awslambda:ap-northeast-2:845288260483:function:PRE-ImageMultiClass`
- `arn:awslambda:ap-northeast-2:845288260483:function:PRE-SemanticSegmentation`

- `arn:awslambda:ap-northeast-2:845288260483:function:PRE-TextMultiClass`
- `arn:awslambda:ap-northeast-2:845288260483:function:PRE-NamedEntityRecognition`

Asia Pacific (Mumbai) (ap-south-1):

- `arn:awslambda:ap-south-1:565803892007:function:PRE-BoundingBox`
- `arn:awslambda:ap-south-1:565803892007:function:PRE-ImageMultiClass`
- `arn:awslambda:ap-south-1:565803892007:function:PRE-SemanticSegmentation`
- `arn:awslambda:ap-south-1:565803892007:function:PRE-TextMultiClass`
- `arn:awslambda:ap-south-1:565803892007:function:PRE-NamedEntityRecognition`

Asia Pacific (Singapore) (ap-southeast-1):

- `arn:awslambda:ap-southeast-1:377565633583:function:PRE-BoundingBox`
- `arn:awslambda:ap-southeast-1:377565633583:function:PRE-ImageMultiClass`
- `arn:awslambda:ap-southeast-1:377565633583:function:PRE-SemanticSegmentation`
- `arn:awslambda:ap-southeast-1:377565633583:function:PRE-TextMultiClass`
- `arn:awslambda:ap-southeast-1:377565633583:function:PRE-NamedEntityRecognition`

Asia Pacific (Sydney) (ap-southeast-2):

- `arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-BoundingBox`
- `arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-ImageMultiClass`
- `arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-SemanticSegmentation`
- `arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-TextMultiClass`
- `arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-NamedEntityRecognition`

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:lambda:[a-z]{2}-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: Yes

PublicWorkforceTaskPrice

The price that you pay for each task performed by an Amazon Mechanical Turk worker.

Type: PublicWorkforceTaskPrice (p. 987) object

Required: No

TaskAvailabilityLifetimeInSeconds

The length of time that a task remains available for labeling by human workers. **If you choose the Amazon Mechanical Turk workforce, the maximum is 12 hours (43200)**. For private and vendor workforces, the maximum is as listed.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 864000.

Required: No

TaskDescription

A description of the task for your human workers.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: Yes

TaskKeywords

Keywords used to describe the task so that workers on Amazon Mechanical Turk can discover the task.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Length Constraints: Minimum length of 1. Maximum length of 30.

Pattern: `^[A-Za-z0-9]+( [A-Za-z0-9]+)*$`

Required: No

TaskTimeLimitInSeconds

The amount of time that a worker has to complete a task.

Type: Integer

Valid Range: Minimum value of 30. Maximum value of 28800.

Required: Yes

TaskTitle

A title for the task for your human workers.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `^[\t\n\r -\uD7FF\uE000-\uFFFD]*$`

Required: Yes

UiConfig

Information about the user interface that workers use to complete the labeling task.

Type: UiConfig (p. 1036) object

Required: Yes

WorkteamArn

The Amazon Resource Name (ARN) of the work team assigned to complete the tasks.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterAlgorithmSpecification

Service: Amazon SageMaker Service

Specifies which training algorithm to use for training jobs that a hyperparameter tuning job launches and the metrics to monitor.

Contents

AlgorithmName

The name of the resource algorithm to use for the hyperparameter tuning job. If you specify a value for this parameter, do not specify a value for `TrainingImage`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\/)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?<!-)$`

Required: No

MetricDefinitions

An array of MetricDefinition (p. 956) objects that specify the metrics that the algorithm emits.

Type: Array of MetricDefinition (p. 956) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

Required: No

TrainingImage

The registry path of the Docker image that contains the training algorithm. For information about Docker registry paths for built-in algorithms, see Algorithms Provided by Amazon SageMaker: Common Parameters. Amazon SageMaker supports both `registry/repository[:tag]` and `registry/repository[@digest]` image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `.*`

Required: No

TrainingInputMode

The input mode that the algorithm supports: File or Pipe. In File input mode, Amazon SageMaker downloads the training data from Amazon S3 to the storage volume that is attached to the training instance and mounts the directory to the Docker volume for the training container. In Pipe input mode, Amazon SageMaker streams data directly from Amazon S3 to the container.

If you specify File mode, make sure that you provision the storage volume that is attached to the training instance with enough capacity to accommodate the training data downloaded from Amazon S3, the model artifacts, and intermediate information.

For more information about input modes, see Algorithms.

Type: String

Valid Values: `Pipe | File`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterSpecification

Service: Amazon SageMaker Service

Defines a hyperparameter to be used by an algorithm.

Contents

DefaultValue

The default value for this hyperparameter. If a default value is specified, a hyperparameter cannot be required.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: No

Description

A brief description of the hyperparameter.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

Required: No

IsRequired

Indicates whether this hyperparameter is required.

Type: Boolean

Required: No

IsTunable

Indicates whether this hyperparameter is tunable in a hyperparameter tuning job.

Type: Boolean

Required: No

Name

The name of this hyperparameter. The name must be unique.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

Required: Yes

Range

The allowed range for this hyperparameter.

Type: ParameterRange (p. 978) object

Required: No

Type

The type of this hyperparameter. The valid types are `Integer`, `Continuous`, `Categorical`, and `FreeText`.

Type: String

Valid Values: `Integer | Continuous | Categorical | FreeText`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterTrainingJobDefinition

Service: Amazon SageMaker Service

Defines the training jobs launched by a hyperparameter tuning job.

Contents

AlgorithmSpecification

The HyperParameterAlgorithmSpecification (p. 912) object that specifies the resource algorithm to use for the training jobs that the tuning job launches.

Type: HyperParameterAlgorithmSpecification (p. 912) object

Required: Yes

CheckpointConfig

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 880) object

Required: No

EnableInterContainerTrafficEncryption

To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

Required: No

EnableManagedSpotTraining

A Boolean indicating whether managed spot training is enabled (`True`) or not (`False`).

Type: Boolean

Required: No

EnableNetworkIsolation

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If network isolation is used for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

> **Note**
> The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

InputDataConfig

An array of Channel (p. 876) objects that specify the input for the training jobs that the tuning job launches.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

OutputDataConfig

Specifies the path to the Amazon S3 bucket where you store model artifacts from the training jobs that the tuning job launches.

Type: OutputDataConfig (p. 976) object

Required: Yes

ResourceConfig

The resources, including the compute instances and storage volumes, to use for the training jobs that the tuning job launches.

Storage volumes store model artifacts and incremental states. Training algorithms might also use storage volumes for scratch space. If you want Amazon SageMaker to use the storage volume to store the training data, choose `File` as the `TrainingInputMode` in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 991) object

Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role associated with the training jobs that the tuning job launches.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

StaticHyperParameters

Specifies the values of hyperparameters that do not change for the tuning job.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

StoppingCondition

Specifies a limit to how long a model hyperparameter training job can run. It also specifies how long you are willing to wait for a managed spot training job to complete. When the job reaches the a limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

Type: StoppingCondition (p. 1004) object

Required: Yes

VpcConfig

The VpcConfig (p. 1039) object that specifies the VPC that you want the training jobs that this hyperparameter tuning job launches to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterTrainingJobSummary

Service: Amazon SageMaker Service

Specifies summary information about a training job.

Contents

CreationTime

The date and time that the training job was created.

Type: Timestamp

Required: Yes

FailureReason

The reason that the training job failed.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

FinalHyperParameterTuningJobObjectiveMetric

The FinalHyperParameterTuningJobObjectiveMetric (p. 904) object that specifies the value of the objective metric of the tuning job that launched this training job.

Type: FinalHyperParameterTuningJobObjectiveMetric (p. 904) object

Required: No

ObjectiveStatus

The status of the objective metric for the training job:

- Succeeded: The final objective metric for the training job was evaluated by the hyperparameter tuning job and used in the hyperparameter tuning process.
- Pending: The training job is in progress and evaluation of its final objective metric is pending.
- Failed: The final objective metric for the training job was not evaluated, and was not used in the hyperparameter tuning process. This typically occurs when the training job failed or did not emit an objective metric.

Type: String

Valid Values: `Succeeded | Pending | Failed`

Required: No

TrainingEndTime

Specifies the time when the training job ends on training instances. You are billed for the time interval between the value of `TrainingStartTime` and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: Timestamp

Required: No

TrainingJobArn

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Required: Yes

TrainingJobName

The name of the training job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

TrainingJobStatus

The status of the training job.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: Yes

TrainingStartTime

The date and time that the training job started.

Type: Timestamp

Required: No

TunedHyperParameters

A list of the hyperparameters for which you specified ranges to search.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: Yes

TuningJobName

The HyperParameter tuning job that launched the training job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterTuningJobConfig

Service: Amazon SageMaker Service

Configures a hyperparameter tuning job.

Contents

HyperParameterTuningJobObjective

The HyperParameterTuningJobObjective (p. 924) object that specifies the objective metric for this tuning job.

Type: HyperParameterTuningJobObjective (p. 924) object

Required: No

ParameterRanges

The ParameterRanges (p. 979) object that specifies the ranges of hyperparameters that this tuning job searches.

Type: ParameterRanges (p. 979) object

Required: No

ResourceLimits

The ResourceLimits (p. 993) object that specifies the maximum number of training jobs and parallel training jobs for this tuning job.

Type: ResourceLimits (p. 993) object

Required: Yes

Strategy

Specifies how hyperparameter tuning chooses the combinations of hyperparameter values to use for the training job it launches. To use the Bayesian search stategy, set this to `Bayesian`. To randomly search, set it to `Random`. For information about search strategies, see How Hyperparameter Tuning Works.

