
Amazon SageMaker

Developer Guide



Amazon SageMaker: Developer Guide

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is Amazon SageMaker?	1
Are You a First-time User of Amazon SageMaker?	1
How It Works	2
Machine Learning with Amazon SageMaker	2
How It Works: Next Topic	4
Explore and Preprocess Data	4
How It Works: Next Topic	4
Model Training	4
Incremental Training	7
Monitor and Analyze Training Jobs Using Metrics	12
How It Works: Next Topic	16
Model Deployment	16
Hosting Services	17
Batch Transform	19
Validating Models	21
How It Works: Next Topic	22
Programming Model	22
How It Works: Next Topic	23
Getting Started	24
Step 1: Setting Up	24
Step 1.1: Create an Account	24
Step 1.2: Create an S3 Bucket	25
Step 2: Create an Amazon SageMaker Notebook Instance	26
Step 2.1: (Optional) Customize a Notebook Instance	29
Next Step	30
Step 3: Train and Deploy a Model	30
Step 3.1: Create a Jupyter Notebook	31
Step 3.2: Download, Explore, and Transform Data	32
Step 3.3: Train a Model	34
Step 3.4: Deploy the Model	37
Step 3.5: Validate the Model	44
Step 4: Clean up	48
Step 5: Additional Considerations	49
Automatic Model Tuning	50
How Hyperparameter Tuning Works	50
Defining Objective Metrics	51
Defining Hyperparameter Ranges	52
Example: Hyperparameter Tuning Job	53
Prerequisites	53
Create a Notebook	53
Get the sagemaker boto3 Client	54
Get the Amazon SageMaker Execution Role	54
Specify a Bucket and Data Output Location	55
Download, Prepare, and Upload Training Data	55
Configure and Launch a Hyperparameter Tuning Job	56
Monitor the Progress of a Hyperparameter Tuning Job	59
Clean up	61
Run a Warm Start Hyperparameter Tuning Job	61
Types of Warm Start Tuning Jobs	62
Warm Start Tuning Restrictions	63
Sample Notebook	63
Create a Warm Start Tuning Job	63
Design Considerations	65
Choosing the Number of Hyperparameters	66

Choosing Hyperparameter Ranges	66
Use Logarithmic Scales for Hyperparameters	66
Choosing the Best Degree of Parallelism	66
Running Training Jobs on Multiple Instances	66
Using Notebook Instances	67
Creating a Notebook Instance	67
Accessing Notebook Instances	68
Limit Access to a Notebook Instance by IP Address	68
Connect to a Notebook Instance Through a VPC Interface Endpoint	69
Using Example Notebooks	72
Set the Notebook Kernel	73
Installing External Libraries and Kernels in Notebook Instances	73
Using Built-in Algorithms	75
Common Information	77
Common Parameters	77
Common Data Formats	82
Suggested Instance Types	90
Logs	90
BlazingText	91
Input/Output Interface	92
EC2 Instance Recommendation	94
Sample Notebooks	95
Hyperparameters	95
Model Tuning	99
DeepAR Forecasting	100
Input/Output Interface	101
Recommended Best Practices	103
EC2 Instance Recommendations	103
Sample Notebooks	104
How It Works	104
Hyperparameters	106
Model Tuning	110
Inference Formats	112
Factorization Machines	114
Input/Output Interface	114
EC2 Instance Recommendation	115
Sample Notebooks	115
How It Works	115
Hyperparameters	116
Model Tuning	121
Inference Formats	122
Image Classification Algorithm	124
Input/Output Interface	124
EC2 Instance Recommendation	125
Sample Notebooks	126
How It Works	126
Hyperparameters	126
Model Tuning	131
IP Insights	133
Input/Output Interface	133
EC2 Instance Recommendation	134
Sample Notebooks	134
How It Works	135
Hyperparameters	136
Model Tuning	138
Data Formats	140
K-Means Algorithm	142

Input/Output Interface	142
EC2 Instance Recommendation	143
Sample Notebooks	143
How It Works	143
Hyperparameters	146
Model Tuning	148
Inference Formats	149
K-Nearest Neighbors	150
Input/Output Interface	150
Sample Notebooks	151
How It Works	151
EC2 Instance Recommendation	152
Hyperparameters	152
Model Tuning	154
Training Formats	155
Inference Formats	156
Latent Dirichlet Allocation (LDA)	159
Input/Output Interface	159
EC2 Instance Recommendation	160
Sample Notebooks	160
How It Works	160
Hyperparameters	162
Model Tuning	163
Linear Learner	164
Input/Output Interface	164
EC2 Instance Recommendation	165
Sample Notebooks	165
How It Works	165
Hyperparameters	166
Model Tuning	174
Inference Formats	176
Neural Topic Model (NTM)	178
Input/Output Interface	178
EC2 Instance Recommendation	179
Sample Notebooks	179
Hyperparameters	179
Model Tuning	182
Inference Formats	183
Object2Vec	184
Sample Notebooks	184
Input/Output Interface	185
EC2 Instance Recommendation	185
How It Works	185
Hyperparameters	187
Model Tuning	192
Training Formats	194
Inference Formats: Scoring	195
Inference Formats: Embeddings	196
Object Detection Algorithm	196
Input/Output Interface	197
EC2 Instance Recommendation	199
Sample Notebooks	199
How It Works	199
Hyperparameters	199
Model Tuning	203
Inference Formats	204
Principal Component Analysis (PCA)	206

Input/Output Interface	206
EC2 Instance Recommendation	206
Sample Notebooks	207
How It Works	207
Hyperparameters	208
Inference Formats	209
Random Cut Forest	210
Input/Output Interface	210
Instance Recommendations	211
Sample Notebooks	211
How It Works	211
Hyperparameters	214
Model Tuning	215
Inference Formats	216
Sequence to Sequence (seq2seq)	218
Input/Output Interface	218
EC2 Instance Recommendation	219
Sample Notebooks	220
How It Works	220
Hyperparameters	220
Model Tuning	228
XGBoost Algorithm	230
Input/Output Interface	231
EC2 Instance Recommendation	231
Sample Notebooks	232
How It Works	232
Hyperparameters	232
Model Tuning	238
Using Your Own Algorithms	241
Using Your Own Training Algorithms	242
How Amazon SageMaker Runs Your Training Image	242
How Amazon SageMaker Provides Training Information	243
Signalling Algorithm Success and Failure	245
How Amazon SageMaker Processes Training Output	245
Next Step	246
Using Your Own Inference Code	246
Using Your Own Inference Code (Hosting Services)	246
Using Your Own Inference Code (Batch Transform)	249
Example: Using Your Own Algorithms	251
Automatically Scaling Amazon SageMaker Models	252
Automatic Scaling Components	252
Required Permissions for Automatic Scaling	253
Service-Linked Role	253
Target Metric	254
Minimum and Maximum Capacity	254
Cooldown Period	254
Before You Begin	255
Related Topics	255
Configure Automatic Scaling for a Variant	255
Configure Automatic Scaling for a Variant (Console)	255
Configure Automatic Scaling for a Variant (AWS CLI or the Application Auto Scaling API)	256
Editing a Scaling Policy	261
Editing a Scaling Policy (Console)	261
Editing a Scaling Policy (AWS CLI or Application Auto Scaling API)	262
Deleting a Scaling Policy	262
Deleting a Scaling Policy (Console)	262
Deleting a Scaling Policy (AWS CLI or Application Auto Scaling API)	262

Load Testing	263
Determine the Performance Characteristics of a Variant	264
Calculate the Target SageMakerVariantInvocationsPerInstance	264
Additional Considerations	265
Test Your Automatic Scaling Configuration	265
Updating Endpoints Configured for Automatic Scaling	265
Deleting Endpoints Configured for Automatic Scaling	265
Using Step Scaling Policies	265
Scaling In When There Is No Traffic	265
Using TensorFlow	267
TensorFlow Model Training Code	267
Examples: Using Amazon SageMaker with TensorFlow	270
Example 1: Using the tf.estimator	271
Using Apache MXNet	277
Apache MXNet Model Training Code	277
Examples: Using Amazon SageMaker with Apache MXNet	283
Example 1: Using the Module API	283
Using Chainer	290
Using PyTorch	291
Using Apache Spark	292
Downloading the Amazon SageMaker Spark Library	292
Integrating Your Apache Spark Application with Amazon SageMaker	293
Example 1: Amazon SageMaker with Apache Spark	294
Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark ..	298
Using the SageMakerEstimator in a Spark Pipeline	299
Additional Examples: Amazon SageMaker with Apache Spark	300
Amazon SageMaker Libraries	301
Authentication and Access Control	302
Authentication	302
Access Control	303
Overview of Managing Access	303
Amazon SageMaker Resources and Operations	304
Understanding Resource Ownership	304
Managing Access to Resources	305
Specifying Policy Elements: Resources, Actions, Effects, and Principals	306
Specifying Conditions in a Policy	307
Using Identity-Based Policies (IAM Policies)	307
Permissions Required to Use the Amazon SageMaker Console	308
AWS Managed (Predefined) Policies for Amazon SageMaker	309
Control Access to Amazon SageMaker Resources by Using Tags	310
Amazon SageMaker API Permissions Reference	312
Amazon SageMaker Roles	315
CreateNotebookInstance API: Execution Role Permissions	316
CreateHyperParameterTuningJob API: Execution Role Permissions	319
CreateTrainingJob API: Execution Role Permissions	321
CreateModel API: Execution Role Permissions	322
Using the AWS Managed Permission Policy (<code>AmazonSageMakerFullAccess</code>) for an Execution Role	324
Monitoring	325
Monitoring with CloudWatch	325
Logging with CloudWatch	328
Logging Amazon SageMaker API Calls with AWS CloudTrail	329
Amazon SageMaker Information in CloudTrail	329
Operations Performed by Automatic Model Tuning	330
Understanding Amazon SageMaker Log File Entries	330
Best Practices	332
Deployment Best Practices	332

Security	333
Notebook Instance Security	333
Notebook Instances Are Internet-Enabled by Default	333
Connect to Amazon SageMaker Through a VPC Interface Endpoint	334
Protect Training Jobs by Using an Amazon Virtual Private Cloud	335
Configuring a Training Job for Amazon VPC Access	335
Configuring Your Private VPC for Amazon SageMaker Training	336
Protect Endpoints by Using an Amazon Virtual Private Cloud	337
Configuring a Model for Amazon VPC Access	338
Configuring Your Private VPC for Amazon SageMaker Hosting	338
Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud	340
Configuring a Batch Transform Job for Amazon VPC Access	340
Configuring Your Private VPC for Amazon SageMaker Batch Transform	341
Limits and Supported Regions	343
API Reference	344
Actions	344
Amazon SageMaker Service	345
Amazon SageMaker Runtime	469
Data Types	473
Amazon SageMaker Service	475
Amazon SageMaker Runtime	551
Common Errors	551
Common Parameters	553
Document History	556
AWS Glossary	558

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 It 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 [Getting Started \(p. 24\)](#)** – 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 [Step 3: Train a Model with a Built-in Algorithm and Deploy It \(p. 30\)](#).
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 TensorFlow or Apache MXNet scripts to train models. For an example, see [TensorFlow Example 1: Using the tf.estimator \(p. 271\)](#) and [Apache MXNet Example 1: Using the Module API \(p. 283\)](#).
 - **Use Amazon SageMaker directly from Apache Spark** – For information, see [Using Apache Spark with Amazon SageMaker \(p. 292\)](#).
 - **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 [Using Your Own Algorithms with Amazon SageMaker \(p. 241\)](#) to learn how Amazon SageMaker interacts with Docker containers, and for the Amazon SageMaker requirements for Docker images.
5. **See the [API Reference \(p. 344\)](#)** – This section describes the Amazon SageMaker API operations.

How It 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

- [Machine Learning with Amazon SageMaker \(p. 2\)](#)
- [Explore and Preprocess Data \(p. 4\)](#)
- [Training a Model with Amazon SageMaker \(p. 4\)](#)
- [Model Deployment in Amazon SageMaker \(p. 16\)](#)
- [Validating Machine Learning Models \(p. 21\)](#)
- [The Amazon SageMaker Programming Model \(p. 22\)](#)

How It Works: Next Topic

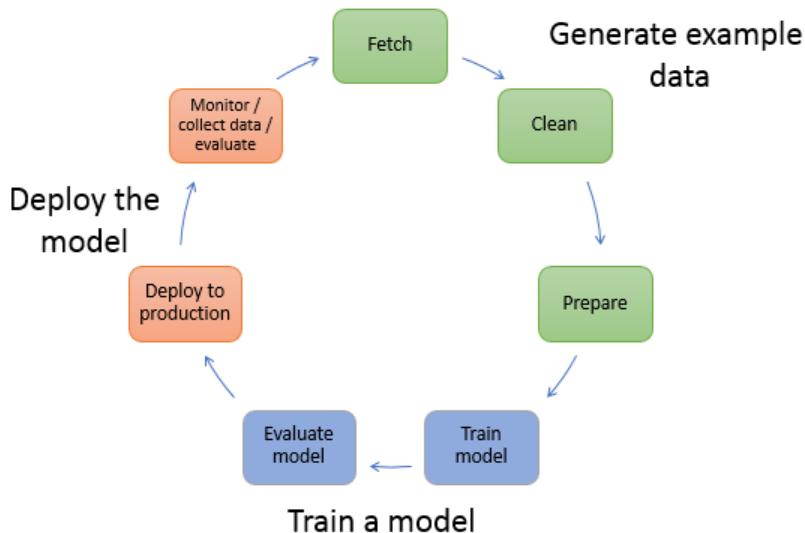
[Machine Learning with Amazon SageMaker \(p. 2\)](#)

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 up 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 [Using Built-in Algorithms with Amazon SageMaker \(p. 75\)](#).

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, small general-purpose instance to a distributed cluster of GPU instances. For more information, see [Training 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 [Deploying a Model on Amazon SageMaker Hosting Services \(p. 17\)](#).

Machine learning is a continuous cycle. After deploying a model, you monitor the inferences, then 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, 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 3.2.3: Transform the Training Dataset and Upload It to S3 \(p. 33\)](#).

To preprocess data use one of the following methods:

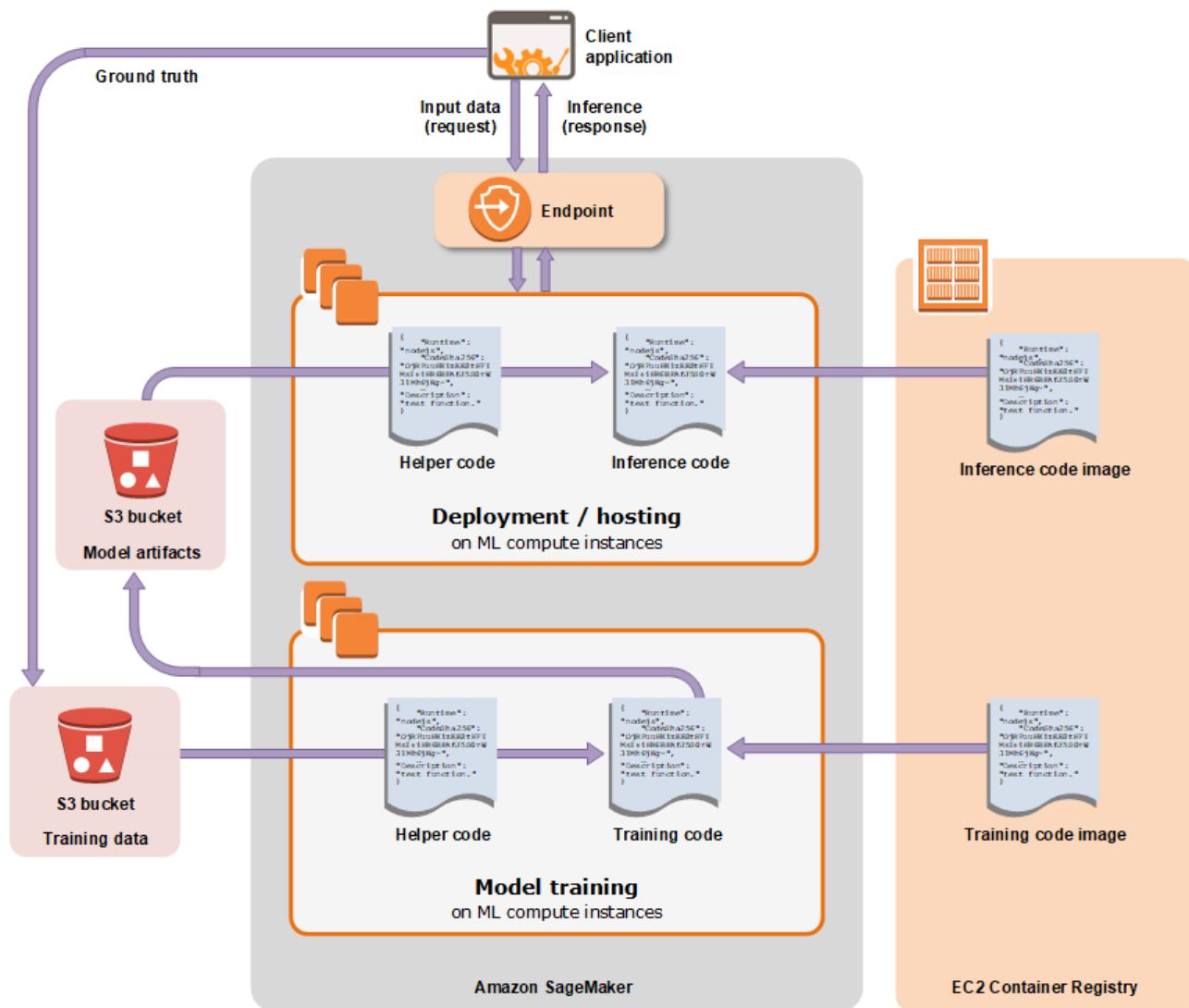
- Use a Jupyter notebook on an Amazon SageMaker notebook instance. You can also use the notebook instance to write code to create model training jobs, deploy models to Amazon SageMaker hosting, and test or validate your models. For more information, see [Using Notebook Instances \(p. 67\)](#)
- You can use a model to transform data by using Amazon SageMaker batch transform. For more information, see [Step 3.4.2: Deploy the Model to Amazon SageMaker Batch Transform \(p. 39\)](#).