Type: String

Valid Values: `Bayesian | Random`

Required: Yes

TrainingJobEarlyStoppingType

Specifies whether to use early stopping for training jobs launched by the hyperparameter tuning job. This can be one of the following values (the default value is `OFF`):

OFF

Training jobs launched by the hyperparameter tuning job do not use early stopping.

AUTO

Amazon SageMaker stops training jobs launched by the hyperparameter tuning job when they are unlikely to perform better than previously completed training jobs. For more information, see Stop Training Jobs Early.

Type: String

Valid Values: `Off | Auto`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterTuningJobObjective

Service: Amazon SageMaker Service

Defines the objective metric for a hyperparameter tuning job. Hyperparameter tuning uses the value of this metric to evaluate the training jobs it launches, and returns the training job that results in either the highest or lowest value for this metric, depending on the value you specify for the `Type` parameter.

Contents

MetricName

The name of the metric to use for the objective metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: Yes

Type

Whether to minimize or maximize the objective metric.

Type: String

Valid Values: `Maximize | Minimize`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterTuningJobSummary

Service: Amazon SageMaker Service

Provides summary information about a hyperparameter tuning job.

Contents

CreationTime

The date and time that the tuning job was created.

Type: Timestamp

Required: Yes

HyperParameterTuningEndTime

The date and time that the tuning job ended.

Type: Timestamp

Required: No

HyperParameterTuningJobArn

The Amazon Resource Name (ARN) of the tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

Required: Yes

HyperParameterTuningJobName

The name of the tuning job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

HyperParameterTuningJobStatus

The status of the tuning job.

Type: String

Valid Values: `Completed | InProgress | Failed | Stopped | Stopping`

Required: Yes

LastModifiedTime

The date and time that the tuning job was modified.

Type: Timestamp

Required: No

ObjectiveStatusCounters

The ObjectiveStatusCounters (p. 974) object that specifies the numbers of training jobs, categorized by objective metric status, that this tuning job launched.

Type: ObjectiveStatusCounters (p. 974) object

Required: Yes

ResourceLimits

The ResourceLimits (p. 993) object that specifies the maximum number of training jobs and parallel training jobs allowed for this tuning job.

Type: ResourceLimits (p. 993) object

Required: No

Strategy

Specifies the search strategy hyperparameter tuning uses to choose which hyperparameters to use for each iteration. Currently, the only valid value is Bayesian.

Type: String

Valid Values: `Bayesian | Random`

Required: Yes

TrainingJobStatusCounters

The TrainingJobStatusCounters (p. 1017) object that specifies the numbers of training jobs, categorized by status, that this tuning job launched.

Type: TrainingJobStatusCounters (p. 1017) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

HyperParameterTuningJobWarmStartConfig

Service: Amazon SageMaker Service

Specifies the configuration for a hyperparameter tuning job that uses one or more previous hyperparameter tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.

All training jobs launched by the new hyperparameter tuning job are evaluated by using the objective metric, and the training job that performs the best is compared to the best training jobs from the parent tuning jobs. From these, the training job that performs the best as measured by the objective metric is returned as the overall best training job.

> **Note**
> All training jobs launched by parent hyperparameter tuning jobs and the new hyperparameter tuning jobs count against the limit of training jobs for the tuning job.

Contents

ParentHyperParameterTuningJobs

An array of hyperparameter tuning jobs that are used as the starting point for the new hyperparameter tuning job. For more information about warm starting a hyperparameter tuning job, see Using a Previous Hyperparameter Tuning Job as a Starting Point.

Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

Type: Array of ParentHyperParameterTuningJob (p. 980) objects

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Required: Yes

WarmStartType

Specifies one of the following:

IDENTICAL_DATA_AND_ALGORITHM

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The objective metric for the new tuning job must be the same as for all parent jobs.

TRANSFER_LEARNING

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. The training image can also be a different version from the version used in the parent hyperparameter tuning job. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The objective metric for the new tuning job must be the same as for all parent jobs.

Type: String

Valid Values: `IdenticalDataAndAlgorithm` | `TransferLearning`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

InferenceSpecification

Service: Amazon SageMaker Service

Defines how to perform inference generation after a training job is run.

Contents

Containers

The Amazon ECR registry path of the Docker image that contains the inference code.

Type: Array of ModelPackageContainerDefinition (p. 958) objects

Array Members: Fixed number of 1 item.

Required: Yes

SupportedContentTypes

The supported MIME types for the input data.

Type: Array of strings

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

SupportedRealtimeInferenceInstanceTypes

A list of the instance types that are used to generate inferences in real-time.

Type: Array of strings

Valid Values: `ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.large | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.large | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge`

Required: Yes

SupportedResponseMIMETypes

The supported MIME types for the output data.

Type: Array of strings

Length Constraints: Maximum length of 1024.

Pattern: `^[-\w]+\/.+$`

Required: Yes

SupportedTransformInstanceTypes

A list of the instance types on which a transformation job can be run or on which an endpoint can be deployed.

Type: Array of strings

Array Members: Minimum number of 1 item.

Valid Values: `ml.m4.xlarge` | `ml.m4.2xlarge` | `ml.m4.4xlarge` | `ml.m4.10xlarge` | `ml.m4.16xlarge` | `ml.c4.xlarge` | `ml.c4.2xlarge` | `ml.c4.4xlarge` | `ml.c4.8xlarge` | `ml.p2.xlarge` | `ml.p2.8xlarge` | `ml.p2.16xlarge` | `ml.p3.2xlarge` | `ml.p3.8xlarge` | `ml.p3.16xlarge` | `ml.c5.xlarge` | `ml.c5.2xlarge` | `ml.c5.4xlarge` | `ml.c5.9xlarge` | `ml.c5.18xlarge` | `ml.m5.large` | `ml.m5.xlarge` | `ml.m5.2xlarge` | `ml.m5.4xlarge` | `ml.m5.12xlarge` | `ml.m5.24xlarge`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

InputConfig

Service: Amazon SageMaker Service

Contains information about the location of input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Contents

DataInputConfig

Specifies the name and shape of the expected data inputs for your trained model with a JSON dictionary form. The data inputs are InputConfig:Framework (p. 932) specific.

- `TensorFlow`: You must specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, `{"input":[1,1024,1024,3]}`
    - If using the CLI, `{\"input\":[1,1024,1024,3]}`
  - Examples for two inputs:
    - If using the console, `{"data1": [1,28,28,1], "data2":[1,28,28,1]}`
    - If using the CLI, `{\"data1\": [1,28,28,1], \"data2\":[1,28,28,1]}`
- `MXNET/ONNX`: You must specify the name and shape (NCHW format) of the expected data inputs in order using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, `{"data":[1,3,1024,1024]}`
    - If using the CLI, `{\"data\":[1,3,1024,1024]}`
  - Examples for two inputs:
    - If using the console, `{"var1": [1,1,28,28], "var2":[1,1,28,28]}`
    - If using the CLI, `{\"var1\": [1,1,28,28], \"var2\":[1,1,28,28]}`
- `PyTorch`: You can either specify the name and shape (NCHW format) of expected data inputs in order using a dictionary format for your trained model or you can specify the shape only using a list format. The dictionary formats required for the console and CLI are different. The list formats for the console and CLI are the same.
  - Examples for one input in dictionary format:
    - If using the console, `{"input0":[1,3,224,224]}`
    - If using the CLI, `{\"input0\":[1,3,224,224]}`
  - Example for one input in list format: `[[1,3,224,224]]`
  - Examples for two inputs in dictionary format:
    - If using the console, `{"input0":[1,3,224,224], "input1":[1,3,224,224]}`
    - If using the CLI, `{\"input0\":[1,3,224,224], \"input1\":[1,3,224,224]}`
  - Example for two inputs in list format: `[[1,3,224,224], [1,3,224,224]]`
- `XGBOOST`: input data name and shape are not needed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `[\S\s]+`

Required: Yes

Framework

Identifies the framework in which the model was trained. For example: TENSORFLOW.

Type: String

Valid Values: `TENSORFLOW | MXNET | ONNX | PYTORCH | XGBOOST`

Required: Yes

S3Uri

The S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

IntegerParameterRange

Service: Amazon SageMaker Service

For a hyperparameter of the integer type, specifies the range that a hyperparameter tuning job searches.

Contents

MaxValue

The maximum value of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

MinValue

The minimum value of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

Name

The name of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: Yes

ScalingType

The scale that hyperparameter tuning uses to search the hyperparameter range. For information about choosing a hyperparameter scale, see Hyperparameter Scaling. One of the following values:

Auto

Amazon SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.

Linear

Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale.

Logarithmic

Hyperparemeter tuning searches the values in the hyperparameter range by using a logarithmic scale.

Logarithmic scaling works only for ranges that have only values greater than 0.

Type: String

Valid Values: `Auto | Linear | Logarithmic | ReverseLogarithmic`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

IntegerParameterRangeSpecification

Service: Amazon SageMaker Service

Defines the possible values for an integer hyperparameter.

Contents

MaxValue

The maximum integer value allowed.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

MinValue

The minimum integer value allowed.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelCounters

Service: Amazon SageMaker Service

Provides a breakdown of the number of objects labeled.

Contents

FailedNonRetryableError

The total number of objects that could not be labeled due to an error.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

HumanLabeled

The total number of objects labeled by a human worker.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

MachineLabeled

The total number of objects labeled by automated data labeling.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

TotalLabeled

The total number of objects labeled.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Unlabeled

The total number of objects not yet labeled.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelCountersForWorkteam

Service: Amazon SageMaker Service

Provides counts for human-labeled tasks in the labeling job.

Contents

HumanLabeled

The total number of data objects labeled by a human worker.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

PendingHuman

The total number of data objects that need to be labeled by a human worker.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Total

The total number of tasks in the labeling job.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobAlgorithmsConfig

Service: Amazon SageMaker Service

Provides configuration information for auto-labeling of your data objects. A
`LabelingJobAlgorithmsConfig` object must be supplied in order to use auto-labeling.

Contents

InitialActiveLearningModelArn

At the end of an auto-label job Amazon SageMaker Ground Truth sends the Amazon Resource Nam
(ARN) of the final model used for auto-labeling. You can use this model as the starting point for
subsequent similar jobs by providing the ARN of the model here.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*`

Required: No

LabelingJobAlgorithmSpecificationArn

Specifies the Amazon Resource Name (ARN) of the algorithm used for auto-labeling. You must select
one of the following ARNs:

- *Image classification*

  `arn:aws:sagemaker:region:027400017018:labeling-job-algorithm-specification/image-classification`
- *Text classification*

  `arn:aws:sagemaker:region:027400017018:labeling-job-algorithm-specification/text-classification`
- *Object detection*

  `arn:aws:sagemaker:region:027400017018:labeling-job-algorithm-specification/object-detection`
- *Semantic Segmentation*

  `arn:aws:sagemaker:region:027400017018:labeling-job-algorithm-specification/semantic-segmentation`

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:.*`

Required: Yes

LabelingJobResourceConfig

Provides configuration information for a labeling job.

Type: LabelingJobResourceConfig (p. 948) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobDataAttributes

Service: Amazon SageMaker Service

Attributes of the data specified by the customer. Use these to describe the data to be labeled.

Contents

ContentClassifiers

Declares that your content is free of personally identifiable information or adult content. Amazon SageMaker may restrict the Amazon Mechanical Turk workers that can view your task based on this information.

Type: Array of strings

Array Members: Maximum number of 256 items.

Valid Values: `FreeOfPersonallyIdentifiableInformation | FreeOfAdultContent`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobDataSource

Service: Amazon SageMaker Service

Provides information about the location of input data.

Contents

S3DataSource

The Amazon S3 location of the input data objects.

Type: LabelingJobS3DataSource (p. 949) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobForWorkteamSummary

Service: Amazon SageMaker Service

Provides summary information for a work team.

Contents

CreationTime

The date and time that the labeling job was created.

Type: Timestamp

Required: Yes

JobReferenceCode

A unique identifier for a labeling job. You can use this to refer to a specific labeling job.

Type: String

Length Constraints: Minimum length of 1.

Pattern: `.+`

Required: Yes

LabelCounters

Provides information about the progress of a labeling job.

Type: LabelCountersForWorkteam (p. 938) object

Required: No

LabelingJobName

The name of the labeling job that the work team is assigned to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

NumberOfHumanWorkersPerDataObject

The configured number of workers per data object.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 9.

Required: No

WorkRequesterAccountId

Type: String

Pattern: `^\d+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobInputConfig

Service: Amazon SageMaker Service

Input configuration information for a labeling job.

Contents

DataAttributes

Attributes of the data specified by the customer.

Type: LabelingJobDataAttributes (p. 941) object

Required: No

DataSource

The location of the input data.

Type: LabelingJobDataSource (p. 942) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobOutput

Service: Amazon SageMaker Service

Specifies the location of the output produced by the labeling job.

Contents

FinalActiveLearningModelArn

The Amazon Resource Name (ARN) for the most recent Amazon SageMaker model trained as part of automated data labeling.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*`

Required: No

OutputDatasetS3Uri

The Amazon S3 bucket location of the manifest file for labeled data.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobOutputConfig

Service: Amazon SageMaker Service

Output configuration information for a labeling job.

Contents

KmsKeyId

The AWS Key Management Service ID of the key used to encrypt the output data, if any.

If you use a KMS key ID or an alias of your master key, the Amazon SageMaker execution role must include permissions to call `kms:Encrypt`. If you don't provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role's account. Amazon SageMaker uses server-side encryption with KMS-managed keys for `LabelingJobOutputConfig`. If you use a bucket policy with an `s3:PutObject` permission that only allows objects with server-side encryption, set the condition key of `s3:x-amz-server-side-encryption` to `"aws:kms"`. For more information, see KMS-Managed Encryption Keys in the *Amazon Simple Storage Service Developer Guide.*

The KMS key policy must grant permission to the IAM role that you specify in your `CreateLabelingJob` request. For more information, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

S3OutputPath

The Amazon S3 location to write output data.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobResourceConfig

Service: Amazon SageMaker Service

Provides configuration information for labeling jobs.

Contents

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the training job. The `VolumeKmsKeyId` can be any of the following formats:

- // KMS Key ID

  `"1234abcd-12ab-34cd-56ef-1234567890ab"`

- // Amazon Resource Name (ARN) of a KMS Key

  `"arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"`

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobS3DataSource

Service: Amazon SageMaker Service

The Amazon S3 location of the input data objects.

Contents

ManifestS3Uri

The Amazon S3 location of the manifest file that describes the input data objects.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobStoppingConditions

Service: Amazon SageMaker Service

A set of conditions for stopping a labeling job. If any of the conditions are met, the job is automatically stopped. You can use these conditions to control the cost of data labeling.

Contents

MaxHumanLabeledObjectCount

The maximum number of objects that can be labeled by human workers.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

MaxPercentageOfInputDatasetLabeled

The maximum number of input data objects that should be labeled.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

LabelingJobSummary

Service: Amazon SageMaker Service

Provides summary information about a labeling job.

Contents

AnnotationConsolidationLambdaArn

The Amazon Resource Name (ARN) of the Lambda function used to consolidate the annotations from individual workers into a label for a data object. For more information, see Annotation Consolidation.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:lambda:[a-z]{2}-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

CreationTime

The date and time that the job was created (timestamp).

Type: Timestamp

Required: Yes

FailureReason

If the `LabelingJobStatus` field is `Failed`, this field contains a description of the error.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

InputConfig

Input configuration for the labeling job.

Type: LabelingJobInputConfig (p. 945) object

Required: No

LabelCounters

Counts showing the progress of the labeling job.

Type: LabelCounters (p. 936) object

Required: Yes

LabelingJobArn

The Amazon Resource Name (ARN) assigned to the labeling job when it was created.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`

Required: Yes

LabelingJobName

The name of the labeling job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

LabelingJobOutput

The location of the output produced by the labeling job.

Type: LabelingJobOutput (p. 946) object

Required: No

LabelingJobStatus

The current status of the labeling job.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: Yes

LastModifiedTime

The date and time that the job was last modified (timestamp).

Type: Timestamp

Required: Yes

PreHumanTaskLambdaArn

The Amazon Resource Name (ARN) of a Lambda function. The function is run before each data object is sent to a worker.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:lambda:[a-z]{2}-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: Yes

WorkteamArn

The Amazon Resource Name (ARN) of the work team assigned to the job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

MemberDefinition

Service: Amazon SageMaker Service

Defines the Amazon Cognito user group that is part of a work team.

Contents

CognitoMemberDefinition

The Amazon Cognito user group that is part of the work team.

Type: CognitoMemberDefinition (p. 883) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

MetricData

Service: Amazon SageMaker Service

The name, value, and date and time of a metric that was emitted to Amazon CloudWatch.

Contents

MetricName

The name of the metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: No

Timestamp

The date and time that the algorithm emitted the metric.

Type: Timestamp

Required: No

Value

The value of the metric.

Type: Float

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

MetricDefinition

Service: Amazon SageMaker Service

Specifies a metric that the training algorithm writes to `stderr` or `stdout` . Amazon SageMakerhyperparameter tuning captures all defined metrics. You specify one metric that a hyperparameter tuning job uses as its objective metric to choose the best training job.

Contents

Name

The name of the metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: Yes

Regex

A regular expression that searches the output of a training job and gets the value of the metric. For more information about using regular expressions to define metrics, see Defining Objective Metrics.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 500.

Pattern: `.+`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelArtifacts

Service: Amazon SageMaker Service

Provides information about the location that is configured for storing model artifacts.

Contents

S3ModelArtifacts

The path of the S3 object that contains the model artifacts. For example, `s3://bucket-name/keynameprefix/model.tar.gz`.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelPackageContainerDefinition

Service: Amazon SageMaker Service

Describes the Docker container for the model package.

Contents

ContainerHostname

The DNS host name for the Docker container.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

Image

The Amazon EC2 Container Registry (Amazon ECR) path where inference code is stored.

If you are using your own custom algorithm instead of an algorithm provided by Amazon SageMaker, the inference code must meet Amazon SageMaker requirements. Amazon SageMaker supports both `registry/repository[:tag]` and `registry/repository[@digest]` image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `[\S]+`

Required: Yes

ImageDigest

An MD5 hash of the training algorithm that identifies the Docker image used for training.

Type: String

Length Constraints: Maximum length of 72.

Pattern: `^[Ss][Hh][Aa]256:[0-9a-fA-F]{64}$`

Required: No

ModelDataUrl

The Amazon S3 path where the model artifacts, which result from model training, are stored. This path must point to a single `gzip` compressed tar archive (`.tar.gz` suffix).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: No

ProductId

The AWS Marketplace product ID of the model package.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelPackageStatusDetails

Service: Amazon SageMaker Service

Specifies the validation and image scan statuses of the model package.

Contents

ImageScanStatuses

The status of the scan of the Docker image container for the model package.

Type: Array of ModelPackageStatusItem (p. 961) objects

Required: No

ValidationStatuses

The validation status of the model package.

Type: Array of ModelPackageStatusItem (p. 961) objects

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelPackageStatusItem

Service: Amazon SageMaker Service

Represents the overall status of a model package.

Contents

FailureReason

if the overall status is `Failed`, the reason for the failure.

Type: String

Required: No

Name

The name of the model package for which the overall status is being reported.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Status

The current status.

Type: String

Valid Values: `NotStarted | InProgress | Completed | Failed`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelPackageSummary

Service: Amazon SageMaker Service

Provides summary information about a model package.

Contents

CreationTime

A timestamp that shows when the model package was created.