How It Works: Next Topic

[Training a Model with Amazon SageMaker \(p. 4\)](#)

Training 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 [Algorithms Provided by Amazon SageMaker: Common Parameters \(p. 77\)](#).

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 [Using Built-in Algorithms with Amazon SageMaker \(p. 75\)](#). To try an exercise that uses an algorithm provided by Amazon SageMaker, see [Getting Started \(p. 24\)](#).

- **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 [Using Apache Spark with Amazon SageMaker \(p. 292\)](#).
- **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 [Using TensorFlow with Amazon SageMaker \(p. 267\)](#) and [Using Apache MXNet with Amazon SageMaker \(p. 277\)](#).
- **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 [Using Your Own Algorithms with Amazon SageMaker \(p. 241\)](#).

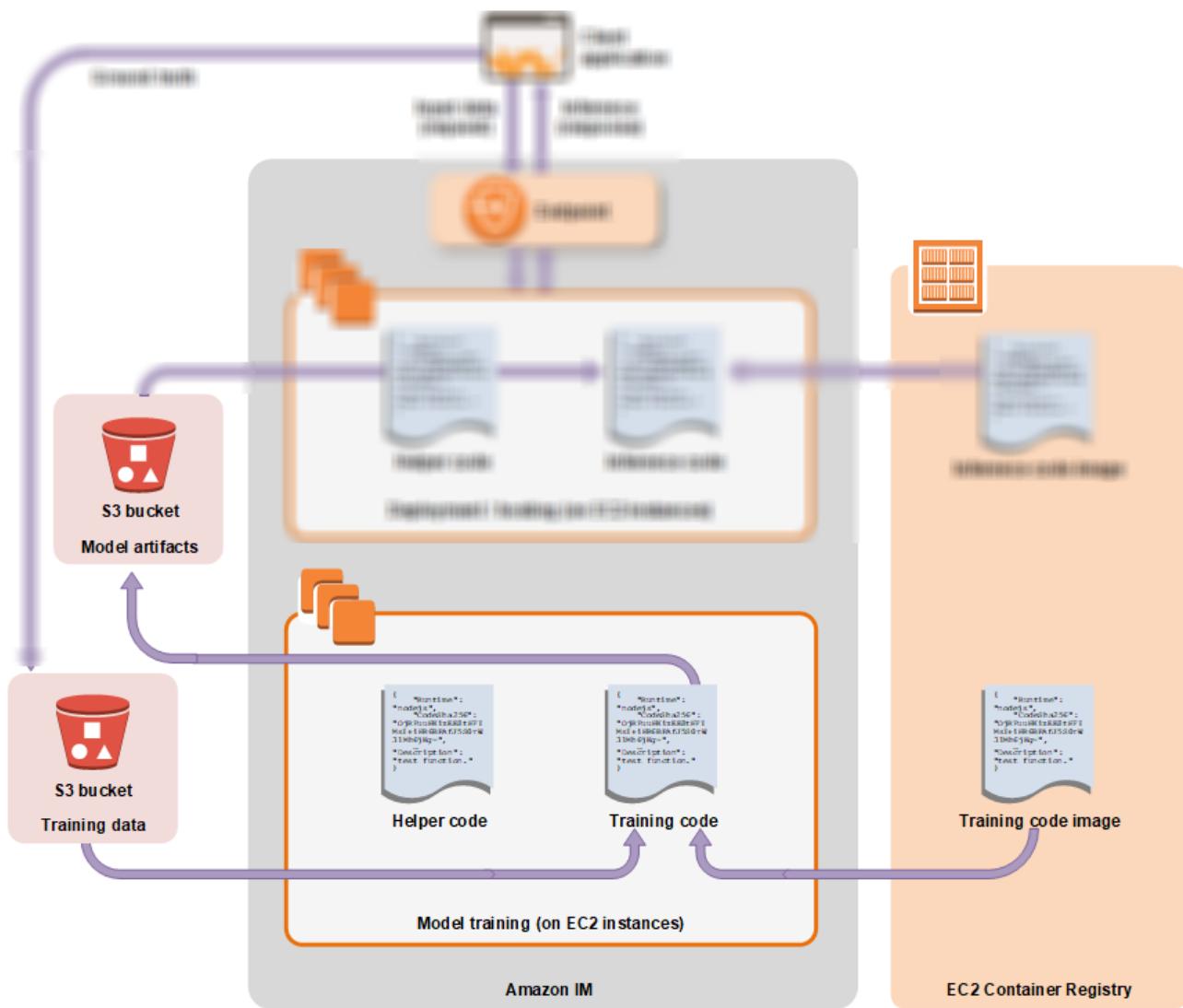
After you create the training job, Amazon SageMaker launches the ML compute instances and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket you specified for that purpose.

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 \(p. 371\)](#) 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 \(p. 530\)](#).

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.



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 [Training 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 \(p. 196\)](#) and [Image Classification Algorithm \(p. 124\)](#).

Topics

- [Performing Incremental Training \(Console\) \(p. 8\)](#)
- [Performing Incremental Training \(API\) \(p. 10\)](#)

Performing 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 [Algorithms Provided by Amazon SageMaker: Common Parameters \(p. 77\)](#).
- 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 [Using Built-in Algorithms with Amazon SageMaker \(p. 75\)](#).
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 is 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 3.4: Deploy the Model to Amazon SageMaker \(p. 37\)](#).

Performing 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.

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 fourour 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 3.4: Deploy the Model to Amazon SageMaker \(p. 37\)](#).

Monitor and Analyze Training Jobs Using Metrics

Monitor and analyze training jobs by viewing metrics that the training jobs send to Amazon CloudWatch. An Amazon SageMaker training job is an iterative process of teaching 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 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 \(p. 410\)](#) operation.

Topics

- [Training Metrics Sample Notebooks \(p. 12\)](#)
- [Define Training Metrics \(p. 12\)](#)
- [Monitor Training Job Metrics in CloudWatch \(p. 14\)](#)
- [Example: View a Training and Validation Curve \(p. 14\)](#)

Training Metrics Sample Notebooks

The following sample notebooks show how to view and plot training metrics.

- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/object2vec_sentence_similarity/object2vec_sentence_similarity.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/xgboost_abalone/xgboost_abalone.ipynb

For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Using Example Notebooks \(p. 72\)](#). 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 notebooks that show how to use training metrics are located in the **Introduction to Amazon algorithms** section, and named `object2vec_sentence_similarity.ipynb` and `xgboost_abalone.ipynb`. To open a notebook, click on its **Use** tab and select **Create copy**.

Define Training Metrics

Amazon SageMaker automatically parses the logs for metrics that built-in algorithms emit and sends those metrics to CloudWatch. If you use your own algorithm, when you configure the training job, you have to specify the metrics that you want Amazon SageMaker to send to CloudWatch. You do this by

specifying 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 AWS Management Console, the Amazon SageMaker Python SDK (<https://github.com/aws/sagemaker-python-sdk>), or the low-level Amazon SageMaker API.

Topics

- [Define Training Metrics \(Low-level Amazon SageMaker API\) \(p. 13\)](#)
- [Define Training Metrics \(Amazon SageMaker Python SDK\) \(p. 13\)](#)
- [Define Metrics \(Console\) \(p. 14\)](#)
- [Define Regular Expressions for Metrics \(p. 14\)](#)

Define 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 \(p. 477\)](#) input parameter that you pass to the [CreateTrainingJob \(p. 371\)](#) operation. For example, if your algorithm emits metrics for training error and validation error, and 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": ".*\\[[0-9]+\\]#011train-error:(\\S+).*"
        },
        {
            "Name": "validation:error",
            "Regex": ".*\\[[0-9]+\\]#011validation-error:(\\S+).*"
        }
    ]
}
```

For more information about defining and running a training job by using the low-level Amazon SageMaker API, see [Step 3.3.2: Create a Training Job \(p. 34\)](#).

Define 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 your algorithm emits metrics for training error and validation error, and you want to monitor both of those 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': '.*\\[[0-9]+\\]#011train-error:(\\S+).*'},
                  {'Name': 'validation:error', 'Regex': '.*\\[[0-9]+\\]#011validation-error:(\\S+).*'}
```

```
    ]  
)
```

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 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) under **Metrics**.

Define 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, make sure that the algorithm writes the metrics you want to capture to logs, and that you define a regular expression that accurately searches the logs to capture the values of the metrics that you want to send to CloudWatch metrics.

Monitor Training Job Metrics in CloudWatch

You can monitor the metrics that a training job emits in real time by using the CloudWatch console.

To monitor training job metrics in CloudWatch

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*.

Example: View a Training and Validation Curve

A common way to analyze the performance of a training job is to plot a training curve against a 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 it makes predictions by calculating predictions for the validation set. Viewing a graph that shows the accuracy for both the training and validation set over time can help you to train your model. For example, if the training accuracy continues to increase over time, but, at some point, the validation accuracy starts to decrease, you are likely overfitting your model. To address this, you might consider making adjustments to your model, such as increasing *regularization*.

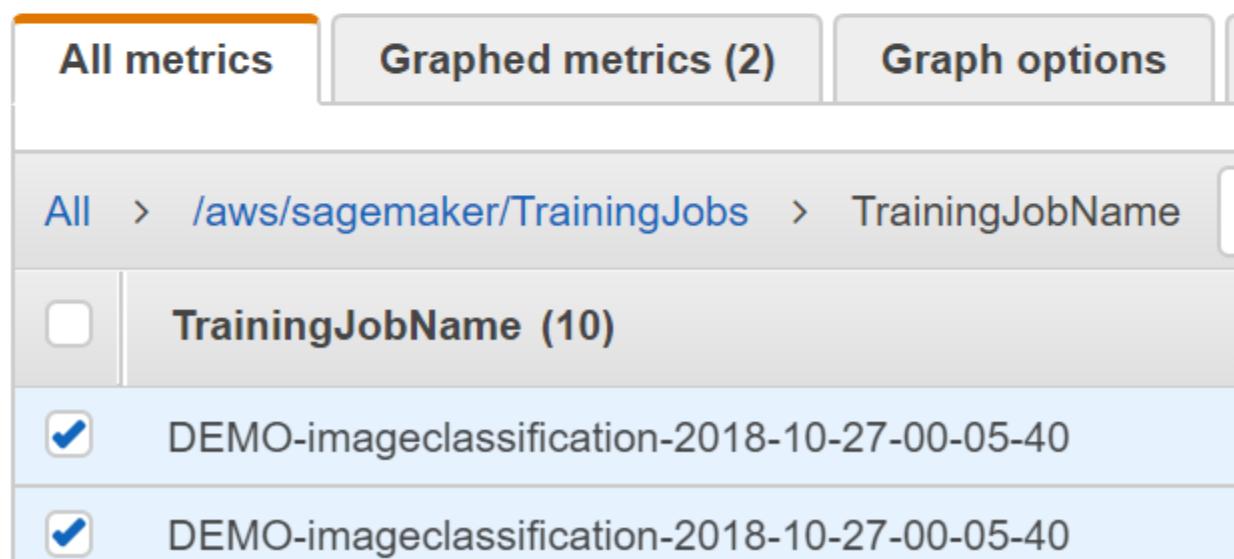
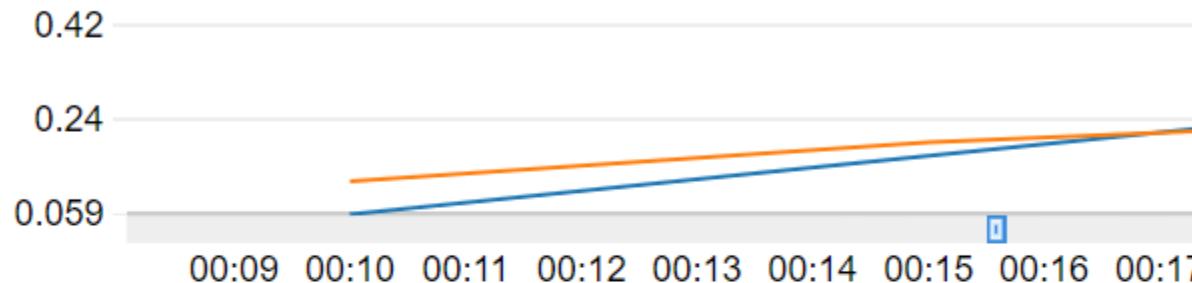
For this example, use the **Image-classification-full-training** example that is in the **Example notebooks** section of your Amazon SageMaker notebook instance. If you do not already have a Amazon SageMaker notebook instance, create one by following the instructions at [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 26\)](#). You can also see this example notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/imageclassification_caltech/Image-classification-fulltraining.ipynb. You also need an Amazon S3 bucket to store the training data and for the model output. If you have not already created a bucket to use for Amazon SageMaker, create one by following the instructions at [Step 1.2: Create an S3 Bucket \(p. 25\)](#).

To plot a training and validation error curve:

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 you want to use, and then choose **Open**.
4. On the dashboard of your notebook instance, choose **SageMaker Examples**.
5. Expand the **Introduction to Amazon Algorithms** section, and then next to **Image-classification-full-training.ipynb**, choose **Use**.
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, select the metric names **train:accuracy** and **validation:accuracy** for the training job you created in the notebook.
13. On the graph, choose a region where you see the values of the metric to zoom in on that section. You should see something like the following:

Untitled graph



A screenshot of the Amazon SageMaker Metrics page. At the top, there are three tabs: "All metrics" (highlighted in orange), "Graphed metrics (2)", and "Graph options". Below the tabs, the breadcrumb navigation shows "All > /aws/sagemaker/TrainingJobs > TrainingJobName". Under "TrainingJobName", there is a list of training jobs:

checkbox	TrainingJobName (10)
<input type="checkbox"/>	DEMO-imageclassification-2018-10-27-00-05-40
<input checked="" type="checkbox"/>	DEMO-imageclassification-2018-10-27-00-05-40

How It Works: Next Topic

[Model Deployment in Amazon SageMaker \(p. 16\)](#)

Model Deployment 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.

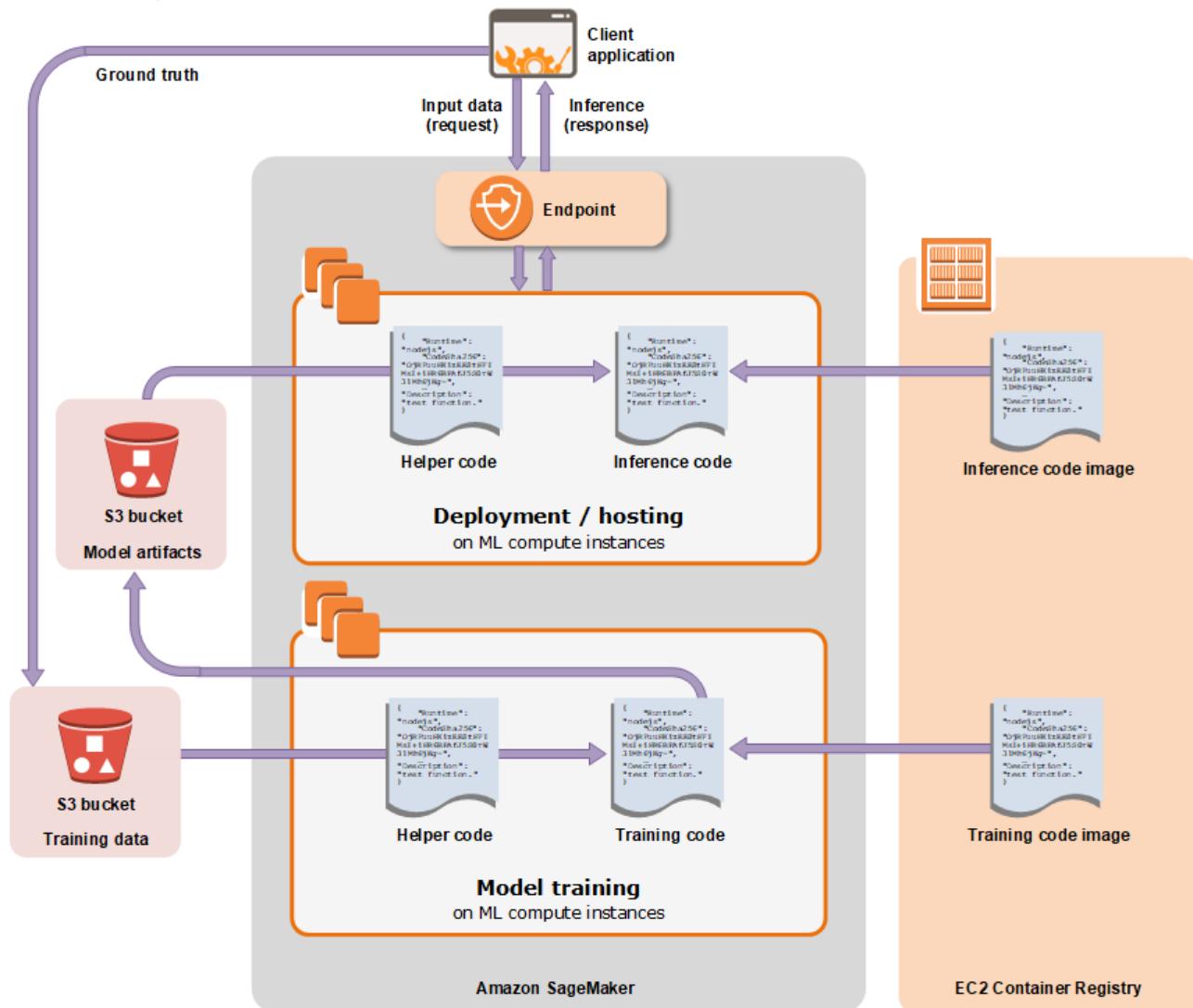
- To get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics

- [Deploying a Model on Amazon SageMaker Hosting Services \(p. 17\)](#)
- [Getting Inferences by Using Amazon SageMaker Batch Transform \(p. 19\)](#)

Deploying 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:

- 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 \(p. 359\)](#) 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 them.