Type: Timestamp

Required: Yes

ModelPackageArn

The Amazon Resource Name (ARN) of the model package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model-package/.*`

Required: Yes

ModelPackageDescription

A brief description of the model package.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

Required: No

ModelPackageName

The name of the model package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

ModelPackageStatus

The overall status of the model package.

Type: String

Valid Values: `Pending | InProgress | Completed | Failed | Deleting`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelPackageValidationProfile

Service: Amazon SageMaker Service

Contains data, such as the inputs and targeted instance types that are used in the process of validating the model package.

The data provided in the validation profile is made available to your buyers on AWS Marketplace.

Contents

ProfileName

The name of the profile for the model package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

TransformJobDefinition

The `TransformJobDefinition` object that describes the transform job used for the validation of the model package.

Type: TransformJobDefinition (p. 1026) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelPackageValidationSpecification

Service: Amazon SageMaker Service

Specifies batch transform jobs that Amazon SageMaker runs to validate your model package.

Contents

ValidationProfiles

An array of `ModelPackageValidationProfile` objects, each of which specifies a batch transform job that Amazon SageMaker runs to validate your model package.

Type: Array of ModelPackageValidationProfile (p. 964) objects

Array Members: Fixed number of 1 item.

Required: Yes

ValidationRole

The IAM roles to be used for the validation of the model package.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ModelSummary

Service: Amazon SageMaker Service

Provides summary information about a model.

Contents

CreationTime

A timestamp that indicates when the model was created.

Type: Timestamp

Required: Yes

ModelArn

The Amazon Resource Name (ARN) of the model.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*`

Required: Yes

ModelName

The name of the model that you want a summary for.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

NestedFilters

Service: Amazon SageMaker Service

Defines a list of `NestedFilters` objects. To satisfy the conditions specified in the `NestedFilters` call, a resource must satisfy the conditions of all of the filters.

For example, you could define a `NestedFilters` using the training job's `InputDataConfig` property to filter on `Channel` objects.

A `NestedFilters` object contains multiple filters. For example, to find all training jobs whose name contains `train` and that have `cat/data` in their `S3Uri` (specified in `InputDataConfig`), you need to create a `NestedFilters` object that specifies the `InputDataConfig` property with the following `Filter` objects:

- `'{Name:"InputDataConfig.ChannelName", "Operator":"EQUALS", "Value":"train"}',`
- `'{Name:"InputDataConfig.DataSource.S3DataSource.S3Uri",`
  `"Operator":"CONTAINS", "Value":"cat/data"}'`

Contents

Filters

A list of filters. Each filter acts on a property. Filters must contain at least one `Filters` value. For example, a `NestedFilters` call might include a filter on the `PropertyName` parameter of the `InputDataConfig` property: `InputDataConfig.DataSource.S3DataSource.S3Uri`.

Type: Array of Filter (p. 901) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: Yes

NestedPropertyName

The name of the property to use in the nested filters. The value must match a listed property name, such as `InputDataConfig`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

NotebookInstanceLifecycleConfigSummary

Service: Amazon SageMaker Service

Provides a summary of a notebook instance lifecycle configuration.

Contents

CreationTime

A timestamp that tells when the lifecycle configuration was created.

Type: Timestamp

Required: No

LastModifiedTime

A timestamp that tells when the lifecycle configuration was last modified.

Type: Timestamp

Required: No

NotebookInstanceLifecycleConfigArn

The Amazon Resource Name (ARN) of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

NotebookInstanceLifecycleConfigName

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

NotebookInstanceLifecycleHook

Service: Amazon SageMaker Service

Contains the notebook instance lifecycle configuration script.

Each lifecycle configuration script has a limit of 16384 characters.

The value of the `$PATH` environment variable that is available to both scripts is `/sbin:bin:/usr/sbin:/usr/bin`.

View CloudWatch Logs for notebook instance lifecycle configurations in log group `/aws/sagemaker/NotebookInstances` in log stream `[notebook-instance-name]/[LifecycleConfigHook]`.

Lifecycle configuration scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Contents

Content

A base64-encoded string that contains a shell script for a notebook instance lifecycle configuration.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 16384.

Pattern: `[\S\s]+`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

NotebookInstanceSummary

Service: Amazon SageMaker Service

Provides summary information for an Amazon SageMaker notebook instance.

Contents

AdditionalCodeRepositories

An array of up to three Git repositories associated with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

CreationTime

A timestamp that shows when the notebook instance was created.

Type: Timestamp

Required: No

DefaultCodeRepository

The Git repository associated with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

InstanceType

The type of ML compute instance that the notebook instance is running on.

Type: String

Valid Values: `ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge`

```
| ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge
```

Required: No

LastModifiedTime

A timestamp that shows when the notebook instance was last modified.

Type: Timestamp

Required: No

NotebookInstanceArn

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

NotebookInstanceLifecycleConfigName

The name of a notebook instance lifecycle configuration associated with this notebook instance.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

NotebookInstanceName

The name of the notebook instance that you want a summary for.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

NotebookInstanceStatus

The status of the notebook instance.

Type: String

Valid Values: `Pending | InService | Stopping | Stopped | Failed | Deleting | Updating`

Required: No

Url

The URL that you use to connect to the Jupyter instance running in your notebook instance.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

NotificationConfiguration

Service: Amazon SageMaker Service

Configures SNS notifications of available or expiring work items for work teams.

Contents

NotificationTopicArn

The ARN for the SNS topic to which notifications should be published.

Type: String

Pattern: `arn:aws[a-z\-]*:sns:[a-z0-9\-]*:[0-9]{12}:[a-zA-Z0-9_.-]*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ObjectiveStatusCounters

Service: Amazon SageMaker Service

Specifies the number of training jobs that this hyperparameter tuning job launched, categorized by the status of their objective metric. The objective metric status shows whether the final objective metric for the training job has been evaluated by the tuning job and used in the hyperparameter tuning process.

Contents

Failed

The number of training jobs whose final objective metric was not evaluated and used in the hyperparameter tuning process. This typically occurs when the training job failed or did not emit an objective metric.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Pending

The number of training jobs that are in progress and pending evaluation of their final objective metric.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Succeeded

The number of training jobs whose final objective metric was evaluated by the hyperparameter tuning job and used in the hyperparameter tuning process.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

OutputConfig

Service: Amazon SageMaker Service

Contains information about the output location for the compiled model and the device (target) that the model runs on.

Contents

S3OutputLocation

Identifies the S3 path where you want Amazon SageMaker to store the model artifacts. For example, s3://bucket-name/key-name-prefix.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

TargetDevice

Identifies the device that you want to run your model on after it has been compiled. For example: ml_c5.

Type: String

Valid Values: `lambda | ml_m4 | ml_m5 | ml_c4 | ml_c5 | ml_p2 | ml_p3 | jetson_tx1 | jetson_tx2 | jetson_nano | rasp3b | deeplens | rk3399 | rk3288 | aisage | sbe_c | qcs605 | qcs603`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

OutputDataConfig

Service: Amazon SageMaker Service

Provides information about how to store model training results (model artifacts).

Contents

KmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the model artifacts at rest using Amazon S3 server-side encryption. The `KmsKeyId` can be any of the following formats:

- // KMS Key ID

  `"1234abcd-12ab-34cd-56ef-1234567890ab"`

- // Amazon Resource Name (ARN) of a KMS Key

  `"arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"`

- // KMS Key Alias

  `"alias/ExampleAlias"`

- // Amazon Resource Name (ARN) of a KMS Key Alias

  `"arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias"`

If you use a KMS key ID or an alias of your master key, the Amazon SageMaker execution role must include permissions to call `kms:Encrypt`. If you don't provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role's account. Amazon SageMaker uses server-side encryption with KMS-managed keys for `OutputDataConfig`. If you use a bucket policy with an `s3:PutObject` permission that only allows objects with server-side encryption, set the condition key of `s3:x-amz-server-side-encryption` to `"aws:kms"`. For more information, see KMS-Managed Encryption Keys in the *Amazon Simple Storage Service Developer Guide.*

The KMS key policy must grant permission to the IAM role that you specify in your `CreateTrainingJob`, `CreateTransformJob`, or `CreateHyperParameterTuningJob` requests. For more information, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide.*

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

S3OutputPath

Identifies the S3 path where you want Amazon SageMaker to store the model artifacts. For example, `s3://bucket-name/key-name-prefix`.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ParameterRange

Service: Amazon SageMaker Service

Defines the possible values for categorical, continuous, and integer hyperparameters to be used by an algorithm.

Contents

CategoricalParameterRangeSpecification

A `CategoricalParameterRangeSpecification` object that defines the possible values for a categorical hyperparameter.

Type: CategoricalParameterRangeSpecification (p. 875) object

Required: No

ContinuousParameterRangeSpecification

A `ContinuousParameterRangeSpecification` object that defines the possible values for a continuous hyperparameter.

Type: ContinuousParameterRangeSpecification (p. 890) object

Required: No

IntegerParameterRangeSpecification

A `IntegerParameterRangeSpecification` object that defines the possible values for an integer hyperparameter.

Type: IntegerParameterRangeSpecification (p. 935) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ParameterRanges

Service: Amazon SageMaker Service

Specifies ranges of integer, continuous, and categorical hyperparameters that a hyperparameter tuning job searches. The hyperparameter tuning job launches training jobs with hyperparameter values within these ranges to find the combination of values that result in the training job with the best performance as measured by the objective metric of the hyperparameter tuning job.

> **Note**
> You can specify a maximum of 20 hyperparameters that a hyperparameter tuning job can search over. Every possible value of a categorical parameter range counts against this limit.

Contents

CategoricalParameterRanges

The array of CategoricalParameterRange (p. 874) objects that specify ranges of categorical hyperparameters that a hyperparameter tuning job searches.

Type: Array of CategoricalParameterRange (p. 874) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

ContinuousParameterRanges

The array of ContinuousParameterRange (p. 888) objects that specify ranges of continuous hyperparameters that a hyperparameter tuning job searches.

Type: Array of ContinuousParameterRange (p. 888) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

IntegerParameterRanges

The array of IntegerParameterRange (p. 933) objects that specify ranges of integer hyperparameters that a hyperparameter tuning job searches.

Type: Array of IntegerParameterRange (p. 933) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ParentHyperParameterTuningJob

Service: Amazon SageMaker Service

A previously completed or stopped hyperparameter tuning job to be used as a starting point for a new hyperparameter tuning job.

Contents

HyperParameterTuningJobName

The name of the hyperparameter tuning job to be used as a starting point for a new hyperparameter tuning job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ProductionVariant

Service: Amazon SageMaker Service

Identifies a model that you want to host and the resources to deploy for hosting it. If you are deploying multiple models, tell Amazon SageMaker how to distribute traffic among the models by specifying variant weights.

Contents

AcceleratorType

The size of the Elastic Inference (EI) instance to use for the production variant. EI instances provide on-demand GPU computing for inference. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: String

Valid Values: `ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge`

Required: No

InitialInstanceCount

Number of instances to launch initially.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

InitialVariantWeight

Determines initial traffic distribution among all of the models that you specify in the endpoint configuration. The traffic to a production variant is determined by the ratio of the `VariantWeight` to the sum of all `VariantWeight` values across all ProductionVariants. If unspecified, it defaults to 1.0.

Type: Float

Valid Range: Minimum value of 0.

Required: No

InstanceType

The ML compute instance type.

Type: String

Valid Values: `ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.large | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.large | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge`

Required: Yes

ModelName

The name of the model that you want to host. This is the name that you specified when creating the model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

VariantName

The name of the production variant.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ProductionVariantSummary

Service: Amazon SageMaker Service

Describes weight and capacities for a production variant associated with an endpoint. If you sent a request to the `UpdateEndpointWeightsAndCapacities` API and the endpoint status is `Updating`, you get different desired and current values.

Contents

CurrentInstanceCount

The number of instances associated with the variant.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

CurrentWeight

The weight associated with the variant.

Type: Float

Valid Range: Minimum value of 0.

Required: No

DeployedImages

An array of `DeployedImage` objects that specify the Amazon EC2 Container Registry paths of the inference images deployed on instances of this `ProductionVariant`.

Type: Array of objects

Required: No

DesiredInstanceCount

The number of instances requested in the `UpdateEndpointWeightsAndCapacities` request.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

DesiredWeight

The requested weight, as specified in the `UpdateEndpointWeightsAndCapacities` request.

Type: Float

Valid Range: Minimum value of 0.

Required: No

VariantName

The name of the variant.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

PropertyNameQuery

Service: Amazon SageMaker Service

A type of `SuggestionQuery`. A suggestion query for retrieving property names that match the specified hint.

Contents

PropertyNameHint

Text that is part of a property's name. The property names of hyperparameter, metric, and tag key names that begin with the specified text in the `PropertyNameHint`.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 100.

Pattern: `.*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

PropertyNameSuggestion

Service: Amazon SageMaker Service

A property name returned from a `GetSearchSuggestions` call that specifies a value in the `PropertyNameQuery` field.

Contents

PropertyName

A suggested property name based on what you entered in the search textbox in the Amazon SageMaker console.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.+`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

PublicWorkforceTaskPrice

Service: Amazon SageMaker Service

Defines the amount of money paid to an Amazon Mechanical Turk worker for each task performed.

Use one of the following prices for bounding box tasks. Prices are in US dollars and should be based on the complexity of the task; the longer it takes in your initial testing, the more you should offer.

- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
- 0.720
- 0.840
- 0.960
- 1.080
- 1.200

Use one of the following prices for image classification, text classification, and custom tasks. Prices are in US dollars.

- 0.012
- 0.024
- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
- 0.720
- 0.840
- 0.960
- 1.080
- 1.200

Use one of the following prices for semantic segmentation tasks. Prices are in US dollars.

- 0.840
- 0.960
- 1.080

- 1.200

Contents

AmountInUsd

Defines the amount of money paid to an Amazon Mechanical Turk worker in United States dollars.

Type: USD (p. 1038) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

RenderableTask

Service: Amazon SageMaker Service

Contains input values for a task.

Contents

Input

A JSON object that contains values for the variables defined in the template. It is made available to the template under the substitution variable `task.input`. For example, if you define a variable `task.input.text` in your template, you can supply the variable in the JSON object as `"text":` `"sample text"`.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 128000.

Pattern: `[\S\s]+`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

RenderingError

Service: Amazon SageMaker Service

A description of an error that occurred while rendering the template.

Contents

Code

A unique identifier for a specific class of errors.

Type: String

Required: Yes

Message

A human-readable message describing the error.

Type: String

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ResourceConfig

Service: Amazon SageMaker Service

Describes the resources, including ML compute instances and ML storage volumes, to use for model training.

Contents

InstanceCount

The number of ML compute instances to use. For distributed training, provide a value greater than 1.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

InstanceType

The ML compute instance type.

Type: String

Valid Values: `ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.p3dn.24xlarge`

Required: Yes

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the training job. The `VolumeKmsKeyId` can be any of the following formats:

- // KMS Key ID

  `"1234abcd-12ab-34cd-56ef-1234567890ab"`
- // Amazon Resource Name (ARN) of a KMS Key

  `"arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"`

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

VolumeSizeInGB

The size of the ML storage volume that you want to provision.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use the ML storage volume for scratch space. If you want to store the training data in the ML storage volume, choose `File` as the `TrainingInputMode` in the algorithm specification.

You must specify sufficient ML storage for your scenario.

> **Note**
> Amazon SageMaker supports only the General Purpose SSD (gp2) ML storage volume type.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ResourceLimits

Service: Amazon SageMaker Service

Specifies the maximum number of training jobs and parallel training jobs that a hyperparameter tuning job can launch.

Contents

MaxNumberOfTrainingJobs

The maximum number of training jobs that a hyperparameter tuning job can launch.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

MaxParallelTrainingJobs

The maximum number of concurrent training jobs that a hyperparameter tuning job can launch.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

S3DataSource

Service: Amazon SageMaker Service

Describes the S3 data source.

Contents

AttributeNames

A list of one or more attribute names to use that are found in a specified augmented manifest file.

Type: Array of strings

Array Members: Maximum number of 16 items.

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `.+`

Required: No

S3DataDistributionType

If you want Amazon SageMaker to replicate the entire dataset on each ML compute instance that is launched for model training, specify `FullyReplicated`.

If you want Amazon SageMaker to replicate a subset of data on each ML compute instance that is launched for model training, specify `ShardedByS3Key`. If there are n ML compute instances launched for a training job, each instance gets approximately $1/n$ of the number of S3 objects. In this case, model training on each machine uses only the subset of training data.

Don't choose more ML compute instances for training than available S3 objects. If you do, some nodes won't get any data and you will pay for nodes that aren't getting any training data. This applies in both File and Pipe modes. Keep this in mind when developing algorithms.

In distributed training, where you use multiple ML compute EC2 instances, you might choose `ShardedByS3Key`. If the algorithm requires copying training data to the ML storage volume (when `TrainingInputMode` is set to `File`), this copies $1/n$ of the number of objects.

Type: String

Valid Values: `FullyReplicated | ShardedByS3Key`

Required: No

S3DataType

If you choose `S3Prefix`, `S3Uri` identifies a key name prefix. Amazon SageMaker uses all objects that match the specified key name prefix for model training.

If you choose `ManifestFile`, `S3Uri` identifies an object that is a manifest file containing a list of object keys that you want Amazon SageMaker to use for model training.

If you choose `AugmentedManifestFile`, S3Uri identifies an object that is an augmented manifest file in JSON lines format. This file contains the data you want to use for model training. `AugmentedManifestFile` can only be used if the Channel's input mode is `Pipe`.

Type: String

Valid Values: `ManifestFile | S3Prefix | AugmentedManifestFile`

Required: Yes

S3Uri

Depending on the value specified for the `S3DataType`, identifies either a key name prefix or a manifest. For example:

- A key name prefix might look like this: `s3://bucketname/exampleprefix`.
- A manifest might look like this: `s3://bucketname/example.manifest`

The manifest is an S3 object which is a JSON file with the following format:

`[`

`{"prefix": "s3://customer_bucket/some/prefix/"},`

`"relative/path/to/custdata-1",`

`"relative/path/custdata-2",`

`...`

`]`

The preceding JSON matches the following `s3Uris`:

`s3://customer_bucket/some/prefix/relative/path/to/custdata-1`

`s3://customer_bucket/some/prefix/relative/path/custdata-2`

`...`

The complete set of `s3uris` in this manifest is the input data for the channel for this datasource. The object that each `s3uris` points to must be readable by the IAM role that Amazon SageMaker uses to perform tasks on your behalf.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

SearchExpression

Service: Amazon SageMaker Service

A multi-expression that searches for the specified resource or resources in a search. All resource objects that satisfy the expression's condition are included in the search results. You must specify at least one subexpression, filter, or nested filter. A `SearchExpression` can contain up to twenty elements.

A `SearchExpression` contains the following components:

- A list of `Filter` objects. Each filter defines a simple Boolean expression comprised of a resource property name, Boolean operator, and value.
- A list of `NestedFilter` objects. Each nested filter defines a list of Boolean expressions using a list of resource properties. A nested filter is satisfied if a single object in the list satisfies all Boolean expressions.
- A list of `SearchExpression` objects. A search expression object can be nested in a list of search expression objects.
- A Boolean operator: `And` or `Or`.

Contents

Filters

A list of filter objects.

Type: Array of Filter (p. 901) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

NestedFilters

A list of nested filter objects.