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 \(p. 352\)](#) 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 \(p. 349\)](#) 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 \(p. 470\)](#) API.

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.

Considerations 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 \(p. 82\)](#).
- 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 \(p. 523\)](#).
- You can configure a `ProductionVariant` to use Application Auto Scaling. For information about configuring automatic scaling, see [Automatically Scaling Amazon SageMaker Models \(p. 252\)](#).
- 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 \(p. 461\)](#) and [UpdateEndpointWeightsAndCapacities \(p. 463\)](#).

- 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 [Getting Inferences by Using Amazon SageMaker Batch Transform \(p. 19\)](#)

How It Works: Next Topic

[Validating Machine Learning Models \(p. 21\)](#)

Getting Inferences by Using Amazon SageMaker Batch Transform

Get inferences for an entire dataset by using Amazon SageMaker batch transform. Batch transform uses a trained model to get inferences on a dataset that is stored in Amazon S3, and saves the inferences in an S3 bucket that you specify when you create a batch transform job. Batch transform manages all compute resources necessary to get inferences. This includes launching instances and deleting them after the transform job completes.

To perform a batch transform, create a transform job, which includes the following information:

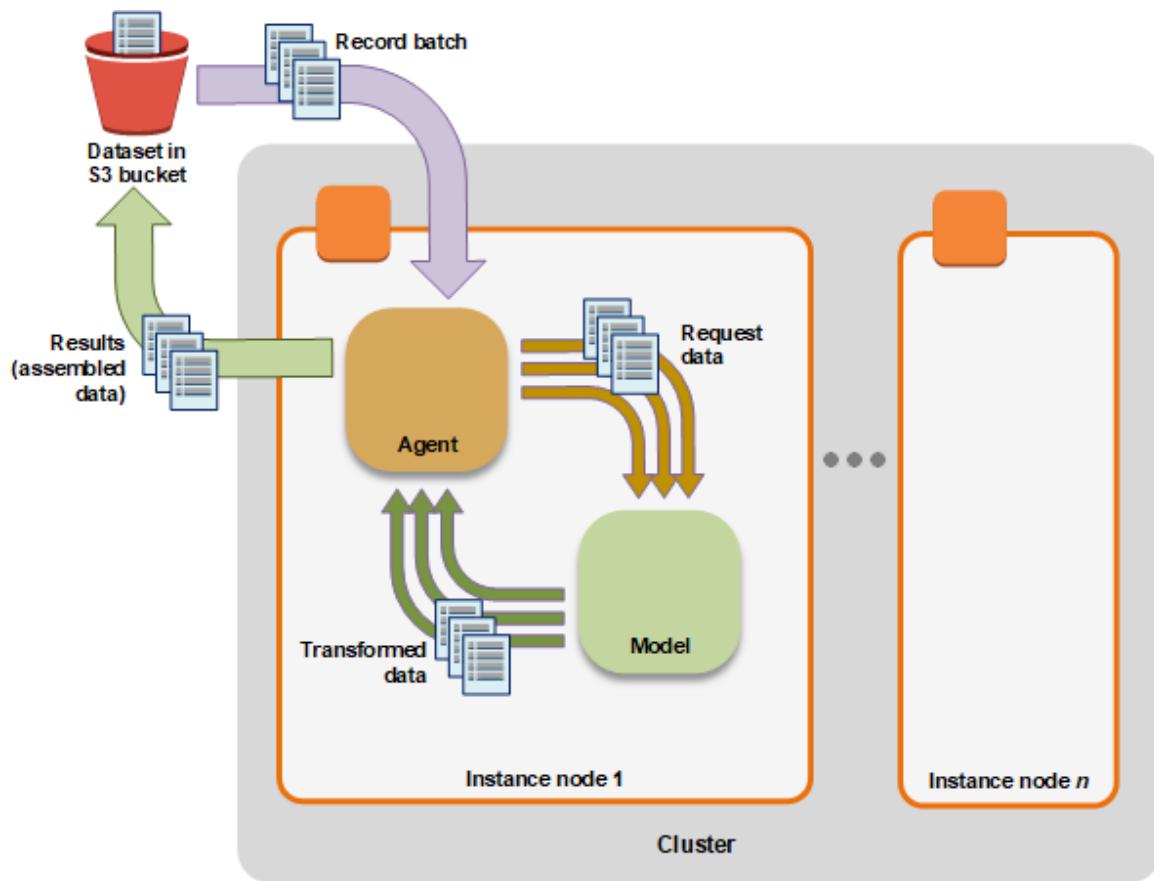
- The path to the S3 bucket where you've stored the data to transform.
- The compute resources that you want Amazon SageMaker to use for the transform job. *Compute resources* are ML compute instances that are managed by Amazon SageMaker.
- The path to the S3 bucket where you want to store the output of the job.
- The name of the model that you want to use in the transform job.

For examples on how to use batch transform, see [Step 3.4.2: Deploy the Model to Amazon SageMaker Batch Transform \(p. 39\)](#).

Batch transform is ideal for situations where:

- You want to get inferences for an entire dataset and store them online.
- You don't need a persistent endpoint that applications (for example, web or mobile apps) can call to get inferences.
- You don't need the sub-second latency that Amazon SageMaker hosted endpoints provide.
- You want to preprocess your data before using the data to train a new model or generate inferences.

The following diagram illustrates the workflow of a batch transform job:



Considerations for Using Amazon SageMaker Batch Transform

- You can create a transform job by using the Amazon SageMaker console or the API. For more information, see [CreateTransformJob \(p. 376\)](#) API.
- After you create a transform job, Amazon SageMaker launches the ML compute instances and uses the model you specify to transform the input data. It stores the results and other output in the S3 bucket that you specified.
- Amazon SageMaker uses [Multipart Upload API](#) to upload output data results from a transform job to S3. Typically all multipart uploads run to completion or are stopped in case of an error the multipart upload is removed from S3. However there are cases such as a network outage where a multipart upload remains incomplete remains in S3. To avoid extra storage charges in these cases, we recommend that you add the [S3 bucket policy](#) to remove incomplete multipart uploads that are stored in the output S3 buckets.
- For testing model variants, create separate transform jobs for each variant using a validation data set. You can then analyze the results using metrics. Make sure you specify a different `ModelName` and a unique `S3OutputPath` location for each transform job.
- For large datasets or data of indeterminate size, such as a video stream, you can create an *infinite stream*. An infinite stream is when a transform Job is set to continuously input and transform data. The transform job completes either when all the data is transformed or when it is instructed to stop. This approach works only with models from supported algorithms. To create an infinite stream using the API, in the `CreateTransformJob` request: set `SplitType` to `None` and set `MaxPayloadInMB` to `0`. To create an infinite stream using the console, choose **None** for **Split type** and set **Max payload size (MB)** to `0`. Amazon SageMaker interprets a `MaxPayloadInMB` of `0` as no limit on the payload size.

How It Works: Next Topic

[Validating Machine Learning Models \(p. 21\)](#)

Validating Machine Learning Models

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 \(p. 22\)](#)

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 \(p. 344\)](#).

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 TensorFlow or Apache MXNet 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 [Using Apache MXNet with Amazon SageMaker \(p. 277\)](#) and [Using TensorFlow with Amazon SageMaker \(p. 267\)](#).
- **The AWS SDK**—The SDKs provide methods that correspond to the Amazon SageMaker API (see [Actions \(p. 344\)](#)). 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 [Getting Started \(p. 24\)](#), 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 [Getting Started \(p. 24\)](#). For more information about these libraries, see [Amazon SageMaker Libraries \(p. 301\)](#).

- **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 [Using Apache Spark with Amazon SageMaker \(p. 292\)](#).

How It Works: Next Topic

[Getting Started \(p. 24\)](#)

Getting Started

In this section, you set up an AWS account, create your first Amazon SageMaker notebook instance, and train a model. You train the model using an algorithm provided by Amazon SageMaker, deploy it, and validate it by sending inference requests to the model's endpoint.

You use this notebook instance for all of the exercises in this guide.

If you're new to Amazon SageMaker, we recommend that you read [How It Works \(p. 2\)](#) before creating your notebook instance. For general information about notebook instances, see [Explore and Preprocess Data \(p. 4\)](#).

Topics

- [Step 1: Setting Up \(p. 24\)](#)
- [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 26\)](#)
- [Step 3: Train a Model with a Built-in Algorithm and Deploy It \(p. 30\)](#)
- [Step 4: Clean up \(p. 48\)](#)
- [Step 5: Additional Considerations: Integrating Amazon SageMaker Endpoints into Internet-facing Applications \(p. 49\)](#)

Step 1: Setting Up

In this section, you set up an AWS account and create an Amazon S3 bucket. You use this bucket to store training data and the results of model training, called model artifacts.

Topics

- [Step 1.1: Create an AWS Account and an Administrator User \(p. 24\)](#)
- [Step 1.2: Create an S3 Bucket \(p. 25\)](#)

Step 1.1: Create an AWS Account and an Administrator User

Before you use Amazon SageMaker for the first time, complete the following tasks:

Topics

- [Step 1.1.1: Create an AWS Account \(p. 24\)](#)
- [Step 1.1.2: Create an IAM Administrator User and Sign In \(p. 25\)](#)

Step 1.1.1: Create 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://aws.amazon.com/>, and then choose **Create an AWS Account**.

Note

If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.

2. Follow the online instructions.

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

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

Step 1.1.2: Create an IAM Administrator User and Sign In

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 [Authentication and Access Control for Amazon SageMaker \(p. 302\)](#).

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 1.2: Create an S3 Bucket \(p. 25\)](#)

Step 1.2: Create an S3 Bucket

In exercises where you create a model training job, you save the following in an Amazon S3 bucket:

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

You can store the training data and artifacts in a single bucket or in two separate buckets. For exercises in this guide, one bucket is sufficient. You can use existing buckets or create new ones.

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 this bucket. You grant permission with an IAM role, which you create in the next step (as part of creating an Amazon SageMaker notebook instance). This IAM role automatically gets permissions to access any bucket with `sagemaker` in the name through the `AmazonSageMakerFullAccess` policy that Amazon SageMaker attaches to the role.

Next Step

[Step 2: Create an Amazon SageMaker Notebook Instance \(p. 26\)](#)

Step 2: Create an Amazon SageMaker Notebook Instance

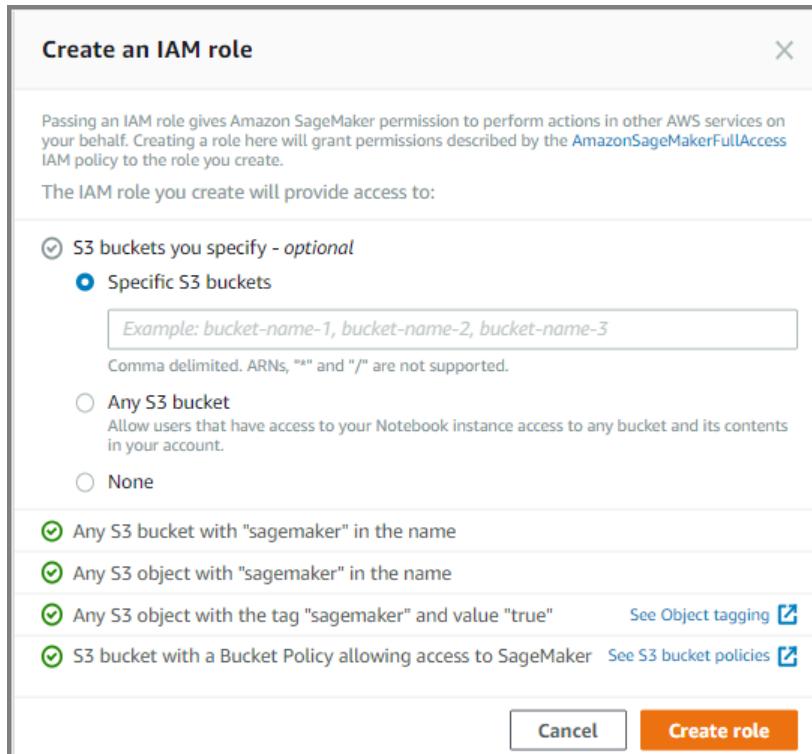
An Amazon SageMaker notebook instance is a fully managed machine learning (ML) EC2 compute instance running the Jupyter Notebook App. For more information, see [Explore and Preprocess Data \(p. 4\)](#).

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:
 - a. For **Notebook instance name**, type `ExampleNotebookInstance`.
 - b. For **Instance type**, choose `ml.t2.medium`.
 - c. For **IAM role**, create an IAM role.
 - i. Choose **Create a new role**.



- ii. (Optional) If you want to use S3 buckets other than the one you created in Step 1 of this tutorial to store your input data and output, choose them.

In Step 1 of this tutorial, you created an S3 bucket with `sagemaker` in its name. This IAM role automatically has permissions to use that bucket. The `AmazonSageMakerFullAccess` policy, which Amazon SageMaker attaches to the role, gives the role those permissions.

The bucket that you created in Step 1 is sufficient for the model training exercise in Getting Started. However, as you explore Amazon SageMaker, you might want to access other S3 buckets from your notebook instance. Give Amazon SageMaker permissions to access those buckets.

To access more S3 buckets from your Amazon SageMaker notebook instance

- A. If you're not concerned about users in your AWS account accessing your data, choose **Any S3 bucket**.
- B. If your account has sensitive data (such as Human Resources information), restrict access 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 [Using the AWS Managed Permission Policy \(AmazonSageMakerFullAccess\) for an Execution Role \(p. 324\)](#).

- iii. Choose **Create role**.

Amazon SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmSS`. 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/>.
The following policies are attached to the role:

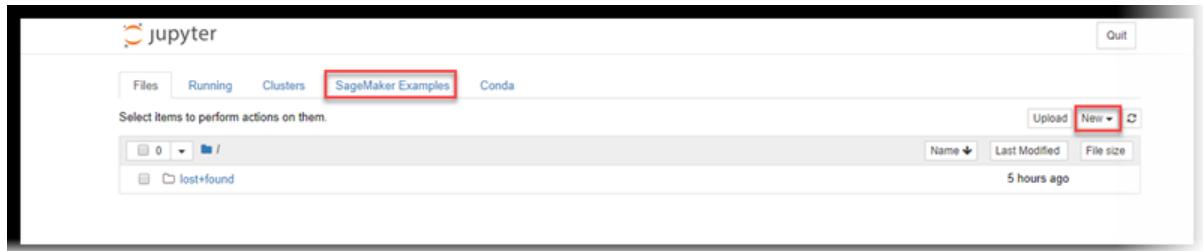
- A trust policy that allows Amazon SageMaker to assume the role.
- The `AmazonSageMakerFullAccess` AWS managed policy.
- If you specified access to additional S3 bucket(s) when creating theis role, the customer managed policy attached to the role. The name of the customer managed policy is `AmazonSageMaker-ExecutionPolicy-YYYYMMDDTHHmSS`.

For more information about creating your own IAM role, see [Amazon SageMaker Roles \(p. 315\)](#).

- d. (Optional) Choose to access resources in a Virtual Private Cloud (VPC).

To access resources in your VPC from the notebook instance

- Choose the **VPC** and a **SubnetId**.
 - For **Security Group**, choose your VPCs default security group. For the exercises in this guide, the inbound and outbound rules of the default security group are sufficient.
 - To enable 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 \(VPC\)](#)
 - If you chose to access resources from your VPC, enable direct internet access. For **Direct internet access**, choose **Enable**. Otherwise, this notebook instance won't have internet access. 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 [Notebook Instances Are Internet-Enabled by Default \(p. 333\)](#).
 - (Optional) To use shell scripts that run when you create or start the instance, specify a lifecycle configuration. For information, see [Step 2.1: \(Optional\) Customize a Notebook Instance \(p. 29\)](#)
 - (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.
 - 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 16384 GB, in 1 GB increments.
 - 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 \(p. 362\)](#) API.
- When the status of the notebook instance is **InService**, choose **Open** next to its name to open the Jupyter dashboard.



The dashboard provides access to:

- A tab that contains sample notebooks. To use a sample notebook, on the **SageMaker Examples** tab, choose the sample notebook you want to use, and choose **use**. For information about the sample notebooks, see the Amazon SageMaker [GitHub repository](#).
- The kernels for Jupyter, including those that provide support for Python 2 and 3, Apache MXNet, TensorFlow, and PySpark. To choose a kernel for your notebook instance, use the **New** menu.

For more information, see [The Jupyter notebook](#).

Step 2.1: (Optional) 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.

Note

Each 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].

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 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.

Next Step

You are now ready to train your first model. For step-by-step instructions, see [Step 3: Train a Model with a Built-in Algorithm and Deploy It \(p. 30\)](#).

Next Step

You are now ready to train your first model. For step-by-step instructions, see [Step 3: Train a Model with a Built-in Algorithm and Deploy It \(p. 30\)](#).

Step 3: Train a Model with a Built-in Algorithm and Deploy It

Now train and deploy your first machine learning model with Amazon SageMaker. For model training, you use the following:

- The MNIST dataset of images of handwritten, single digit numbers—This dataset provides 60,000 example images of handwritten single-digit numbers and a test dataset of 10,000 images. You provide this dataset to the k-means algorithm for model training. For more information, see [MNIST Dataset](#).
- A built-in algorithm—You use the k-means algorithm provided by Amazon SageMaker. K-means is a clustering algorithm. During model training, the algorithm groups the example data of handwritten numbers into 10 clusters (one for each number, 0 through 9). For more information about the algorithm, see [K-Means Algorithm \(p. 142\)](#).

In this exercise, you do the following:

1. Download the MNIST dataset to your Amazon SageMaker notebook instance, then review the data and preprocess it. For efficient training, you convert the dataset from the `numpy.array` format to the `RecordIO protobuf` format.
2. Start an Amazon SageMaker training job.
3. Deploy the model in Amazon SageMaker.
4. Validate the model by sending inference requests to the model's endpoint. You send images of handwritten, single-digit numbers. The model returns the number of the cluster (0 through 9) that the images belong to.