Type: Array of NestedFilters (p. 967) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

Operator

A Boolean operator used to evaluate the search expression. If you want every conditional statement in all lists to be satisfied for the entire search expression to be true, specify `And`. If only a single conditional statement needs to be true for the entire search expression to be true, specify `Or`. The default value is `And`.

Type: String

Valid Values: `And | Or`

Required: No

SubExpressions

A list of search expression objects.

Type: Array of SearchExpression (p. 996) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

SearchRecord

Service: Amazon SageMaker Service

An individual search result record that contains a single resource object.

Contents

TrainingJob

A `TrainingJob` object that is returned as part of a `Search` request.

Type: TrainingJob (p. 1009) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

SecondaryStatusTransition

Service: Amazon SageMaker Service

An array element of DescribeTrainingJob:SecondaryStatusTransitions (p. 749). It provides additional details about a status that the training job has transitioned through. A training job can be in one of several states, for example, starting, downloading, training, or uploading. Within each state, there are a number of intermediate states. For example, within the starting state, Amazon SageMaker could be starting the training job or launching the ML instances. These transitional states are referred to as the job's secondary status.

Contents

EndTime

A timestamp that shows when the training job transitioned out of this secondary status state into another secondary status state or when the training job has ended.

Type: Timestamp

Required: No

StartTime

A timestamp that shows when the training job transitioned to the current secondary status state.

Type: Timestamp

Required: Yes

Status

Contains a secondary status information from a training job.

Status might be one of the following secondary statuses:

InProgress

- `Starting` - Starting the training job.
- `Downloading` - An optional stage for algorithms that support `File` training input mode. It indicates that data is being downloaded to the ML storage volumes.
- `Training` - Training is in progress.
- `Uploading` - Training is complete and the model artifacts are being uploaded to the S3 location.

Completed

- `Completed` - The training job has completed.

Failed

- `Failed` - The training job has failed. The reason for the failure is returned in the `FailureReason` field of `DescribeTrainingJobResponse`.

Stopped

- `MaxRuntimeExceeded` - The job stopped because it exceeded the maximum allowed runtime.
- `Stopped` - The training job has stopped.

Stopping

- `Stopping` - Stopping the training job.

We no longer support the following secondary statuses:

- `LaunchingMLInstances`
- `PreparingTrainingStack`

- `DownloadingTrainingImage`

Type: String

Valid Values: `Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed | Interrupted | MaxWaitTimeExceeded`

Required: Yes

StatusMessage

A detailed description of the progress within a secondary status.

Amazon SageMaker provides secondary statuses and status messages that apply to each of them:
Starting

- Starting the training job.
- Launching requested ML instances.
- Insufficient capacity error from EC2 while launching instances, retrying!
- Launched instance was unhealthy, replacing it!
- Preparing the instances for training.

Training

- Downloading the training image.
- Training image download completed. Training in progress.

Important
Status messages are subject to change. Therefore, we recommend not including them in code that programmatically initiates actions. For examples, don't use status messages in if statements.

To have an overview of your training job's progress, view `TrainingJobStatus` and `SecondaryStatus` in DescribeTrainingJob (p. 744), and `StatusMessage` together. For example, at the start of a training job, you might see the following:

- `TrainingJobStatus` - InProgress
- `SecondaryStatus` - Training
- `StatusMessage` - Downloading the training image

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

ShuffleConfig

Service: Amazon SageMaker Service

A configuration for a shuffle option for input data in a channel. If you use `S3Prefix` for `S3DataType`, the results of the S3 key prefix matches are shuffled. If you use `ManifestFile`, the order of the S3 object references in the `ManifestFile` is shuffled. If you use `AugmentedManifestFile`, the order of the JSON lines in the `AugmentedManifestFile` is shuffled. The shuffling order is determined using the `Seed` value.

For Pipe input mode, shuffling is done at the start of every epoch. With large datasets, this ensures that the order of the training data is different for each epoch, and it helps reduce bias and possible overfitting. In a multi-node training job when `ShuffleConfig` is combined with `S3DataDistributionType` of `ShardedByS3Key`, the data is shuffled across nodes so that the content sent to a particular node on the first epoch might be sent to a different node on the second epoch.

Contents

Seed

Determines the shuffling order in `ShuffleConfig` value.

Type: Long

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

SourceAlgorithm

Service: Amazon SageMaker Service

Specifies an algorithm that was used to create the model package. The algorithm must be either an algorithm resource in your Amazon SageMaker account or an algorithm in AWS Marketplace that you are subscribed to.

Contents

AlgorithmName

The name of an algorithm that was used to create the model package. The algorithm must be either an algorithm resource in your Amazon SageMaker account or an algorithm in AWS Marketplace that you are subscribed to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\/)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?<!-)$`

Required: Yes

ModelDataUrl

The Amazon S3 path where the model artifacts, which result from model training, are stored. This path must point to a single `gzip` compressed tar archive (`.tar.gz` suffix).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

SourceAlgorithmSpecification

Service: Amazon SageMaker Service

A list of algorithms that were used to create a model package.

Contents

SourceAlgorithms

A list of the algorithms that were used to create a model package.

Type: Array of SourceAlgorithm (p. 1002) objects

Array Members: Fixed number of 1 item.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

StoppingCondition

Service: Amazon SageMaker Service

Specifies a limit to how long a model training or compilation job can run. It also specifies how long you are willing to wait for a managed spot training job to complete. When the job reaches the time limit, Amazon SageMaker ends the training or compilation job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

The training algorithms provided by Amazon SageMaker automatically save the intermediate results of a model training job when possible. This attempt to save artifacts is only a best effort case as model might not be in a state from which it can be saved. For example, if training has just started, the model might not be ready to save. When saved, this intermediate data is a valid model artifact. You can use it to create a model with `CreateModel`.

> **Note**
> The Neural Topic Model (NTM) currently does not support saving intermediate model artifacts. When training NTMs, make sure that the maximum runtime is sufficient for the training job to complete.

Contents

MaxRuntimeInSeconds

The maximum length of time, in seconds, that the training or compilation job can run. If job does not complete during this time, Amazon SageMaker ends the job. If value is not specified, default value is 1 day. The maximum value is 28 days.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

MaxWaitTimeInSeconds

The maximum length of time, in seconds, how long you are willing to wait for a managed spot training job to complete. It is the amount of time spent waiting for Spot capacity plus the amount of time the training job runs. It must be equal to or greater than `MaxRuntimeInSeconds`.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

SubscribedWorkteam

Service: Amazon SageMaker Service

Describes a work team of a vendor that does the a labelling job.

Contents

ListingId

   Type: String

   Required: No

MarketplaceDescription

   The description of the vendor from the Amazon Marketplace.

   Type: String

   Length Constraints: Minimum length of 1. Maximum length of 200.

   Pattern: `.+`

   Required: No

MarketplaceTitle

   The title of the service provided by the vendor in the Amazon Marketplace.

   Type: String

   Length Constraints: Minimum length of 1. Maximum length of 200.

   Pattern: `.+`

   Required: No

SellerName

   The name of the vendor in the Amazon Marketplace.

   Type: String

   Required: No

WorkteamArn

   The Amazon Resource Name (ARN) of the vendor that you have subscribed.

   Type: String

   Length Constraints: Maximum length of 256.

   Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

   Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

SuggestionQuery

Service: Amazon SageMaker Service

Limits the property names that are included in the response.

Contents

PropertyNameQuery

A type of `SuggestionQuery`. Defines a property name hint. Only property names that match the specified hint are included in the response.

Type: PropertyNameQuery (p. 985) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

Tag

Service: Amazon SageMaker Service

Describes a tag.

Contents

Key

The tag key.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-@]*)$`

Required: Yes

Value

The tag value.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-@]*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TrainingJob

Service: Amazon SageMaker Service

Contains information about a training job.

Contents

AlgorithmSpecification

Information about the algorithm used for training, and algorithm metadata.

Type: AlgorithmSpecification (p. 863) object

Required: No

CreationTime

A timestamp that indicates when the training job was created.

Type: Timestamp

Required: No

EnableInterContainerTrafficEncryption

To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

Required: No

EnableNetworkIsolation

If the `TrainingJob` was created with network isolation, the value is set to `true`. If network isolation is enabled, nodes can't communicate beyond the VPC they run in.

Type: Boolean

Required: No

FailureReason

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

FinalMetricDataList

A list of final metric values that are set when the training job completes. Used only if the training job was configured to use metrics.

Type: Array of MetricData (p. 955) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

Required: No

HyperParameters

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

InputDataConfig

An array of `Channel` objects that describes each data input channel.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

LabelingJobArn

The Amazon Resource Name (ARN) of the labeling job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`

Required: No

LastModifiedTime

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

Required: No

ModelArtifacts

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: ModelArtifacts (p. 957) object

Required: No

OutputDataConfig

The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.

Type: OutputDataConfig (p. 976) object

Required: No

ResourceConfig

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: ResourceConfig (p. 991) object

Required: No

RoleArn

The AWS Identity and Access Management (IAM) role configured for the training job.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: No

SecondaryStatus

Provides detailed information about the state of the training job. For detailed information about the secondary status of the training job, see `StatusMessage` under SecondaryStatusTransition (p. 999).

Amazon SageMaker provides primary statuses and secondary statuses that apply to each of them:

InProgress

- `Starting` - Starting the training job.
- `Downloading` - An optional stage for algorithms that support `File` training input mode. It indicates that data is being downloaded to the ML storage volumes.
- `Training` - Training is in progress.
- `Uploading` - Training is complete and the model artifacts are being uploaded to the S3 location.

Completed

- `Completed` - The training job has completed.

Failed

- `Failed` - The training job has failed. The reason for the failure is returned in the `FailureReason` field of `DescribeTrainingJobResponse`.

Stopped

- `MaxRuntimeExceeded` - The job stopped because it exceeded the maximum allowed runtime.
- `Stopped` - The training job has stopped.

Stopping

- `Stopping` - Stopping the training job.