Important

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

- The high-level Python library provided by Amazon SageMaker
- The AWS SDK for Python (Boto)

The high-level library abstracts several implementation details, and is easy to use. This exercise provides separate code examples using both libraries. If you're a first-time Amazon SageMaker user, we recommend that you use the high-level Python library. For more information, see [The Amazon SageMaker Programming Model \(p. 22\)](#).

There are two ways to use this exercise:

- Follow the steps to create, deploy, and validate the model. You create a Jupyter notebook in your Amazon SageMaker notebook instance, and copy code, paste it into the notebook, and run it.
- If you're familiar with using sample notebooks, open and run the following example notebooks that Amazon SageMaker provides in the **SageMaker Python SDK** section of the **SageMaker Examples** tab of your notebook instance:

```
kmeans_mnist.ipynb  
kmeans_mnist_lowlevel.ipynb
```

Topics

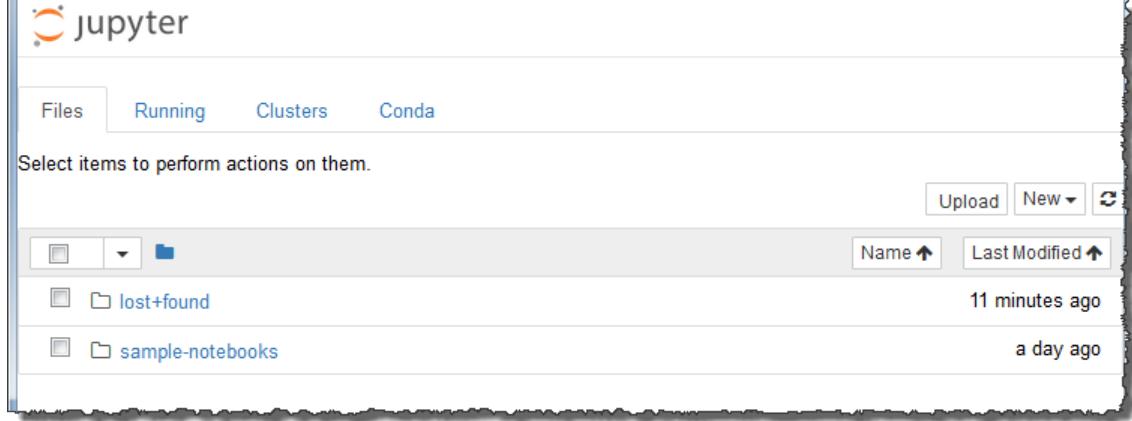
- [Step 3.1: Create a Jupyter Notebook and Initialize Variables \(p. 31\)](#)
- [Step 3.2: Download, Explore, and Transform the Training Data \(p. 32\)](#)
- [Step 3.3: Train a Model \(p. 34\)](#)
- [Step 3.4: Deploy the Model to Amazon SageMaker \(p. 37\)](#)
- [Step 3.5: Validate the Model \(p. 44\)](#)

Step 3.1: Create a Jupyter Notebook and Initialize Variables

In this section, you create a Jupyter notebook in your Amazon SageMaker notebook instance and initialize variables.

To create a Jupyter notebook

1. Create the notebook.
 - a. Sign in to the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
 - b. Open the notebook instance, by choosing **Open** next to its name. The Jupyter notebook server page appears:



- c. To create a notebook, in the **Files** tab, choose **New**, and **conda_python3**. This pre-installed environment includes the default Anaconda installation and Python 3.
d. Name the notebook.
2. Copy the following Python code and paste it into your notebook. Add the name of the S3 bucket that you created in [Step 1: Setting Up \(p. 24\)](#), and run the code. The `get_execution_role` function retrieves the IAM role you created at the time of creating your notebook instance.

```
from sagemaker import get_execution_role
role = get_execution_role()
bucket = 'bucket-name' # Use the name of your S3 bucket here
```

Next Step

[Step 3.2: Download, Explore, and Transform the Training Data \(p. 32\)](#)

Step 3.2: Download, Explore, and Transform the Training Data

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

You transform the data by changing its format from `numpy.array` to `RecordIO`. The `RecordIO` format is more efficient for the algorithms provided by Amazon SageMaker. For information about the `RecordIO` format, see [Data Format](#).

Topics

- [Step 3.2.1: Download the MNIST Dataset \(p. 32\)](#)
- [Step 3.2.2: Explore the Training Dataset \(p. 33\)](#)
- [Step 3.2.3: Transform the Training Dataset and Upload It to S3 \(p. 33\)](#)

Step 3.2.1: 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, numpy, urllib.request, json

# 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')
```

The code does the following:

1. Downloads the MNIST dataset (`mnist.pkl.gz`) from the deeplearning.net website to your Amazon SageMaker notebook instance.
2. Unzips the file and reads the following three datasets into the notebook's memory:
 - `train_set`—You use these images of handwritten numbers to train a model.
 - `valid_set`—After you train the model, you validate it using the images in this dataset.
 - `test_set`—You don't use this dataset in this exercise.

Next Step

[Step 3.2.2: Explore the Training Dataset \(p. 33\)](#)

Step 3.2.2: 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. Simply display one of the images in the `train_set` dataset.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (2,10)

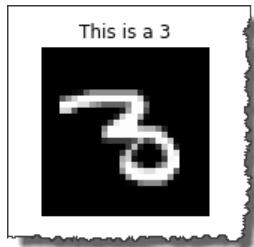
def show_digit(img, caption='', subplot=None):
    if subplot == None:
        _, (subplot) = plt.subplots(1,1)
    img = img.reshape((28,28))
    subplot.axis('off')
    subplot.imshow(img, cmap='gray')
    subplot.title(caption)

show_digit(train_set[0][30], 'This is a {}'.format(train_set[1][30]))
```

`train_set` contains the following data:

- `train_set[0]` contains images.
- `train_set[1]` contains labels.

The code uses the `matplotlib` library to get and display the 31st image from the training dataset.



Next Step

[Step 3.2.3: Transform the Training Dataset and Upload It to S3 \(p. 33\)](#)

Step 3.2.3: Transform the Training Dataset and Upload It to S3

For efficient model training, transform the dataset from the `numpy.array` format to the `RecordIO protobuf` format. The `RecordIO protobuf` format is more efficient for all of the algorithms provided by Amazon SageMaker.

Important

For this and subsequent steps, you can choose to use the high-level Python library provided by Amazon SageMaker or the low-level AWS SDK for Python (Boto). If you're a first-time user of Amazon SageMaker, we recommend that you follow the code examples for the high-level Python library.

To transform the dataset, choose one of the following options

- **Use the high-level Python library provided by Amazon SageMaker**

If you are using the high-level Python library, you skip this step and go to the next step. In the next section, you use the `fit` method for model training, which performs the necessary transformation and upload to S3 before starting a model training job.

- **Use the SDK for Python**

The following code first uses the high-level Python library function, `write_numpy_to_dense_tensor`, to convert the training data into the protobuf format, which is efficient for model training. Then the code uses the SDK for Python low-level API to upload data to S3.

```
%time
from sagemaker.amazon.common import write_numpy_to_dense_tensor
import io
import boto3

bucket = 'bucket-name' # Use the name of your s3 bucket here
data_key = 'kmeans_lowlevel_example/data'
data_location = 's3://{}{}'.format(bucket, data_key)
print('training data will be uploaded to: {}'.format(data_location))

# Convert the training data into the format required by the SageMaker KMeans algorithm
buf = io.BytesIO()
write_numpy_to_dense_tensor(buf, train_set[0], train_set[1])
buf.seek(0)

boto3.resource('s3').Bucket(bucket).Object(data_key).upload_fileobj(buf)
```

Next Step

[Step 3.3: Train a Model \(p. 34\)](#)

Step 3.3: Train a Model

To start model training, you send a request to the [CreateTrainingJob \(p. 371\)](#) API. In the request, you specify the Amazon Elastic Container Registry path to the training image, the location of the S3 bucket containing your training data, and the resources to use (the type and number of ML compute instances to launch).

Topics

- [Step 3.3.1: Choose the Training Algorithm \(p. 34\)](#)
- [Step 3.3.2: Create a Training Job \(p. 34\)](#)

Step 3.3.1: Choose the Training Algorithm

To choose the right algorithm for your model, you typically follow an evaluation process. For this exercise, you use the k-means algorithm provided by Amazon SageMaker, so no evaluation is required. For information about choosing algorithms, see [Using Built-in Algorithms with Amazon SageMaker \(p. 75\)](#).

Next Step

[Step 3.3.2: Create a Training Job \(p. 34\)](#)

Step 3.3.2: Create a Training Job

To train a model, Amazon SageMaker provides the [CreateTrainingJob \(p. 371\)](#) API. You provide the following information when making this API call:

- 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 [Algorithms Provided by Amazon SageMaker: Common Parameters \(p. 77\)](#). In the following examples, when using the high-level Python library, you don't need to explicitly specify this path. The `sagemaker.amazon.KMeans` object knows the path.
- Algorithm-specific hyperparameters—Specify algorithm-specific hyperparameters to influence the final quality of the model. For information, see [K-Means Hyperparameters \(p. 146\)](#).
- 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).

The low-level AWS SDK for Python provides the corresponding `create_training_job` method and the high-level Python library provide the `fit` method.

To train the model, choose one of the following options.

- **Use the high-level Python library provided by Amazon SageMaker.**

This Python library provides the `KMeans` estimator, which is a class in the `sagemaker.amazon.KMeans` module. To start model training, call the `fit` method.

1. Create an instance of the `sagemaker.amazon.KMeans` class.

```
from sagemaker import KMeans

data_location = 's3://{{}}/kmeans_highlevel_example/data'.format(bucket)
output_location = 's3://{{}}/kmeans_highlevel_example/output'.format(bucket)

print('training data will be uploaded to: {}'.format(data_location))
print('training artifacts will be uploaded to: {}'.format(output_location))

kmeans = KMeans(role=role,
                 train_instance_count=2,
                 train_instance_type='ml.c4.8xlarge',
                 output_path=output_location,
                 k=10,
                 data_location=data_location)
```

In the constructor, you specify the following parameters:

- `role`—The 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).
- `output_path`—The S3 location where Amazon SageMaker stores the training results.
- `train_instance_count` and `train_instance_type`—The type and number of ML compute instances to use for model training.
- `k`—The number of clusters to create. For more information, see [K-Means Hyperparameters \(p. 146\)](#).
- `data_location`—The S3 location where the high-level library uploads the transformed training data.

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

```
%%time

kmeans.fit(kmeans.record_set(train_set[0]))
```

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

The model training in this example takes about 15 minutes.

- **Use the SDK for Python.**

This low-level SDK for Python provides the `create_training_job` method, which maps to the [CreateTrainingJob \(p. 371\)](#) Amazon SageMaker API.

```
%%time
import boto3
from time import gmtime, strftime

job_name = 'kmeans-lowlevel-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Training job", job_name)

from sagemaker.amazon.amazon_estimator import get_image_uri
image = get_image_uri(boto3.Session().region_name, 'kmeans')

output_location = 's3://{}{}/kmeans_lowlevel_example/output'.format(bucket)
print('training artifacts will be uploaded to: {}'.format(output_location))

create_training_params = \
{
    "AlgorithmSpecification": {
        "TrainingImage": image,
        "TrainingInputMode": "File"
    },
    "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": output_location
    },
    "ResourceConfig": {
        "InstanceCount": 2,
        "InstanceType": "ml.c4.8xlarge",
        "VolumeSizeInGB": 50
    },
    "TrainingJobName": job_name,
    "HyperParameters": {
        "k": "10",
        "feature_dim": "784",
        "mini_batch_size": "500"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 60 * 60
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": data_location,
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None"
        }
    ]
}
```

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

sagemaker.create_training_job(**create_training_params)

status = sagemaker.describe_training_job(TrainingJobName=job_name)[ 'TrainingJobStatus' ]
print(status)

try:

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

The code uses a `Waiter` to wait until training is complete before returning.

You now have trained a model. The resulting artifacts are stored in your S3 bucket.

Next Step

[Step 3.4: Deploy the Model to Amazon SageMaker \(p. 37\)](#)

Step 3.4: Deploy the Model to Amazon SageMaker

To deploy a model to get predictions, you can choose one of two ways:

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

Topics

- [Step 3.4.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 37\)](#)
- [Step 3.4.2: Deploy the Model to Amazon SageMaker Batch Transform \(p. 39\)](#)

Step 3.4.1: Deploy the Model to Amazon SageMaker Hosting Services

Deploying a model in Amazon SageMaker is a 3-step process:

1. Create a model in Amazon SageMaker— Send a [CreateModel \(p. 359\)](#) 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. In the next step, you provide the model when you create an endpoint configuration.
2. Create an endpoint configuration— Send a [CreateEndpointConfig \(p. 352\)](#) request to provide the resource configuration for hosting. This includes the type and number of ML compute instances to launch for deploying the model. In the next step, you create an endpoint with the [CreateEndpoint \(p. 349\)](#) API using this endpoint configuration.
3. Create an endpoint— Send a [CreateEndpoint \(p. 349\)](#) request to create an endpoint. Amazon SageMaker launches the ML compute instances and deploys the model. In the response, Amazon

SageMaker returns an endpoint. Applications can send requests to this endpoint to get inferences from the model.

The low-level AWS SDK for Python provides corresponding methods. However, the high-level Python library provides the `deploy` method that does all these tasks for you.

To deploy the model, choose one of the following options.

- **Use the high-level Python library provided by Amazon SageMaker.**

The `sagemaker.amazon.KMeans` class provides the `deploy` method for deploying a model. It performs all three steps of the model deployment process.

```
%%time

kmeans_predictor = kmeans.deploy(initial_instance_count=1,
                                   instance_type='ml.m4.xlarge')
```

The `sagemaker.amazon.KMeans` instance knows the registry path of the image that contains the k-means inference code, so you don't need to provide it.

This is a synchronous operation. The method waits until the deployment completes before returning. It returns a `kmeans_predictor`.

- **Use the SDK for Python.**

The low-level SDK for Python provides methods that map to the underlying Amazon SageMaker API. To deploy the model, you make three calls.

1. Create an Amazon SageMaker model by identifying the location of model artifacts and the Docker image that contains inference code.

```
%%time
import boto3
from time import gmtime, strftime

model_name = job_name
print(model_name)

info = sagemaker.describe_training_job(TrainingJobName=job_name)
model_data = info['ModelArtifacts']['S3ModelArtifacts']

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

create_model_response = sagemaker.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.

```
from time import gmtime, strftime
```

```

endpoint_config_name = 'KMeansEndpointConfig-' + 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'}])

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

```

3. Create an Amazon SageMaker endpoint. This code uses a Waiter to wait until the deployment is complete before returning.

```

%%time
import time

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

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

try:
    sagemaker.get_waiter('endpoint_in_service').wait(EndpointName=endpoint_name)
finally:
    resp = sagemaker.describe_endpoint(EndpointName=endpoint_name)
    status = resp['EndpointStatus']
    print("Arn: " + resp['EndpointArn'])
    print("Create endpoint ended with status: " + status)

    if status != 'InService':
        message = sagemaker.describe_endpoint(EndpointName=endpoint_name)
        ['FailureReason']
        print('Create endpoint failed with the following error: {}'.format(message))
        raise Exception('Endpoint creation did not succeed')

```

Next Step

[Step 3.5: Validate the Model \(p. 44\)](#)

Step 3.4.2: Deploy the Model to Amazon SageMaker Batch Transform

Amazon SageMaker offers the ability to use batch transform either through the console or by using the API. For more information, see [Getting Inferences by Using Amazon SageMaker Batch Transform \(p. 19\)](#). The following examples assume that you have already run a training job to create a model (see [Step 3.3: Train a Model \(p. 34\)](#)) and created a Model based on the training output (see [Step 3.4.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 37\)](#)).

Topics

- [Using Batch Transform \(Console\) \(p. 40\)](#)
- [Using the Batch Transform API \(p. 41\)](#)

Using Batch Transform (Console)

To generate inferences, create transform jobs by using the console.

Creating a Batch Transform Job (Console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>
2. In the navigation pane, choose **Inference**, then choose **Batch transform jobs**.
3. Choose **Create batch transform job**.
4. For **Batch transform job configuration**, provide the following:
 - a. For **Job name**, type a name for the new batch transform job.
 - b. For **Model name**, choose **Choose model**, a list of available models appears.
 - c. Select the model that you want to use from the list of available models and choose **Choose model**.
 - d. For **Instance type**, choose the instance type that you want to use for the batch transform job. To use models created by built-in algorithms to transform moderately-sized datasets, ml.m4.xlarge or ml.m5.large should suffice.
 - e. For **Instance count**, type the number of compute instances to use for the transform job. For most use cases, 1 should suffice. By setting a higher value, you use more resources to reduce the time it takes to complete the transform job.
 - f. (Optional) For **Max concurrent transforms**, type the maximum number of parallel requests to use for each instance node. For most use cases, 1 should suffice. To allow Amazon SageMaker to determine the appropriate number of concurrent transforms, set the value to 0 or leave it blank.
 - g. (Optional) For **Batch strategy**, if you want to use only one record per batch, select **SingleRecord**. If you want to use the maximum number of records per batch, select **MultiRecord**.
 - h. (Optional) For **Max payload size (MB)**, type the maximum payload size, in MB. You can approximate an appropriate value by: dividing the size of your dataset by the number of records and multiply by the number of records you want per batch. The value you enter should be proportional to the number of records you want per batch. It is recommended to enter a slightly higher value to ensure the records will fit within the maximum payload size. The default value is 6 MB. For infinite stream mode, where a transform job is set to continuously input and transform data, set the value to 0.
 - i. (Optional) for **Environment variables**, specify a **key** and **value** for the internal Docker container to use. To add more entries to input more variables, choose **Add environment variables**. To remove an entry choose **Remove** next to the respective entry.
5. For **Input data configuration**, provide the following information:
 - a. For **S3 location**, type the path to the location where the input data is stored in Amazon S3.
 - b. For **S3 data type**, select the S3 data type. The default value is **ManifestFile**. If the object is a manifest file that contains a list of objects to transform, choose **ManifestFile**. To use all objects with a specified key name prefix, choose **S3Prefix**.
 - c. For **Compression type**, you specify the compression used on the input data. The default value is **None**. If the input data is uncompressed select **None**. If the input data is compressed using Gzip, select **Gzip**.
 - d. (Optional) For **Split type**, choose how to split the data into smaller batches for the transform job. The default is **None**. If you don't want to split the input data, choose **None**. If you want to split the data using newline characters, choose **Line**. If you want to split the data according to the RecordIO format, choose **RecordIO**. For infinite stream mode, where a transform job is set to continuously input and transform data, choose **None**.
 - e. (Optional) For **Content type**, specified the MIME type that you want to use.
6. 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](#).
 - c. (Optional) For **Accept**, specify the MIME type you want to be used for the output data.
7. (Optional) For **Tags**, add tags to the transform job. A tag is a label you assign to help manage your training jobs. A tag consists of a key and a value both of which you define.
8. Choose **Create job**.

[Viewing Details for a Batch Transform Job \(Console\)](#)

The following explain how to view details relating to a batch transform job:

- To see details about a batch transform job, such as job name or status, choose a batch transform job from the batch transform jobs table.
- To view the details of a model used in a batch transform job, on the batch transform job details page, choose the link under model name.

[Stopping a Batch Transform Job \(Console\)](#)

You can stop a batch transform job from the **Batch transform job** page or from the **Batch detail** page.

- **To stop a batch transform job using the Actions menu**
 1. Choose the checkbox near the batch transform job's name.
 2. In the **Actions** menu, choose **Stop**.
- **To stop a batch transform job from the Batch detail page**
 1. Choose the transformation job name to view its details.
 2. Choose **Stop**.

[Using the Batch Transform API](#)

There are several ways to interact with transform jobs. For more information, see [CreateTransformJob \(p. 376\)](#). You can use the high-level Python library or the low-level SDK (boto3).

[Using the high-level Python Library](#)

The following shows how to create a transform job using the high-level Python library:

```
import boto3
import sagemaker
import json

input_key = 'kmeans_batch_example/input/valid-data.csv'
input_location = 's3://{}{}'.format(bucket, input_key)
output_location = 's3://{}kmeans_batch_example/output'.format(bucket)

### Convert the validation set numpy array to a csv file and upload to s3
numpy.savetxt('valid-data.csv', valid_set[0], delimiter=',', fmt='%g')
s3_client = boto3.client('s3')
s3_client.upload_file('valid-data.csv', bucket, input_key)

# Initialize the transformer object
transformer = sagemaker.transformer.Transformer(
```

```

base_transform_job_name='Batch-Transform',
model_name=model_name,
instance_count=1,
instance_type='ml.c4.xlarge',
output_path=output_location
)
# To start a transform job:
transformer.transform(input_location, content_type='text/csv', split_type='Line')
# Then wait until transform job is completed
transformer.wait()

# To fetch validation result
s3_client.download_file(bucket, 'kmeans_batch_example/output/valid-data.csv.out', 'valid-
result')
with open('valid-result') as f:
    results = f.readlines()
print("Sample transform result: {}".format(results[0]))

```

Using the low-level SDK (boto3)

The following examples demonstrate how to interact with transform jobs using The Amazon SageMaker low-level SDK:

Using CreateTransformJob

To start using batch transform, you need to create a transform job. The low-level AWS SDK for Python provides the corresponding `create_transform_job` method. First, import the needed libraries (`boto3`) and classes (`gtime` and `strftime` from the `time` library). Second, set the needed variables (`job_name`, `bucket`, `prefix`, `sm`, and `model_name`). The `sm`, variable is especially important as it allows us to use the Amazon SageMaker API. Third you write the request. For information, see [CreateTransformJob \(p. 376\)](#).

To help prevent errors and improve readability, use the variables you created in the previous step in creating the request. Finally, use the request with `create_transform_job` to start a batch transform job. The following code shows how to use the `create_transform_job` operation:

```

import boto3
import sagemaker
import json
from urllib.parse import urlparse
from time import gmtime, strftime

batch_job_name = 'Batch-Transform-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
input_location = 's3://{}//kmeans_batch_example/input'.format(bucket)
output_location = 's3://{}//kmeans_batch_example/output'.format(bucket)

### Convert the validation set numpy array to a csv file and upload to s3
numpy.savetxt('valid-data.csv', valid_set[0], delimiter=',', fmt='%g')
s3_client = boto3.client('s3')
input_key = "{}//valid_data.csv".format(urlparse(input_location).path.lstrip('/'))
s3_client.upload_file('valid-data.csv', bucket, input_key)

### Create a transform job
sm = boto3.client('sagemaker')

request = \
{
    "TransformJobName": batch_job_name,
    "ModelName": model_name,
    "MaxConcurrentTransforms": 4,
}

```

```

        "MaxPayloadInMB": 6,
        "BatchStrategy": "MultiRecord",
        "TransformOutput": {
            "S3OutputPath": output_location
        },
        "TransformInput": {
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": input_location
                }
            },
            "ContentType": "text/csv",
            "SplitType": "Line",
            "CompressionType": "None"
        },
        "TransformResources": {
            "InstanceType": "ml.m4.xlarge",
            "InstanceCount": 1
        }
    }

sm.create_transform_job(**request)

print("Created Transform job with name: ", batch_job_name)

### Wait until job completion
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)

### Fetch transform output
output_key = "{}/valid_data.csv.out".format(urlparse(output_location).path.lstrip('/'))
s3_client.download_file(bucket, output_key, 'valid-result')
with open('valid-result') as f:
    results = f.readlines()
print("Sample transform result: {}".format(results[0]))

```

Using DescribeTransformJob

To see the details of a specific transform job, specify the name in a describe transform jobs request. The low-level AWS SDK for Python provides the corresponding `describe_transform_jobs` method. The following code shows how to use the `describe_transform_job` method:

```
response = sm.describe_transform_job(TransformJobName=job_name)
```

Using ListTransformJobs

To list previous transform jobs, specify criteria to list and to sort the list. The low-level AWS SDK for Python provides the corresponding `list_transform_jobs` method. The following code shows how to use the `list_transform_jobs` method:

```
import boto3
from time import gmtime, strftime

request = \
{
    "StatusEquals": "Completed",
    "SortBy": "CreationTime",
    "SortOrder": "Descending",
    "MaxResults": 20,
}

response = sm.list_transform_jobs(**request)
```

Tip

When using the `list_transform_jobs` method, you can specify any or all of the members in the request structure. For more information, see [ListTransformJobs \(p. 448\)](#).

Next Step

[Step 3.5: Validate the Model \(p. 44\)](#)

Step 3.5: Validate the Model

You now have a model that is deployed in Amazon SageMaker. To validate the model, send sample requests and get inferences. To send requests to an Amazon SageMaker endpoint, use the [InvokeEndpoint \(p. 470\)](#) API.

To validate your model, choose one of the following options.

- **Use the high-level Python library provided by Amazon SageMaker.**

The `kmeans_predictor` returned by the `deploy` call in the preceding step provides the `predict` method. To get inferences from the model, call this method.

1. Get an inference for the 30th image of a handwritten number in the `valid_set` dataset.

```
result = kmeans_predictor.predict(valid_set[0][30:31])
print(result)
```

Example response:

```
[label {
    key: "closest_cluster"
    value {
        float32_tensor {
            values: 3.0
        }
    }
}
label {
    key: "distance_to_cluster"
    value {
        float32_tensor {
            values: 7.221197605133057
        }
    }
]
```

The response shows that the input image belongs to cluster 3. It also shows the mean squared distance for that cluster.

Note

In the k-means implementation, the cluster numbers and digit they represent don't align. For example, the algorithm might group images of the handwritten number 3 in cluster 0, and images of the number 4 in cluster 9.

2. Get inferences for the first 100 images.

```
%%time

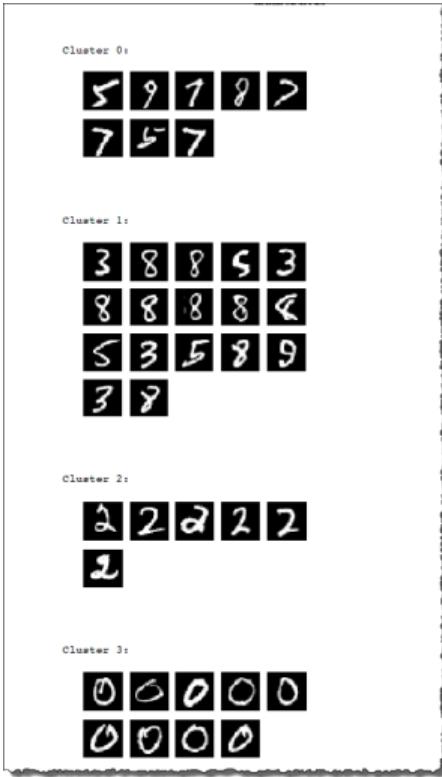
result = kmeans_predictor.predict(valid_set[0][0:100])
clusters = [r.label['closest_cluster'].float32_tensor.values[0] for r in result]
```

Visualize the results.

```
for cluster in range(10):
    print('\n\nCluster {}'.format(int(cluster)))
    digits = [ img for l, img in zip(clusters, valid_set[0]) if int(l) == cluster ]
    height = ((len(digits)-1)//5) + 1
    width = 5
    plt.rcParams["figure.figsize"] = (width,height)
    _, subplots = plt.subplots(height, width)
    subplots = numpy.ndarray.flatten(subplots)
    for subplot, image in zip(subplots, digits):
        show_digit(image, subplot=subplot)
    for subplot in subplots[len(digits):]:
        subplot.axis('off')

plt.show()
```

This code takes the first 100 images of handwritten numbers from the `valid_set` dataset and generates inferences for them. The result is a set of clusters that group similar images. The following visualization shows four of the clusters that the model returned:



- **Use the SDK for Python.**

To send requests to the endpoint, use the `invoke_endpoint` method.

1. Send a request that sends the 30th image in the `train_set` as input. Each image is a 28x28 (total of 784) pixel image. The request sends all 784 pixels in the image as comma-separated values.

```
import json

# Simple function to create a csv from our numpy array
def np2csv(arr):
    csv = io.BytesIO()
    numpy.savetxt(csv, arr, delimiter=',', fmt='%g')
    return csv.getvalue().decode().rstrip()

runtime = boto3.Session().client('sagemaker-runtime')

payload = np2csv(train_set[0][30:31])

response = runtime.invoke_endpoint(EndpointName=endpoint_name,
                                    ContentType='text/csv',
                                    Body=payload)
result = json.loads(response['Body'].read().decode())
print(result)
```

In the following example response, the inference classifies the image as belonging to cluster 7 (`labels` identifies the cluster). The inference also shows the mean squared distance for that cluster.

```
{'predictions': [{}{'distance_to_cluster': 7.2033820152282715, 'closest_cluster': 7.0}]}}
```

- Run another test. The following code takes the first 100 images from the `valid_set` validation set and generates inferences for them. This test identifies the cluster that the input images belong to and provides a visual representation of the result.

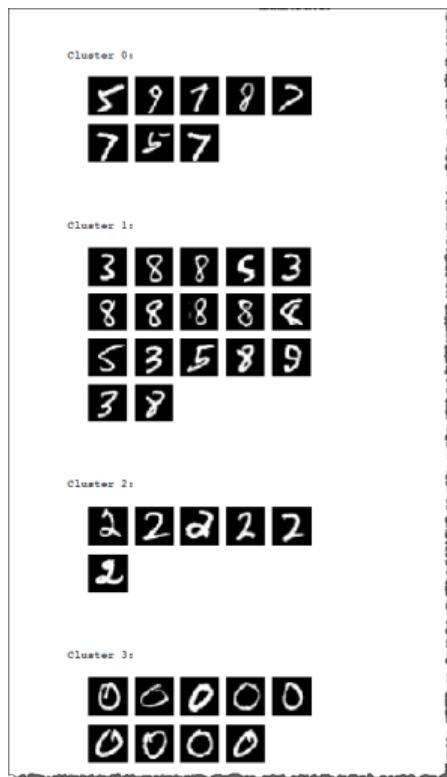
```
%time

payload = np2csv(valid_set[0][0:100])
response = runtime.invoke_endpoint(EndpointName=endpoint_name,
                                    ContentType='text/csv',
                                    Body=payload)
result = json.loads(response['Body'].read().decode())
clusters = [p['closest_cluster'] for p in result['predictions']]

for cluster in range(10):
    print('\n\nCluster {}'.format(int(cluster)))
    digits = [ img for l, img in zip(clusters, valid_set[0]) if int(l) == cluster ]
    height = ((len(digits)-1)//5) + 1
    width = 5
    plt.rcParams["figure.figsize"] = (width,height)
    _, subplots = plt.subplots(height, width)
    subplots = numpy.ndarray.flatten(subplots)
    for subplot, image in zip(subplots, digits):
        show_digit(image, subplot=subplot)
    for subplot in subplots[len(digits):]:
        subplot.axis('off')

plt.show()
```

The result is a set of clusters that group similar images. The following visualization shows four of the clusters that the model returned:



3. To get an idea of how accurate the model is, review the clusters and the numbers in them to see how well the model clustered similar looking digits. To improve the model, you might make the following changes to the training job:
 - Change the model training parameters—for example, increase the number of epochs or tweak hyperparameters, such `extra_center_factor`. For more information, see [K-Means Hyperparameters \(p. 146\)](#).
 - Consider switching the algorithm—The images in the MNIST dataset include information that identifies the digits, called labels. Similarly, you might be able to label your training data for other problems. You might then use the label information and a supervised algorithm, such as the linear learner algorithm provided by Amazon SageMaker. For more information, see [Linear Learner \(p. 164\)](#).
 - Try a more specialized algorithm—Try a specialized algorithm, such as the image classification algorithm provided by Amazon SageMaker instead of the linear learner algorithm. For more information, see [Image Classification Algorithm \(p. 124\)](#).
 - Use a custom algorithm—Consider using a custom neural network algorithm built on Apache MXNet or TensorFlow. For more information, see [Using Apache MXNet with Amazon SageMaker \(p. 277\)](#) and [Using TensorFlow with Amazon SageMaker \(p. 267\)](#).

Next Step

[Step 4: Clean up \(p. 48\)](#)

Step 4: 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. This also deletes the ML compute instance or instances.
 - The endpoint configuration.
 - The model.
 - The notebook instance. You will need to 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 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 5: Additional Considerations: 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](#).

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 \(p. 230\)](#) 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 [Train a Model with a Built-in Algorithm and Deploy It \(p. 30\)](#).

Topics

- [How Hyperparameter Tuning Works \(p. 50\)](#)
- [Defining Objective Metrics \(p. 51\)](#)
- [Defining Hyperparameter Ranges \(p. 52\)](#)
- [Example: Hyperparameter Tuning Job \(p. 53\)](#)
- [Run a Warm Start Hyperparameter Tuning Job \(p. 61\)](#)
- [Design Considerations \(p. 65\)](#)

How Hyperparameter Tuning Works

Hyperparameter tuning is a supervised machine learning regression problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. You can choose any metric that the algorithm you use defines. 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 guesses. 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 it knows about this problem so far. Sometimes it chooses a point that is likely to produce an incremental improvement on the best result found so far. This allows hyperparameter tuning to exploit the best known results. Other times it chooses a set of hyperparameters far removed from those it has tried. This allows it to explore the space 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:

- [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](#)

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 ranges specified are correct.

Defining Objective 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 [Using Built-in Algorithms with Amazon SageMaker \(p. 75\)](#).

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 [Logging Amazon SageMaker with Amazon CloudWatch \(p. 328\)](#).

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`.

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 match metrics that

your algorithm emits. You pass these metrics to the [CreateHyperParameterTuningJob \(p. 355\)](#) operation in the `TrainingJobDefinition` parameter as the `MetricDefinitions` field of the `AlgorithmSpecification` field.

Metric definitions have the following structure:

```
[  
  {  
    "Name": "validation:rmse",  
    "Regex": ".*\\[[0-9]+\\].*#011validation-rmse:(\\S+)"  
  },  
  {  
    "Name": "validation:auc",  
    "Regex": ".*\\[[0-9]+\\].*#011validation-auc:(\\S+)"  
  },  
  {  
    "Name": "train:auc",  
    "Regex": ".*\\[[0-9]+\\]#011train-auc:(\\S+).*"  
  }  
]
```

A regular expression (regex) matches what is in the log, like a search function. Special characters in the regex affect the search. For example, in the regex for the `valid-acc` metric defined above, the parentheses tell the regex to capture what's inside them. We want it to capture the number, which is the metric. The expression `[0-9\\.]+` inside the parentheses tells the regex to look for one or more occurrences of any digit or period `.`. The double backslash is an escape character that means "look for a literal period." (Normally, a regex interprets a period to mean match any single character.)

Choose one of the metrics that you define as the objective metric for the tuning job. If you are using the `Specify` the value of the `name` key in the `HyperParameterTuningJobObjective` field of the `HyperParameterTuningJobConfig` parameter that you send to the [CreateHyperParameterTuningJob \(p. 355\)](#) operation.

Defining 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 [Design Considerations \(p. 65\)](#).

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 \(p. 355\)](#) 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"  
}  
,  
"IntegerParameterRanges": [  
    {  
        "Name": "max_depth",  
        "MaxValue": "10",  
        "MinValue": "1",  
    }  
]  
}
```

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 \(p. 230\)](#) 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 \(p. 24\)](#)
- [An Amazon S3 bucket for storing your training dataset and the model artifacts created during training \(p. 25\)](#)
- [A running Amazon SageMaker notebook instance \(p. 26\)](#)