  Important
  Valid values for `SecondaryStatus` are subject to change.

We no longer support the following secondary statuses:

- `LaunchingMLInstances`
- `PreparingTrainingStack`
- `DownloadingTrainingImage`

Type: String

Valid Values: `Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed | Interrupted | MaxWaitTimeExceeded`

Required: No

SecondaryStatusTransitions

A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of SecondaryStatusTransition (p. 999) objects

Required: No

StoppingCondition

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1004) object

Required: No

Tags

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1008) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingEndTime

Indicates the time when the training job ends on training instances. You are billed for the time interval between the value of `TrainingStartTime` and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: Timestamp

Required: No

TrainingJobArn

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Required: No

TrainingJobName

The name of the training job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

TrainingJobStatus

The status of the training job.

Training job statuses are:

- `InProgress` - The training is in progress.
- `Completed` - The training job has completed.
- `Failed` - The training job has failed. To see the reason for the failure, see the `FailureReason` field in the response to a `DescribeTrainingJobResponse` call.
- `Stopping` - The training job is stopping.
- `Stopped` - The training job has stopped.

For more detailed information, see `SecondaryStatus`.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: No

TrainingStartTime

Indicates the time when the training job starts on training instances. You are billed for the time interval between this time and the value of `TrainingEndTime`. The start time in CloudWatch Logs might be later than this time. The difference is due to the time it takes to download the training data and to the size of the training container.

Type: Timestamp

Required: No

TuningJobArn

The Amazon Resource Name (ARN) of the associated hyperparameter tuning job if the training job was launched by a hyperparameter tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

Required: No

VpcConfig

A VpcConfig (p. 1039) object that specifies the VPC that this training job has access to. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1039) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TrainingJobDefinition

Service: Amazon SageMaker Service

Defines the input needed to run a training job using the algorithm.

Contents

HyperParameters

The hyperparameters used for the training job.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.*`

Required: No

InputDataConfig

An array of `Channel` objects, each of which specifies an input source.

Type: Array of Channel (p. 876) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: Yes

OutputDataConfig

the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 976) object

Required: Yes

ResourceConfig

The resources, including the ML compute instances and ML storage volumes, to use for model training.

Type: ResourceConfig (p. 991) object

Required: Yes

StoppingCondition

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts.

Type: StoppingCondition (p. 1004) object

Required: Yes

TrainingInputMode

The input mode used by the algorithm for the training job. For the input modes that Amazon SageMaker algorithms support, see Algorithms.

If an algorithm supports the `File` input mode, Amazon SageMaker downloads the training data from S3 to the provisioned ML storage Volume, and mounts the directory to docker volume for training container. If an algorithm supports the `Pipe` input mode, Amazon SageMaker streams data directly from S3 to the container.

Type: String

Valid Values: `Pipe | File`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TrainingJobStatusCounters

Service: Amazon SageMaker Service

The numbers of training jobs launched by a hyperparameter tuning job, categorized by status.

Contents

Completed

The number of completed training jobs launched by the hyperparameter tuning job.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

InProgress

The number of in-progress training jobs launched by a hyperparameter tuning job.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

NonRetryableError

The number of training jobs that failed and can't be retried. A failed training job can't be retried if it failed because a client error occurred.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

RetryableError

The number of training jobs that failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Stopped

The number of training jobs launched by a hyperparameter tuning job that were manually stopped.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TrainingJobSummary

Service: Amazon SageMaker Service

Provides summary information about a training job.

Contents

CreationTime

A timestamp that shows when the training job was created.

Type: Timestamp

Required: Yes

LastModifiedTime

Timestamp when the training job was last modified.

Type: Timestamp

Required: No

TrainingEndTime

A timestamp that shows when the training job ended. This field is set only if the training job has one of the terminal statuses (`Completed`, `Failed`, or `Stopped`).

Type: Timestamp

Required: No

TrainingJobArn

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Required: Yes

TrainingJobName

The name of the training job that you want a summary for.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

TrainingJobStatus

The status of the training job.

Type: String

Valid Values: `InProgress | Completed | Failed | Stopping | Stopped`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TrainingSpecification

Service: Amazon SageMaker Service

Defines how the algorithm is used for a training job.

Contents

MetricDefinitions

A list of `MetricDefinition` objects, which are used for parsing metrics generated by the algorithm.

Type: Array of MetricDefinition (p. 956) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

Required: No

SupportedHyperParameters

A list of the `HyperParameterSpecification` objects, that define the supported hyperparameters. This is required if the algorithm supports automatic model tuning.>

Type: Array of HyperParameterSpecification (p. 914) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

Required: No

SupportedTrainingInstanceTypes

A list of the instance types that this algorithm can use for training.

Type: Array of strings

Valid Values: `ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.p3dn.24xlarge`

Required: Yes

SupportedTuningJobObjectiveMetrics

A list of the metrics that the algorithm emits that can be used as the objective metric in a hyperparameter tuning job.

Type: Array of HyperParameterTuningJobObjective (p. 924) objects

Required: No

SupportsDistributedTraining

Indicates whether the algorithm supports distributed training. If set to false, buyers can't request more than one instance during training.

Type: Boolean

Required: No

TrainingChannels

A list of `ChannelSpecification` objects, which specify the input sources to be used by the algorithm.

Type: Array of ChannelSpecification (p. 878) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: Yes

TrainingImage

The Amazon ECR registry path of the Docker image that contains the training algorithm.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `[\S]+`

Required: Yes

TrainingImageDigest

An MD5 hash of the training algorithm that identifies the Docker image used for training.

Type: String

Length Constraints: Maximum length of 72.

Pattern: `^[Ss][Hh][Aa]256:[0-9a-fA-F]{64}$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TransformDataSource

Service: Amazon SageMaker Service

Describes the location of the channel data.

Contents

S3DataSource

The S3 location of the data source that is associated with a channel.

Type: TransformS3DataSource (p. 1034) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TransformInput

Service: Amazon SageMaker Service

Describes the input source of a transform job and the way the transform job consumes it.

Contents

CompressionType

If your transform data is compressed, specify the compression type. Amazon SageMaker automatically decompresses the data for the transform job accordingly. The default value is `None`.

Type: String

Valid Values: `None | Gzip`

Required: No

ContentType

The multipurpose internet mail extension (MIME) type of the data. Amazon SageMaker uses the MIME type with each http call to transfer data to the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: No

DataSource

Describes the location of the channel data, which is, the S3 location of the input data that the model can consume.

Type: TransformDataSource (p. 1023) object

Required: Yes

SplitType

The method to use to split the transform job's data files into smaller batches. Splitting is necessary when the total size of each object is too large to fit in a single request. You can also use data splitting to improve performance by processing multiple concurrent mini-batches. The default value for `SplitType` is `None`, which indicates that input data files are not split, and request payloads contain the entire contents of an input object. Set the value of this parameter to `Line` to split records on a newline character boundary. `SplitType` also supports a number of record-oriented binary data formats.

When splitting is enabled, the size of a mini-batch depends on the values of the `BatchStrategy` and `MaxPayloadInMB` parameters. When the value of `BatchStrategy` is `MultiRecord`, Amazon SageMaker sends the maximum number of records in each request, up to the `MaxPayloadInMB` limit. If the value of `BatchStrategy` is `SingleRecord`, Amazon SageMaker sends individual records in each request.

> **Note**
> Some data formats represent a record as a binary payload wrapped with extra padding bytes. When splitting is applied to a binary data format, padding is removed if the value of `BatchStrategy` is set to `SingleRecord`. Padding is not removed if the value of `BatchStrategy` is set to `MultiRecord`.

For more information about the RecordIO, see Data Format in the MXNet documentation.
For more information about the TFRecord, see Consuming TFRecord data in the TensorFlow
documentation.

Type: String

Valid Values: `None | Line | RecordIO | TFRecord`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TransformJobDefinition

Service: Amazon SageMaker Service

Defines the input needed to run a transform job using the inference specification specified in the algorithm.

Contents

BatchStrategy

A string that determines the number of records included in a single mini-batch.

`SingleRecord` means only one record is used per mini-batch. `MultiRecord` means a mini-batch is set to contain as many records that can fit within the `MaxPayloadInMB` limit.

Type: String

Valid Values: `MultiRecord | SingleRecord`

Required: No

Environment

The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: `[a-zA-Z_][a-zA-Z0-9_]*`

Value Length Constraints: Maximum length of 10240.

Value Pattern: `[\S\s]*`

Required: No

MaxConcurrentTransforms

The maximum number of parallel requests that can be sent to each instance in a transform job. The default value is 1.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

MaxPayloadInMB

The maximum payload size allowed, in MB. A payload is the data portion of a record (without metadata).

Type: Integer

Valid Range: Minimum value of 0.

Required: No

TransformInput

A description of the input source and the way the transform job consumes it.

Type: TransformInput (p. 1024) object

Required: Yes

TransformOutput

Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.

Type: TransformOutput (p. 1030) object

Required: Yes

TransformResources

Identifies the ML compute instances for the transform job.

Type: TransformResources (p. 1032) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TransformJobSummary

Service: Amazon SageMaker Service

Provides a summary of a transform job. Multiple `TransformJobSummary` objects are returned as a list after in response to a call.

Contents

CreationTime

A timestamp that shows when the transform Job was created.

Type: Timestamp

Required: Yes

FailureReason

If the transform job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

LastModifiedTime

Indicates when the transform job was last modified.

Type: Timestamp

Required: No

TransformEndTime

Indicates when the transform job ends on compute instances. For successful jobs and stopped jobs, this is the exact time recorded after the results are uploaded. For failed jobs, this is when Amazon SageMaker detected that the job failed.

Type: Timestamp

Required: No

TransformJobArn

The Amazon Resource Name (ARN) of the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*`

Required: Yes

TransformJobName

The name of the transform job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

TransformJobStatus

The status of the transform job.