Topics

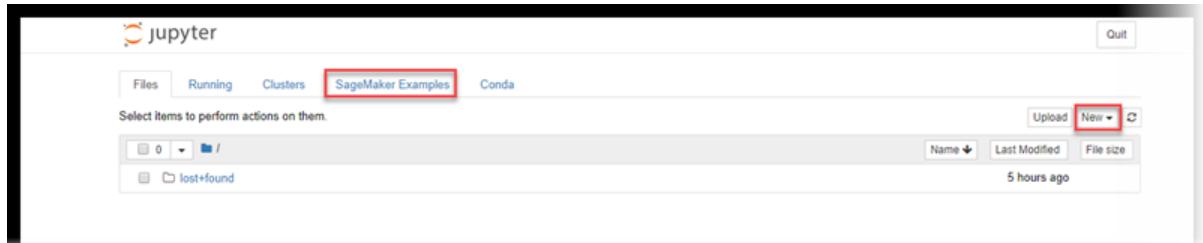
- [Create a Notebook \(p. 53\)](#)
- [Get the sagemaker boto3 Client \(p. 54\)](#)
- [Get the Amazon SageMaker Execution Role \(p. 54\)](#)
- [Specify a Bucket and Data Output Location \(p. 55\)](#)
- [Download, Prepare, and Upload Training Data \(p. 55\)](#)
- [Configure and Launch a Hyperparameter Tuning Job \(p. 56\)](#)
- [Monitor the Progress of a Hyperparameter Tuning Job \(p. 59\)](#)
- [Clean up \(p. 61\)](#)

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 sagemaker boto3 Client \(p. 54\)](#)

Get the sagemaker boto3 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 \(p. 54\)](#)

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 = sagemaker.get_execution_role()
```

```
print(role)
```

Next Step

[Specify a Bucket and Data Output Location \(p. 55\)](#)

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 start with `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 1.2: Create an S3 Bucket \(p. 25\)](#).

`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 \(p. 55\)](#)

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 \(p. 56\)](#)

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 \(p. 56\)](#)
- [Configure the Training Jobs \(p. 57\)](#)
- [Name and Launch the Hyperparameter Tuning Job \(p. 59\)](#)
- [Next Step \(p. 59\)](#)

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 \(p. 355\)](#) 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 [Defining Hyperparameter Ranges \(p. 52\)](#)
- 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.

The hyperparameter tuning job defines ranges for the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters of the [XGBoost Algorithm \(p. 230\)](#) 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 \(p. 355\)](#) call.

In this JSON object, you specify:

- Optional—Metrics that the training jobs emit.

Note

Specify metrics only when you use a custom training algorithm. Because this example uses a built-in algorithm, you don't specify metrics.

- 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 \(p. 230\)](#) 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 \(p. 355\)](#) 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 \(p. 59\)](#)

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 Hyperparameter Tuning Job Status \(p. 59\)](#)
- [View the Status of the Training Jobs \(p. 60\)](#)
- [View the Best Training Job \(p. 60\)](#)

View the Hyperparameter Tuning Job Status

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**.

The screenshot shows the Amazon SageMaker console interface. On the left, there is a navigation sidebar with options like Dashboard, Notebook, Lifecycle configurations, Training (with 'Hyperparameter tuning jobs' highlighted), Inference, Models, Endpoint configurations, and Endpoints. The main content area is titled 'Hyperparameter tuning jobs' and shows a table with the following data:

Name	Status	Training progress	Creation time	Duration
xgboost-tuningjob-03-04-44-33	InProgress	0 / 3 0	Jun 03, 2018 04:44 UTC	4 minutes
xgboost-tuningjob-03-03-06-18	Completed	20 / 20 0	Jun 03, 2018 03:06 UTC	an hour
xgboost-tuningjob-03-02-45-39	Completed	20 / 20 0	Jun 03, 2018 02:45 UTC	an hour
xgboost-tuningjob-30-17-03-34	Completed	20 / 20 0	May 30, 2018 17:03 UTC	an hour
xgboost-tuningjob-29-23-40-59	Failed	0 / 0 0	May 29, 2018 23:41 UTC	5 minutes

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**.

Name	Status	Objective metric value	Creation time	Duration
xgboost-tuningjob-03-04-44-33-003-99bc2095	InProgress	—	Jun 03, 2018 04:45 UTC	—
xgboost-tuningjob-03-04-44-33-002-63d1d0c7	InProgress	—	Jun 03, 2018 04:44 UTC	—
xgboost-tuningjob-03-04-44-33-001-d46f78ce	InProgress	—	Jun 03, 2018 04:44 UTC	—

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**.

The screenshot shows the Amazon SageMaker console interface. At the top, there is a navigation bar with tabs: 'Best training job' (which is highlighted in red), 'Training jobs', 'Job configuration', 'Hyperparameter configuration', and 'Tags'. Below the navigation bar, there is a section titled 'Best training job summary' containing a table with four columns: 'Name' (xgboost-tuningjob-03-04-44-33-003-99bc2095), 'Status' (Completed), 'Objective metric' (validation:auc), and 'Value' (0.772566020488739). To the right of this table is a button labeled 'Create model' with a red border. Below the summary is another section titled 'Best training job hyperparameters' with a table showing five hyperparameters: '_tuning_objective_metric' (Static, validation:auc), 'alpha' (Continuous, 1.9818243759579417), 'eta' (Continuous, 0.07404334782758304), and 'eval_metric' (Static, auc). There is also a search bar above the table.

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 \(p. 61\)](#)

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/.

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 Bayesian search to choose combinations

of hyperparameter values from ranges that you specify. For more information, see [How Hyperparameter Tuning Works \(p. 50\)](#). 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 \(p. 62\)](#)
- [Warm Start Tuning Restrictions \(p. 63\)](#)
- [Warm Start Tuning Sample Notebook \(p. 63\)](#)
- [Create a Warm Start Tuning Job \(p. 63\)](#)

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 \(p. 395\)](#) named `OverallBestTrainingJob`. The value of this field is the [TrainingJobSummary \(p. 538\)](#) 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 [Using Example Notebooks \(p. 72\)](#). 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 by Using Boto 3 \(p. 64\)](#)
- [Create a Warm Start Tuning Job by Using the Amazon SageMaker Python SDK \(p. 64\)](#)

Create a Warm Start Tuning Job by Using Boto 3

To use warm start tuning, you specify the values of a [HyperParameterTuningJobWarmStartConfig \(p. 507\)](#) object, and pass that as the `WarmStartConfig` field in a call to [CreateHyperParameterTuningJob \(p. 355\)](#).

The following code shows how to create a [HyperParameterTuningJobWarmStartConfig \(p. 507\)](#) object and pass it to [CreateHyperParameterTuningJob \(p. 355\)](#) 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 by Using the 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 \(p. 124\)](#) 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 [Defining Hyperparameter Ranges \(p. 52\)](#).

```
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)
```

Design Considerations

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 \(p. 66\)](#)
- [Choosing Hyperparameter Ranges \(p. 66\)](#)
- [Use Logarithmic Scales for Hyperparameters \(p. 66\)](#)
- [Choosing the Best Degree of Parallelism \(p. 66\)](#)
- [Running Training Jobs on Multiple Instances \(p. 66\)](#)

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 ranges 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, you will get better results by limiting your search to a small range where all possible values in the range are reasonable. If you get the best metric values within a part of a range, consider limiting the range to that part.

Use 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 Degree of Parallelism

Running more hyperparameter tuning jobs in parallel 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. Design distributed training jobs so that you get the metric report you want.

Using 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, and to write code to train models, deploy models to Amazon SageMaker hosting, and to test or validate your models.

Topics

- [Creating a Notebook Instance \(p. 67\)](#)
- [Accessing Notebook Instances \(p. 68\)](#)
- [Using Example Notebooks \(p. 72\)](#)
- [Set the Notebook Kernel \(p. 73\)](#)
- [Installing External Libraries and Kernels in Notebook Instances \(p. 73\)](#)

Creating a Notebook Instance

To create a notebook instance, use either the Amazon SageMaker console or the [CreateNotebookInstance \(p. 362\)](#) API. For an example of using the Amazon SageMaker console to create a notebook instance, see [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 26\)](#).

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 [Notebook Instance Security \(p. 333\)](#).
- **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.

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. For more information, see [Amazon EC2 Instance Store](#) in the *Amazon Elastic Compute Cloud User Guide*.

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

Accessing Notebook Instances

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

- Use the console.

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, choose the **Open** action for the instance.

The screenshot shows the Amazon SageMaker console interface. On the left is a navigation sidebar with links for Dashboard, Notebook instances (which is highlighted in orange), Jobs, Resources, Models, Endpoint configuration, and Endpoints. The main content area is titled 'Notebook instances' and shows a table with a single row. The row contains the following data: Name (ExampleNotebookInstance2), Instance (ml.t2.medium), Creation time (Nov 21, 2017 22:19 UTC), Status (InService), and Actions (with a red box around the 'Open' button). Above the table are buttons for Open, Start, Update settings, Actions, and Create notebook instance.

The console uses your sign-in credentials to send a [CreatePresignedNotebookInstanceUrl \(p. 369\)](#) 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.

- Use the API.

To get the URL for the notebook instance, call the [CreatePresignedNotebookInstanceUrl \(p. 369\)](#) 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>.

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 \(p. 369\)](#) 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](#). 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 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": "Deny",  
        "Action": "sagemaker>CreatePresignedNotebookInstanceUrl",  
        "Resource": "*",  
        "Condition": {  
            "NotIpAddress": {  
                "aws:SourceIP": "192.0.2.0/24",  
                "aws:SourceIP": "203.0.113.0/24"  
            }  
        }  
    }  
}
```

```
"Condition": {
    "NotIpAddress": {
        "aws:SourceIp": [
            "192.0.2.0/24",
            "203.0.113.0/24"
        ]
    }
}
"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.

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 \(p. 69\)](#).

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 so that when users call [CreatePresignedNotebookInstanceIdUrl](#) (p. 369) 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. 334).

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](#).

When you create the interface endpoint, specify `aws.sagemaker.region.notebook` as the service name. We recommend that you enable private DNS hostnames for your VPC endpoint. If you don't enable private DNS hostnames, users that connect to the notebook instance through the console will not be connecting through the interface endpoint (that is, the console will attempt to connect over the internet).

After you have created a VPC endpoint, users can use it to connect to your notebook instance from within your VPC. If you enable private DNS hostnames for your VPC endpoint, users do not need to specify the VPC endpoint when connecting to the notebook instance. 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 VPC endpoint.

If you do not enable private DNS hostnames for your VPC endpoint, users need to specify the VPC endpoint name when connecting to the notebook instance.

Amazon SageMaker notebook instances support VPC endpoints in all AWS Regions where both [Amazon VPC](#) and [Amazon SageMaker](#) are available.

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 Direct Connect. For information about VPN, see [VPN Connections in the Amazon Virtual Private Cloud User Guide](#). For information about Direct Connect, see [Creating a Connection](#) in the [AWS Direct Connect User Guide](#).

Restricting 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 will not be able to 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, and then add that policy to every AWS Identity and Access Management user, group, or role used to access the notebook instance.

```
{  
    "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:SourceVpc` 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": {  
                "StringEquals": {  
                    "aws:SourceVpc": "vpc-111bbaaa"  
                }  
            }  
        }  
    ]  
}
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "Enable Notebook Access",
        "Effect": "Allow",
        "Action": [
            "sagemaker>CreatePresignedNotebookInstanceUrl",
            "sagemaker:DescribeNotebookInstance"
        ],
        "Resource": "*",
        "Condition": {
            "ForAllValues:StringEquals": {
                "aws:sourceVpce": [
                    "vpce-11bbccc",
                    "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 \(p. 334\)](#). In the second example, one of the values for `aws:SourceVpce` is the ID of the interface endpoint for the notebook instance, and the other is the ID of the interface endpoint for the Amazon SageMaker API.

The policy examples here include [DescribeNotebookInstance \(p. 403\)](#) because typically you would call `DescribeNotebookInstance` to make sure that the `NotebookInstanceState` 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",
    "NotebookInstanceState": "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

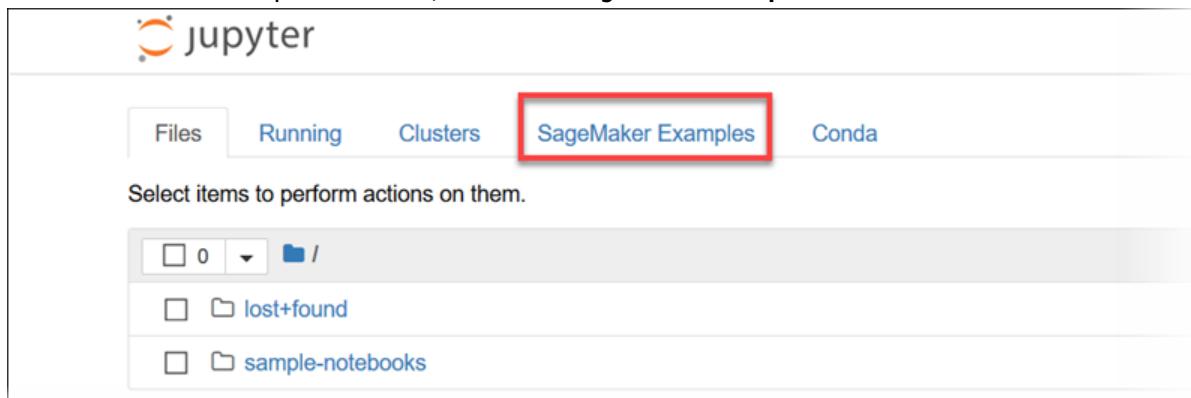
{
    "AuthorizedUrl": "https://mynotebookinstance.notebook.us-west-2.sagemaker.aws?",
    authToken=AuthToken
}
```

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 need to specify the endpoint URL in the call. For example, the call to `create-presigned-notebook-instance-url` would be:

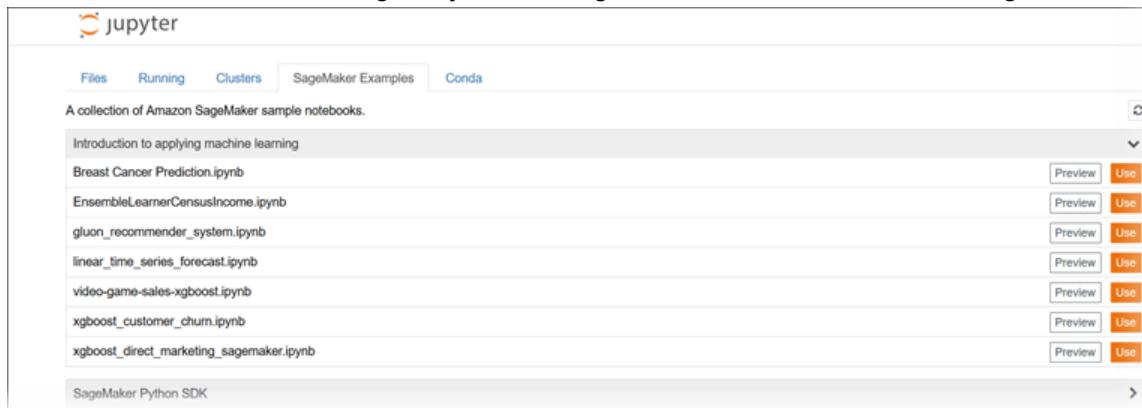
```
aws sagemaker create-preserved-notebook-instance-url  
--notebook-instance-name myNotebookInstance --endpoint-url  
VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
```

Using 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>. To view or use the example notebooks, choose the **SageMaker Examples** tab.



To view a read-only version of an example notebook, 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.



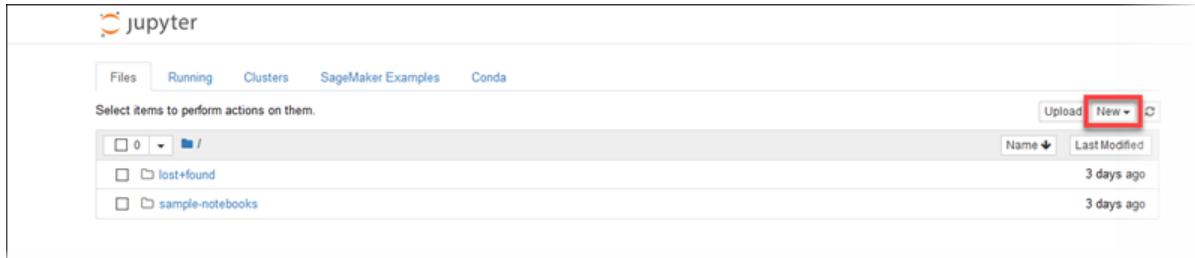
For more information about the example notebooks, see the [Amazon SageMaker examples GitHub repository](#).

Note

Example notebooks typically download datasets from the internet. If you disable Amazon SageMaker-provided internet access when you create your notebook instance, example notebooks might not work. For more information, see [Notebook Instances Are Internet-Enabled by Default \(p. 333\)](#).

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.



Installing 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`.

For example, to install the R kernel:

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

```
conda install --yes --name JupyterSystemEnv --channel r r-essentials=1.6.0
```

4. When the command completes, exit from the terminal, and refresh the Jupyter dashboard page.
5. In the Jupyter dashboard page, choose **New**. **R** now appears as a **Notebook** option.

Notebook instances come with git installed. You can use git to install projects from GitHub.

For example, to install the Scala kernel:

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

```
git clone https://github.com/jupyter-scala/jupyter-scala.git
cd jupyter-scala
./jupyter-scala
```

4. When the command completes, exit from the terminal, and refresh the Jupyter dashboard page.
5. In the Jupyter dashboard page, choose **New**. **Scala** now appears as a **Notebook** option.

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
```

Using Built-in Algorithms with Amazon SageMaker

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 \(p. 164\)](#) and the [XGBoost Algorithm \(p. 230\)](#). 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 \(p. 164\)](#) or the [XGBoost Algorithm \(p. 230\)](#) 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 \(p. 114\)](#) 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 \(p. 142\)](#).
- "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 \(PCA\) \(p. 206\)](#) algorithm.

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

- [Image Classification Algorithm \(p. 124\)](#)—Use this algorithm to classify images. It uses example data with answers (referred to as *supervised algorithm*).
- [Sequence to Sequence \(p. 218\)](#)—This supervised algorithm is commonly used for neural machine translation.
- [Latent Dirichlet Allocation \(LDA\) \(p. 159\)](#)—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 \(NTM\) \(p. 178\)](#)—Another unsupervised technique for determining topics in a set of documents, using a neural network approach.

Topics

- [Algorithms Provided by Amazon SageMaker: Common Information \(p. 77\)](#)
- [BlazingText \(p. 91\)](#)
- [DeepAR Forecasting \(p. 100\)](#)
- [Factorization Machines \(p. 114\)](#)
- [Image Classification Algorithm \(p. 124\)](#)
- [IP Insights Algorithm \(p. 133\)](#)
- [K-Means Algorithm \(p. 142\)](#)

- [K-Nearest Neighbors \(p. 150\)](#)
- [Latent Dirichlet Allocation \(LDA\) \(p. 159\)](#)
- [Linear Learner \(p. 164\)](#)
- [Neural Topic Model \(NTM\) \(p. 178\)](#)
- [Object2Vec \(p. 184\)](#)
- [Object Detection Algorithm \(p. 196\)](#)
- [Principal Component Analysis \(PCA\) \(p. 206\)](#)
- [Random Cut Forest \(p. 210\)](#)
- [Sequence to Sequence \(p. 218\)](#)
- [XGBoost Algorithm \(p. 230\)](#)

Algorithms Provided by Amazon SageMaker: Common Information

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

Topics

- [Algorithms Provided by Amazon SageMaker: Common Parameters \(p. 77\)](#)
- [Algorithms Provided by Amazon SageMaker: Common Data Formats \(p. 82\)](#)
- [Algorithms Provided by Amazon SageMaker: Suggested Instance Types \(p. 90\)](#)
- [Algorithms Provided by Amazon SageMaker: Logs \(p. 90\)](#)

Algorithms Provided by Amazon SageMaker: Common Parameters

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
BlazingText	train	<ecr_path>/blazingtext:<tag>	File	Text file (one sentence per line with space-separated tokens)	GPU (single instance only) or CPU
DeepAR Forecasting	train and (optionally) test	<ecr_path>/forecasting-deepar:<tag>	File	JSON Lines or Parquet	GPU or CPU
Factorization Machines	train and (optionally) test	<ecr_path>/factorization-machines:<tag>	File or Pipe	recordIO-protobuf	CPU (GPU for dense data)

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class
Image Classification	train and validation, (optionally) train_lst and validation_lst	<ecr_path>/image-classification:<tag>	File	recordIO or image files (.jpg or .png)	GPU
IP Insights	train and (optionally) validation	<ecr_path>/ipinsights:<tag>	File	CSV	GPU or GPU
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)
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)
LDA	train and (optionally) test	<ecr_path>/lda:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU (single instance only)
Linear Learner	train and (optionally) validation, test, or both	<ecr_path>/linear-learner:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU or GPU
Neural Topic Model	train and (optionally) validation, test, or both	<ecr_path>/ntm:<tag>	File or Pipe	recordIO-protobuf or CSV	GPU or CPU
Object2Vec	train and (optionally) validation, test, or both	<ecr_path>/object2vec:<tag>	File	JSON Lines	GPU or CPU (single instance only)

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class
Object Detection	train and validation, (optionally) train_annotation and validation_annotation	<ecr_path>/object-detection:<tag>	File	recordIO or image files (.jpg or .png)	GPU
PCA	train and (optionally) test	<ecr_path>/pca:<tag>	File or Pipe	recordIO-protobuf or CSV	GPU or CPU
Random Cut Forest	train and (optionally) test	<ecr_path>/randomcutforest:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU
Seq2Seq Modeling	train, validation, and vocab	<ecr_path>/seq2seq:<tag>	File	recordIO-protobuf	GPU (single instance only)
XGBoost	train and (optionally) validation	<ecr_path>/xgboost:<tag>	File	CSV or LibSVM	CPU

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-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

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
	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-west-1	438346466558.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.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
	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, Seq2Seq, and XGBoost	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-northeast-1	501404015308.dkr.ecr.ap-northeast-1.amazonaws.com

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
DeepAR Forecasting	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-west-1	685385470294.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.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-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-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

Use the paths and training input mode as follows:

- To create a training job (with a request to the [CreateTrainingJob \(p. 371\)](#) 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 \(p. 359\)](#) 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).

Algorithms Provided by Amazon SageMaker: Common Data Formats

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

Topics

- [Common Data Formats—Training \(p. 82\)](#)
- [Common Data Formats—Inference \(p. 85\)](#)

Common Data Formats—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.

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 \(p. 477\)](#) 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 [Step 3.2.3: Transform the Training Dataset and Upload It to S3 \(p. 33\)](#).

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 SageMaker's 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-dimensonial 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-dimensonial 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-dimensonial 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 be '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—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)

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 ]  
        }  
    ]  
}
```

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

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

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

Content-type: text/csv;label_size=0

Note

CSV support is not available for factorization machines.

RECORDIO

Content-type: application/x-recordio-protobuf

Using 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 \(p. 209\)](#)
- [Linear Learner Response Formats \(p. 176\)](#)
- [NTM Response Formats \(p. 183\)](#)
- [k-means Response Formats \(p. 149\)](#)
- [Factorization Machine Response Formats \(p. 122\)](#)

Algorithms Provided by Amazon SageMaker: Suggested Instance Types

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](#).

Algorithms Provided by Amazon SageMaker: Logs

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

Example 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'}]}\n
```

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

The Amazon SageMaker BlazingText algorithm provides highly optimized implementations of the Word2vec and text classification algorithms. Word2vec is useful for many downstream natural language processing (NLP) tasks such as sentiment analysis, named entity recognition, machine translation etc. whereas text classification is an important task for applications like web search, 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. Word embeddings thus 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 subsequently 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. But with BlazingText, you can scale to large datasets easily. Similar to Word2vec, it provides the Skip-gram and continuous bag-of-words (CBOW) training architectures.

The implementation of supervised multi class/label text classification algorithm by BlazingText extends the fastText text classifier to leverage GPU acceleration using custom [CUDA](#) kernels .The model can be trained on more than a billion words in a couple of minutes using a multi-core CPU or a GPU, while achieving performance on par with the state-of-the-art deep learning text classification algorithms.

Amazon SageMaker BlazingText provides the following features:

- Accelerated training of 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 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 \(p. 92\)](#)
- [EC2 Instance Recommendation \(p. 94\)](#)
- [BlazingText Sample Notebooks \(p. 95\)](#)
- [BlazingText Hyperparameters \(p. 95\)](#)
- [Tuning a BlazingText Model \(p. 99\)](#)

Input/Output Interface

BlazingText expects a single preprocessed text file with space separated tokens and each line of 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

Word2Vec algorithm (`skipgram`, `cbow`, and `batch_skipgram` modes)

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

Text Classification algorithm (supervised 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 shortlived .
```

Note

The order of labels within the sentence does not matter.

Upload the training file under train channel and (optionally) the validation file under the validation channel.

Model artifacts and Inference

Word2Vec algorithm (skipgram, cbow, and batch_skipgram modes)

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. `vectors.txt` stores the vectors in a format that is compatible with other tools like Gensim and Spacy. For example, a Gensim user is able to execute 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 evaluation parameter is set `True`, then an additional file `eval.json` is created, which 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 are not 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 vectors 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"]
}
```

Text Classification algorithm (supervised mode)

Training with supervised will output 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.

Sample JSON request

Mime-type: application/json

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

By default, the server will return only one prediction, the one with the highest probability. For retrieving the top k predictions, you can set k in configuration as shown below:

```
{  
  "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 is as 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 is as 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. Binaries produced by BlazingText can be used by fastText and the model binaries created with fastText can be hosted using BlazingText.

For more details on dataset formats and model hosting, see the example notebooks [here](#), [here](#), and [here](#).

EC2 Instance Recommendation

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 on accuracy, we recommend using a 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 S3DataSource field of the [S3DataSource \(p. 530\)](#)

object that you pass to [CreateTrainingJob \(p. 371\)](#) 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 2GB. 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 Blazing Text algorithm to train and deploy supervised binary and multiclass classification models, see [Blazing Text classification on the DBPedia dataset](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Using Notebook Instances \(p. 67\)](#). 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**.

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. As BlazingText can be used in two different modes, *Word2Vec (unsupervised)* and *Text Classification (supervised)*, the hyperparameters are itemized in separate sections.

Word2Vec Hyperparameters

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

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

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

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

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
<code>mode</code>	<p>The training mode.</p> <p>Required</p> <p>Valid values: <code>supervised</code></p>
<code>buckets</code>	<p>The number of hash buckets to use for word n-grams.</p> <p>Optional</p> <p>Valid values: positive integer</p>

Parameter Name	Description
	Default value: 2000000
early_stopping	<p>Whether to stop training if validation accuracy doesn't improve after a patience number of epochs.</p> <p>Optional</p> <p>Valid values: (Boolean) <code>True</code> or <code>False</code></p> <p>Default value: <code>False</code></p>
epochs	<p>The maximum number of complete passes through the training data.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
learning_rate	<p>The step size used for parameter updates.</p> <p>Optional</p> <p>Valid values: Positive float</p> <p>Default value: 0.05</p>
min_count	<p>Words that appear less than <code>min_count</code> times are discarded.</p> <p>Optional</p> <p>Valid values: Non-negative integer</p> <p>Default value: 5</p>
min_epochs	<p>The minimum number of epochs to train before early stopping logic is invoked.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
patience	<p>The number of epochs to wait before early stop if no progress is made on the validation set. Used only when <code>early_stopping</code> is <code>True</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 4</p>

Parameter Name	Description
<code>vector_dim</code>	<p>The imension of the embedding layer.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 100</p>
<code>word_ngrams</code>	<p>The number of word n-grams features to use.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 2</p>

Tuning 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 \(p. 50\)](#).

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.

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 this metrics as the objective.

Metric Name	Description	Optimization Direction
<code>train:mean_rho</code>	Mean rho (Spearman's rank correlation coefficient) on WS-353 word similarity datasets .	Maximize
<code>validation:accuracy</code>	Classification accuracy on user specified validation dataset.	Maximize

Tunable BlazingText Hyperparameters

Tunable Hyperparameters for Word2Vec

Tune the 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
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 Text Classification

Tune the Amazon SageMaker BlazingText Text Classification model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
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

Amazon SageMaker DeepAR is a supervised learning algorithm for forecasting scalar (that is, 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, and then use that model to extrapolate the time series into the future. In many applications, however, you encounter many similar time series across a set of cross-sectional units. Examples of such time series groupings are demand for different products, server loads, and requests for web pages. In this case, it can be beneficial to train a single model jointly over all of these time series. DeepAR takes this approach, outperforming the standard ARIMA and ETS methods when your dataset contains hundreds of related time series. The trained model can also be used for generating forecasts for new time series that are similar to the ones it has been trained on.

For the training phase, the dataset consists of one or preferably more than one time series (`target`). Target time series can be optionally associated with a vector of categorical features (`cat`) and a vector of feature time series (`dynamic_feat`). The DeepAR model is trained by randomly sampling training examples from each target time series in the training dataset. Each such 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 `prediction_length` how far in the future predictions can be made. For a more detailed explanation, check the [How DeepAR Works \(p. 104\)](#) section.

Topics

- [Input/Output Interface \(p. 101\)](#)
- [Recommended Best Practices \(p. 103\)](#)
- [EC2 Instance Recommendations \(p. 103\)](#)
- [DeepAR Sample Notebooks \(p. 104\)](#)
- [How DeepAR Works \(p. 104\)](#)
- [DeepAR Hyperparameters \(p. 106\)](#)
- [Tuning a DeepAR Model \(p. 110\)](#)
- [DeepAR Inference Formats \(p. 112\)](#)

Input/Output Interface

DeepAR supports two data channels. The `train` channel describes the training dataset and is required. The `test` channel describes a dataset which is used to evaluate a set of accuracy metrics for the model after training and is optional. You can provide training and test datasets as [JSON Lines](#), which can be zipped, or [Parquet](#) files.

The training and the test paths can point to a single file or a directory containing multiple files (possibly nested in subdirectories). If the path is a directory, DeepAR considers all files in the directory, except files starting with a dot and files named `_SUCCESS`, as inputs for the corresponding channel. This convention 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 (either `.json` or `.json.gz` or `.parquet`) of the specified input path. If the path does not end in one of these suffixes, you must specify the format explicitly. In the Python SDK, this is achieved using the `content_type` parameter of the [`s3_input`](#) class.

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

- `start`: a string of the format `YYYY-MM-DD HH:MM:SS`. The start time stamp should not contain any time zone information.
- `target`: an array of floats (or integers) that represent the time series. Missing values can be encoded as `null` literals or `"NaN"` strings (in JSON), or as `nan` float values (in Parquet).
- `dynamic_feat` (optional): an array of arrays of floats (or integers) that represents the vector of custom feature time series (dynamic features). If the field is set, all records must have the same number of inner arrays (that is, 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.
- `cat` (optional): an array of categorical features that can be used to encode groups to which the record belongs. 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}`. Clients must ensure that all values from each categorical domain are represented in the training dataset. This restriction is necessary because during inference we can only forecast for categories which have been observed

during training. Further, each categorical feature is embedded in a low dimensional space whose dimensionality is controlled by the `embedding_dimension` hyperparameter (see [DeepAR Hyperparameters \(p. 106\)](#)).

If you use a JSON file, it must be in the [JSON Lines](#) format. The following is an example of JSON data:

```
{"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 categorical features and one time series features associated.

For Parquet, you use the same three fields as columns. In addition, "start" can be the `datetime` type. Parquet files can be compressed using `gzip` and `snappy` compression.

For training data:

- Different time series may differ in their start time and their length but all series must have the same frequency, the same number of categorical features and the same number of dynamic features.
- The training file should be shuffled with respect to the position of the time series in the file. In other words, the time series should occur in a random order in the file.
- The `start` time stamp is used for deriving the internal features. So it is important that the `start` field is set correctly.
- If you use categorical features (`cat`), all time series must have the same number of categorical features. If the dataset contains the `cat` field, it is used and the cardinality of the groups is extracted from the dataset (`cardinality = "auto"` is default). If the dataset contains the `cat` field, but you do not want to use, you can disable the use of the categorical feature by setting (`cardinality = ""`). If a model was trained using a `cat` feature, this feature has to be provided for prediction.
- If your dataset contains the field `dynamic_feat`, it is used 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 and entry in `dynamic_feat` should have the same length as the `target`. If the dataset contains the `dynamic_feat` field, but you do not want to use it, you can disable the use by setting (`num_dynamic_feat = "`). If the model was trained with the `dynamic_feat` field this field has to be provided for inference and each of the features has to have the length of the provided `target + prediction_length`. (i.e., the feature value in the future has to be provided).

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:

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

where $y_{i,t}$ is the true value of time series i at time t and $\hat{y}_{i,t}$ 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 (see [DeepAR Hyperparameters \(p. 106\)](#)).

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

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

Here, $q_{i,t}^{(\tau)}$ is the τ -quantile of the distribution that the model predicts. Set the `test_quantiles` hyperparameter to specify which quantiles for which the algorithm calculates quantile loss. In addition to these, the average of the prescribed quantile losses is reported as part of the training logs. For information, see [DeepAR Hyperparameters \(p. 106\)](#).

For inference, DeepAR accepts JSON format with an "instances" field which includes one or more time series in JSON Lines format, and a name of "configuration", which includes parameters for generating the forecast. For details, see [DeepAR Inference Formats \(p. 112\)](#).

Recommended Best Practices

To achieve optimal result, the following points should be considered when preparing the time series data:

- Except for the purpose of the train/test split discussed below, always provide entire time series for training, testing, and when calling the model for prediction. Do not cut the time series up or provide only a part of the time series irrespective of how you set `context_length`. The model uses data points further back than `context_length` for the lagged values feature.
- For [Tuning a DeepAR Model \(p. 110\)](#), you can split the dataset into a training and a test dataset. In a typical evaluation scenario, you would like to test the model on the same time series used in the training, but on the future `prediction_length` time points following immediately after the last time point visible during training. A simple way to create train and test sets satisfying these criteria is to use the entire dataset (that is, the full length of all time series available) as a test set and remove the last `prediction_length` points from each time series for training. This way, during training, the model does not see the target values for time points on which it is evaluated during testing. In the test phase, the last `prediction_length` points of each time series in the test set are withheld and a prediction is generated. The forecast is then compared 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 end points. This results in accuracy metrics averaged over multiple forecasts from different time points.
- Try not to use very large values (> 400) for the `prediction_length`, since this makes the model slow and less accurate. If you want to forecast further into the future, consider aggregating to a higher frequency. For example, use `5min` instead of `1min`.
- Due to the use of lags, a model can look further back in the time series than `context_length` and so it is not necessary to set this parameter to a large value. A good starting value for this parameter is to use the same value as used for the `prediction_length`.
- We recommend training DeepAR model on as many time series as available. While a DeepAR model trained on a single time series may already work well, standard forecasting methods such as ARIMA or ETS may be more accurate and are more tailored to this use case. The DeepAR approach really starts to outperform the standard methods when your dataset contains hundreds of related time series.

EC2 Instance Recommendations

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

For inference, DeepAR only supports CPU instances.