Type: String

Valid Values: `InProgress` | `Completed` | `Failed` | `Stopping` | `Stopped`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TransformOutput

Service: Amazon SageMaker Service

Describes the results of a transform job.

Contents

Accept

The MIME type used to specify the output data. Amazon SageMaker uses the MIME type with each http call to transfer data from the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `.*`

Required: No

AssembleWith

Defines how to assemble the results of the transform job as a single S3 object. Choose a format that is most convenient to you. To concatenate the results in binary format, specify `None`. To add a newline character at the end of every transformed record, specify `Line`.

Type: String

Valid Values: `None | Line`

Required: No

KmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the model artifacts at rest using Amazon S3 server-side encryption. The `KmsKeyId` can be any of the following formats:

- // KMS Key ID

  `"1234abcd-12ab-34cd-56ef-1234567890ab"`

- // Amazon Resource Name (ARN) of a KMS Key

  `"arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"`

- // KMS Key Alias

  `"alias/ExampleAlias"`

- // Amazon Resource Name (ARN) of a KMS Key Alias

  `"arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias"`

If you don't provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role's account. For more information, see KMS-Managed Encryption Keys in the *Amazon Simple Storage Service Developer Guide.*

The KMS key policy must grant permission to the IAM role that you specify in your `CreateTramsformJob` request. For more information, see Using Key Policies in AWS KMS in the *AWS Key Management Service Developer Guide.*

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

S3OutputPath

The Amazon S3 path where you want Amazon SageMaker to store the results of the transform job. For example, `s3://bucket-name/key-name-prefix`.

For every S3 object used as input for the transform job, batch transform stores the transformed data with an `.out` suffix in a corresponding subfolder in the location in the output prefix. For example, for the input data stored at `s3://bucket-name/input-name-prefix/dataset01/data.csv`, batch transform stores the transformed data at `s3://bucket-name/output-name-prefix/input-name-prefix/data.csv.out`. Batch transform doesn't upload partially processed objects. For an input S3 object that contains multiple records, it creates an `.out` file only if the transform job succeeds on the entire file. When the input contains multiple S3 objects, the batch transform job processes the listed S3 objects and uploads only the output for successfully processed objects. If any object fails in the transform job batch transform marks the job as failed to prompt investigation.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TransformResources

Service: Amazon SageMaker Service

Describes the resources, including ML instance types and ML instance count, to use for transform job.

Contents

InstanceCount

The number of ML compute instances to use in the transform job. For distributed transform jobs, specify a value greater than 1. The default value is `1`.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

InstanceType

The ML compute instance type for the transform job. If you are using built-in algorithms to transform moderately sized datasets, we recommend using ml.m4.xlarge or `ml.m5.large` instance types.

Type: String

Valid Values: `ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge`

Required: Yes

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the batch transform job. The `VolumeKmsKeyId` can be any of the following formats:

- // KMS Key ID

  `"1234abcd-12ab-34cd-56ef-1234567890ab"`

- // Amazon Resource Name (ARN) of a KMS Key

  `"arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"`

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

TransformS3DataSource

Service: Amazon SageMaker Service

Describes the S3 data source.

Contents

S3DataType

If you choose `S3Prefix`, `S3Uri` identifies a key name prefix. Amazon SageMaker uses all objects with the specified key name prefix for batch transform.

If you choose `ManifestFile`, `S3Uri` identifies an object that is a manifest file containing a list of object keys that you want Amazon SageMaker to use for batch transform.

The following values are compatible: `ManifestFile`, `S3Prefix`

The following value is not compatible: `AugmentedManifestFile`

Type: String

Valid Values: `ManifestFile | S3Prefix | AugmentedManifestFile`

Required: Yes

S3Uri

Depending on the value specified for the `S3DataType`, identifies either a key name prefix or a manifest. For example:

- A key name prefix might look like this: `s3://bucketname/exampleprefix`.
- A manifest might look like this: `s3://bucketname/example.manifest`

  The manifest is an S3 object which is a JSON file with the following format:

  [

  {"prefix": "s3://customer_bucket/some/prefix/"},

  "relative/path/to/custdata-1",

  "relative/path/custdata-2",

  ...

  ]

  The preceding JSON matches the following `S3Uris`:

  s3://customer_bucket/some/prefix/relative/path/to/custdata-1

  s3://customer_bucket/some/prefix/relative/path/custdata-1

  ...

  The complete set of `S3Uris` in this manifest constitutes the input data for the channel for this datasource. The object that each `S3Uris` points to must be readable by the IAM role that Amazon SageMaker uses to perform tasks on your behalf.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

UiConfig

Service: Amazon SageMaker Service

Provided configuration information for the worker UI for a labeling job.

Contents

UiTemplateS3Uri

The Amazon S3 bucket location of the UI template. For more information about the contents of a UI template, see Creating Your Custom Labeling Task Template.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?(.*)$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

UiTemplate

Service: Amazon SageMaker Service

The Liquid template for the worker user interface.

Contents

Content

The content of the Liquid template for the worker user interface.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128000.

Pattern: `[\S\s]+`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

USD

Service: Amazon SageMaker Service

Represents an amount of money in United States dollars/

Contents

Cents

The fractional portion, in cents, of the amount.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 99.

Required: No

Dollars

The whole number of dollars in the amount.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 1.

Required: No

TenthFractionsOfACent

Fractions of a cent, in tenths.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 9.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

VpcConfig

Service: Amazon SageMaker Service

Specifies a VPC that your training jobs and hosted models have access to. Control access to and from your training and model containers by configuring the VPC. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud and Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Contents

SecurityGroupIds

The VPC security group IDs, in the form sg-xxxxxxxx. Specify the security groups for the VPC that is specified in the `Subnets` field.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Required: Yes

Subnets

The ID of the subnets in the VPC to which you want to connect your training job or model.

> **Note**
> Amazon EC2 P3 accelerated computing instances are not available in the c/d/e availability zones of region us-east-1. If you want to create endpoints with P3 instances in VPC mode in region us-east-1, create subnets in a/b/f availability zones instead.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 16 items.

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

Workteam

Service: Amazon SageMaker Service

Provides details about a labeling work team.

Contents

CreateDate

The date and time that the work team was created (timestamp).

Type: Timestamp

Required: No

Description

A description of the work team.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `.+`

Required: Yes

LastUpdatedDate

The date and time that the work team was last updated (timestamp).

Type: Timestamp

Required: No

MemberDefinitions

The Amazon Cognito user groups that make up the work team.

Type: Array of MemberDefinition (p. 954) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: Yes

NotificationConfiguration

Configures SNS notifications of available or expiring work items for work teams.

Type: NotificationConfiguration (p. 973) object

Required: No

ProductListingIds

The Amazon Marketplace identifier for a vendor's work team.

Type: Array of strings

Required: No

SubDomain

The URI of the labeling job's user interface. Workers open this URI to start labeling your data objects.

Type: String

Required: No

WorkteamArn

The Amazon Resource Name (ARN) that identifies the work team.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

Required: Yes

WorkteamName

The name of the work team.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

Amazon SageMaker Runtime

Currently Amazon SageMaker Runtime does not support any data types.

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400

IncompleteSignature

The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

InvalidParameterCombination

Parameters that must not be used together were used together.

HTTP Status Code: 400

InvalidParameterValue

An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

InvalidQueryParameter

The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

MalformedQueryString

The query string contains a syntax error.

HTTP Status Code: 404

MissingAction

The request is missing an action or a required parameter.

HTTP Status Code: 400

MissingAuthenticationToken

The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

MissingParameter

A required parameter for the specified action is not supplied.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the *Amazon Web Services General Reference*.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: `AWS4-HMAC-SHA256`

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: *access_key*/*YYYYMMDD*/*region*/*service*/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: `20120325T120000Z`.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the *Amazon Web Services General Reference*.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the *IAM User Guide*.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

Document History for Amazon SageMaker

update-history-change	update-history-description	update-history-date
New features re:Invent 2018	Amazon SageMaker Ground Truth, Using Elastic Inference in Amazon SageMaker, Amazon SageMaker Resources in AWS Marketplace, Amazon SageMaker Inference Pipelines, Amazon SageMaker Neo, Manage Machine Learning Experiments with Search , Use Reinforcement Learning in Amazon SageMaker, Associating Git Repositories with Amazon SageMaker Notebook Instances, Semantic Segmentation, Using Augmented Manifest Files in TrainingJobs	November 28, 2018
Configuring notebook instances	You can use shell scripts to configure notebook instances when you create or start them. For more information, see Customize a Notebook Instance.	May 1, 2018
Disable direct internet access	You can now disable direct internet access for notebook instances. For more information, see Notebook Instances Are Enabled with Internet Access by Default.	March 15, 2018
Application Auto Scaling support	Amazon SageMaker now supports Application Auto Scaling for production variants. For information, see Automatically Scaling Amazon SageMaker SageMaker Models	February 28, 2018
TensorFlow 1.5 and MXNet 1.0 support (p. 1045)	Amazon SageMaker Deep Learning containers now support TensorFlow 1.5 and Apache MXNet 1.0.	February 27, 2018
BlazingText algorithm	Amazon SageMaker now supports the BlazingText algorithm.	January 18, 2018

KMS encryption support for training and hosting	Amazon SageMaker now supports KMS encryption for hosting instances and training model artifacts at rest. You can specify a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to a hosting endpoint by using the `KmsKeyId` request parameter in a call to CreateEndpointConfig. You can specify an AWS KMS key that Amazon SageMaker uses to encrypt training model artifacts at rest by setting the `KmsKeyId` field of the OutputDataConfig object you use to configure your training job.	January 17, 2018
CloudTrail support	Amazon SageMaker now supports logging with AWS CloudTrail.	January 11, 2018
DeepAR Forecasting algorithm	Amazon SageMaker now supports the DeepAR algorithm for time series forecasting.	January 8, 2018

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.