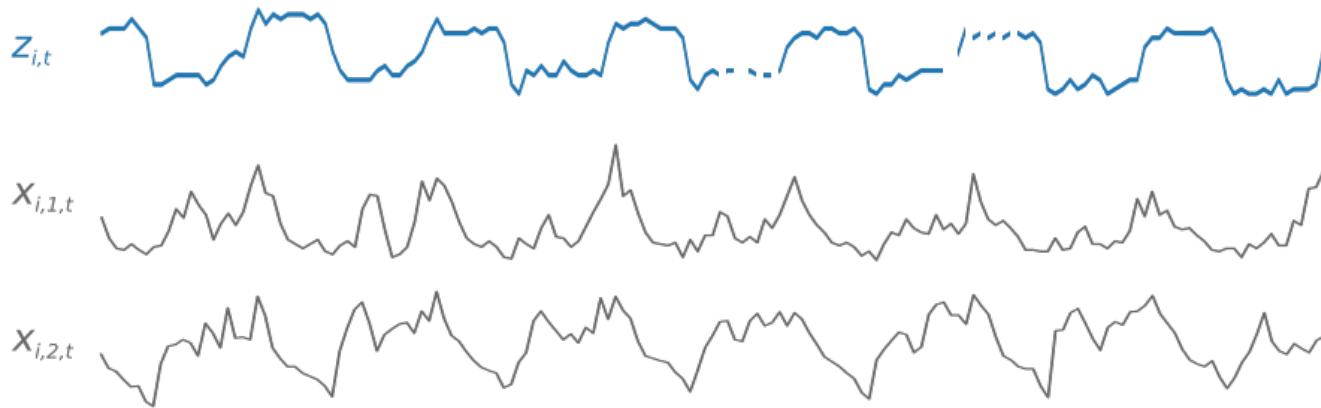
Large values of `context_length`, `prediction_length`, `num_cells`, `num_layers`, `mini_batch_size`, can lead to large model sizes that may not fit into small instances. In this case, use a larger machine or reduce the values for these parameters. This problem also frequently occurs when running hyperparameter 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](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Using Notebook Instances \(p. 67\)](#). 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 DeepAR Works

During training, DeepAR accepts a training dataset and an optional test dataset, which is used to evaluate the final trained model. In general, these two datasets do not have to contain the same set of time series. A model trained on a given training set can be used to generate forecasts not only for the future of the time series in the training set, but also for other time series. Both the training and the test datasets consist of (preferably more than one) *target time series*, optionally associated with a vector of *feature time series* and a vector of *categorical features*. See [Input/Output Interface \(p. 101\)](#) for details. As an illustration, consider the following example for 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 $X_{i,2,t}$.

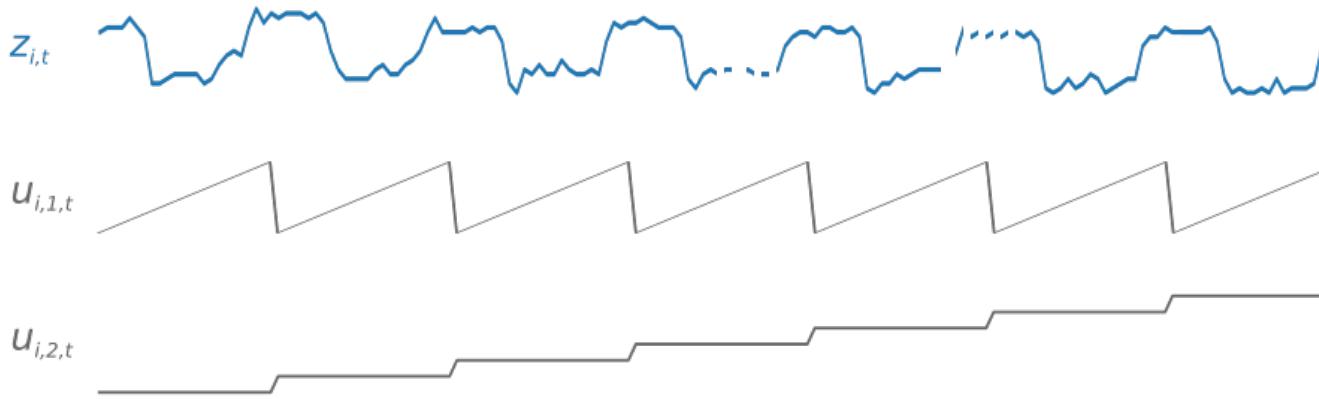


The target time series may contain missing values, represented by line breaks in the time series. DeepAR only supports feature time series that are known in the future. This allows one to run counterfactual "what-if" scenarios. What happens, for example, if I change the price of a product in some way? Each target time series can also be associated with a number of categorical features. These can be used to encode that a time series belongs to certain groupings. Categorical features allow the model to learn typical behavior for such groups, which can be used to increase the overall 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.

Under the Hood

In order to facilitate the learning of 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,

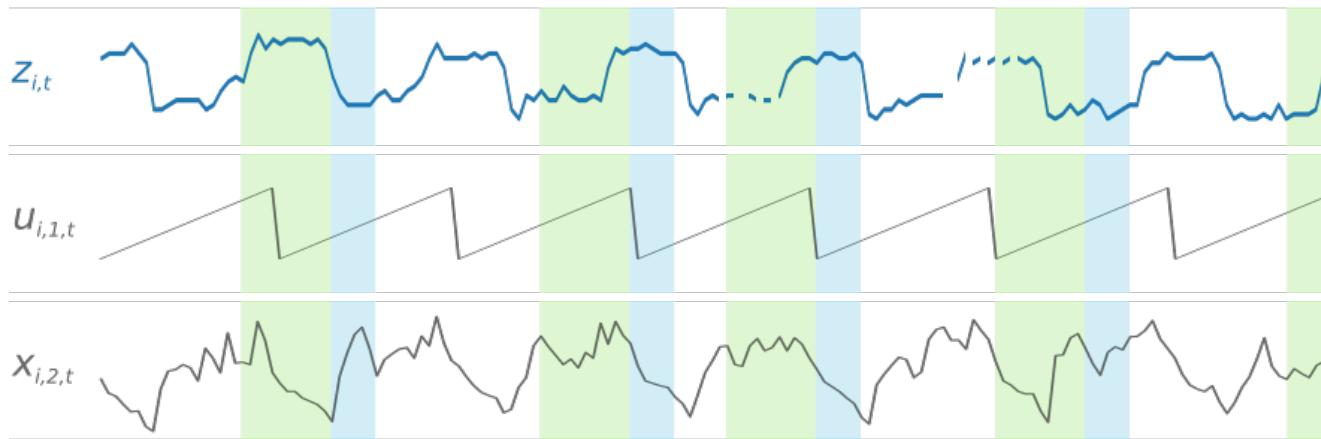
DeepAR creates two feature time series (day of the month and day of the year) for a weekly time series frequency. These derived feature time series are used along with the custom feature time series provided by the user during training and inference. The following figure shows two of these derived time series features: $u_{i,1,t}$ represents the hour of the day, and $u_{i,2,t}$ the day of the week.



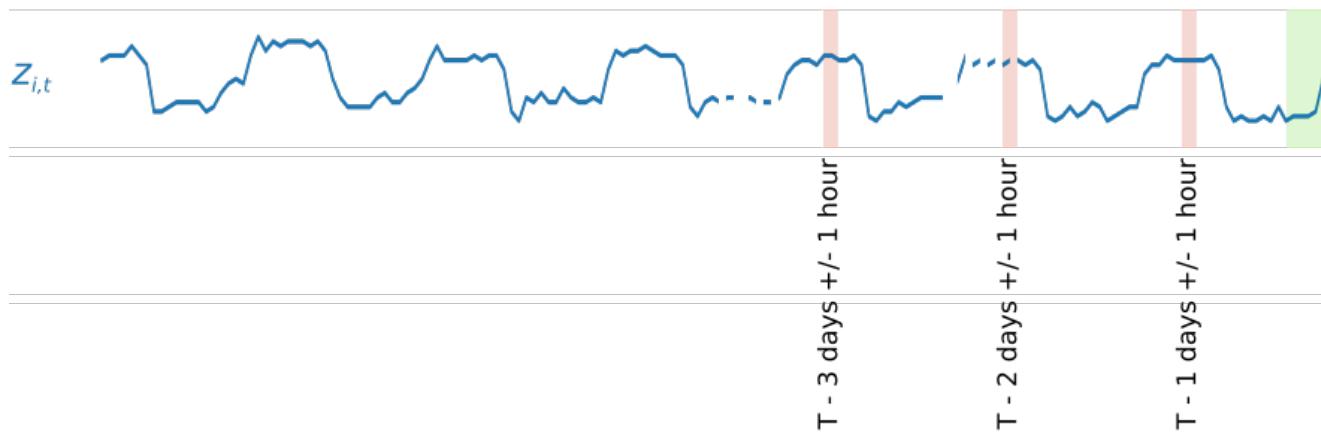
These feature time series are automatically generated and don't have to be provided by the user. Here is the list of derived features for the supported basic time frequencies.

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

The DeepAR model is trained by randomly sampling several training examples from each of the time series in the training dataset. Each such 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 `prediction_length` controls how far in the future predictions can be made. Training set elements containing time series that are shorter than a specified prediction length are ignored during training. The following figure represents five samples with context lengths of 12 hours and prediction lengths of 6 hours drawn from element i . The feature time series $x_{i,1,t}$ and $u_{i,2,t}$ have been omitted for brevity.



To capture seasonality patterns, DeepAR also automatically feeds lagged values from the target time series. In our running 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
<code>context_length</code>	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 <code>prediction_length</code> . The model also receives lagged inputs from the target, so <code>context_length</code> 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

Parameter Name	Description
	<p>time series. For example, lag values for daily frequency are previous week, 2 weeks, 3 weeks, 4 weeks, and year.</p> <p>Required</p> <p>Valid values: positive integer</p>
epochs	<p>The maximum number of passes over the training data. The optimal value depends on your data size and learning rate. See also <code>early_stopping_patience</code>. Typical values range from 10 to 1000.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>prediction_length</code>	<p>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 <code>prediction_length</code> is fixed when a model is trained and it cannot be changed later.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>time_freq</code>	<p>The granularity of the time series in the dataset. Use <code>time_freq</code> to select appropriate date features and lags. The model supports the following basic frequencies. It also supports multiples of these basic frequencies. For example, <code>5min</code> specifies a frequency of 5 minutes.</p> <ul style="list-style-type: none"> • <code>M</code>: monthly • <code>W</code>: weekly • <code>D</code>: daily • <code>H</code>: hourly • <code>min</code>: every minute <p>Required</p> <p>Valid values: An integer followed by <code>M</code>, <code>W</code>, <code>D</code>, <code>H</code>, or <code>min</code>. For example, <code>5min</code></p>

Parameter Name	Description
cardinality	<p>When using the categorical features (<code>cat</code>), <code>cardinality</code> is an array specifying the number of categories (groups) per categorical feature. Set this to <code>auto</code> to use the <code>cat</code> feature if it is present in the data and extract the cardinalities automatically from the dataset. Set it to an empty string to disable use of the <code>cat</code> field even if it is present in the data. If you include categorical features, <code>cardinality</code> specifies the number of categories (groups).</p> <p>For a fixed array index i with <code>cardinality[i] = N</code>, for all time series the value of <code>cat[i]</code> must range between 0 and $N-1$.</p> <p>Optional</p> <p>Valid values: array of positive integers, empty string, or <code>auto</code></p> <p>Default value: <code>auto</code></p>
dropout_rate	<p>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.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0.1</p>
early_stopping_patience	<p>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.</p> <p>Optional</p> <p>Valid values: integer</p>
embedding_dimension	<p>Size of embedding vector learned per categorical feature (same value is used for all categorical features).</p> <p>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 <code>embedding_dimension</code> for each group, capturing the common properties of all time series in the group. A larger <code>embedding_dimension</code> allows the model to capture more complex patterns. However, because increasing the <code>embedding_dimension</code> 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.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10</p>

Parameter Name	Description
<code>learning_rate</code>	<p>The learning rate used in training. Typical values range from 1e-4 to 1e-1.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1e-3</p>
<code>likelihood</code>	<p>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:</p> <ul style="list-style-type: none"> • <i>gaussian</i>: Use for real-valued data. • <i>beta</i>: Use for real-valued targets between 0 and 1 inclusive. • <i>negative-binomial</i>: Use for count data (non-negative integers). • <i>student-T</i>: An alternative for real-valued data that works well for bursty data. • <i>deterministic-L1</i>: A loss function that does not estimate uncertainty and only learns a point forecast. <p>Optional</p> <p>Valid values: One of <i>gaussian</i>, <i>beta</i>, <i>negative-binomial</i>, <i>student-T</i>, or <i>deterministic-L1</i>.</p> <p>Default value: <i>student-T</i></p>
<code>mini_batch_size</code>	<p>The size of mini-batches used during training. Typical values range from 32 to 512.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 128</p>
<code>num_cells</code>	<p>The number of cells to use in each hidden layer of the RNN. Typical values range from 30 to 100.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 40</p>

Parameter Name	Description
<code>num_dynamic_feat</code>	<p>The number of <code>dynamic_feat</code> provided in the data. For example, if two dynamic features are provided, set this to 2. Set this to <code>auto</code> to use the <code>dynamic_feat</code> field if it is present in the data to extract the number from the dataset. Set it to an empty string to disable using the <code>dynamic_feat</code> field even if it is present in the data.</p> <p>Optional</p> <p>Valid values: positive integer, empty string, or <code>auto</code></p> <p>Default value: <code>auto</code></p>
<code>num_eval_samples</code>	<p>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.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 100</p>
<code>num_layers</code>	<p>The number of hidden layers in the RNN. Typical values range from 1 to 4.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 2</p>
<code>test_quantiles</code>	<p>Quantiles for which to calculate quantile loss on the test channel.</p> <p>Optional</p> <p>Valid values: array of floats</p> <p>Default value: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]</p>

Tuning 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 \(p. 50\)](#).

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 [Recommended Best Practices \(p. 103\)](#).

Metric Name	Description	Optimization Direction
<code>test:RMSE</code>	Root mean square error between forecast and actual target computed on the test set.	Minimize
<code>test:mean_wQuantileLoss</code>	Average overall quantile losses computed on the test set. Setting the <code>test_quantiles</code> hyperparameter controls which quantiles are used.	Minimize
<code>train:final_loss</code>	Training negative log-likelihood loss averaged over the last training epoch for the model.	Minimize

Tunable Hyperparameters

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
<code>mini_batch_size</code>	<code>IntegerParameterRanges</code>	MinValue: 32, MaxValue: 1028
<code>epochs</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 1000
<code>context_length</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 200
<code>num_cells</code>	<code>IntegerParameterRanges</code>	MinValue: 30, MaxValue: 200
<code>num_layers</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 8
<code>dropout_rate</code>	<code>ContinuousParameterRange</code>	MinValue: 0.00, MaxValue: 0.2
<code>embedding_dimension</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 50
<code>learning_rate</code>	<code>ContinuousParameterRange</code>	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 custom feature time series (`dynamic_feat`), the same number of `dynamic_feat` have to be provided 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. Since the forecast for each time series typically takes somewhere around 300 - 1000 milliseconds, or longer depending on the model size, passing too many time series in a single request may lead to timeouts. In this case, it is better to send fewer time series per request and send more requests. Since DeepAR uses multiple workers per instance one 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 size is large and there is not 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 SageMaker's [CreateModel \(p. 359\)](#) API.

Batch Transform

DeepAR forecasting supports getting inferences by using batch transform using the JSON lines format where 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 \(p. 101\)](#). 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]]}
```

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 \(p. 376\)](#) 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": [...] }
```

Factorization Machines

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 \(p. 114\)](#)
- [EC2 Instance Recommendation \(p. 115\)](#)
- [Factorization Machines Sample Notebooks \(p. 115\)](#)
- [How Factorization Machines Work \(p. 115\)](#)
- [Factorization Machines Hyperparameters \(p. 116\)](#)
- [Tuning a Factorization Machines Model \(p. 121\)](#)
- [Factorization Machine Response Formats \(p. 122\)](#)

Input/Output Interface

The factorization machine algorithm can be run in either binary classification mode or regression mode. In each mode, a dataset can be provided to the test channel along with the train channel dataset. 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).

The factorization machines algorithm currently supports training only on 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 binary classification models, both the score and the predicted label are returned. For regression, just the score is returned.

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

EC2 Instance Recommendation

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 [Using Notebook Instances \(p. 67\)](#). 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_i) 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:

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

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 i^{th} variable.
- The $\langle v_i, v_j \rangle$ factorization terms model the pairwise interaction between the i^{th} 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

Parameter Name	Description
<code>feature_dim</code>	<p>The dimension of the input feature space. This could be very high with sparse input.</p> <p>Required</p> <p>Valid values: Positive integer. Suggested value range: [10000,10000000]</p>
<code>num_factors</code>	<p>The dimensionality of factorization.</p> <p>Required</p> <p>Valid values: Positive integer. Suggested value range: [2,1000], 64 usually optimal.</p>
<code>predictor_type</code>	<p>The type of predictor.</p> <ul style="list-style-type: none"> • <code>binary_classifier</code>: For binary classification tasks. • <code>regressor</code>: For regression tasks. <p>Required</p> <p>Valid values: String: <code>binary_classifier</code> or <code>regressor</code></p>
<code>bias_init_method</code>	<p>The initialization method for the bias term:</p> <ul style="list-style-type: none"> • <code>normal</code>: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>bias_init_sigma</code>. • <code>uniform</code>: Initializes weights with random values uniformly sampled from a range specified by <code>-bias_init_scale</code>, <code>+bias_init_scale</code>. • <code>constant</code>: Initializes the weights to a scalar value specified by <code>bias_init_value</code>. <p>Optional</p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code></p>

Parameter Name	Description
	Default value: <code>normal</code>
<code>bias_init_scale</code>	<p>Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>bias_init_sigma</code>	<p>The standard deviation for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>bias_init_value</code>	<p>The initial value of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>constant</code>.</p> <p>Optional</p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>bias_lr</code>	<p>The learning rate for the bias term.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.1</p>
<code>bias_wd</code>	<p>The weight decay for the bias term.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>clip_gradient</code>	<p>Gradient clipping optimizer parameter. Clips the gradient by projecting onto the interval [-<code>clip_gradient</code>, +<code>clip_gradient</code>].</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: None</p>

Parameter Name	Description
<code>epochs</code>	<p>The number of training epochs to run.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 1</p>
<code>eps</code>	<p>Epsilon parameter to avoid division by 0.</p> <p>Optional</p> <p>Valid values: Float. Suggested value: small.</p> <p>Default value: None</p>
<code>factors_init_method</code>	<p>The initialization method for factorization terms:</p> <ul style="list-style-type: none"> <code>normal</code>: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>factors_init_sigma</code>. <code>uniform</code>: Initializes weights with random values uniformly sampled from a range specified by <code>[-factors_init_scale, +factors_init_scale]</code>. <code>constant</code>: Initializes the weights to a scalar value specified by <code>factors_init_value</code>. <p>Optional</p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>
<code>factors_init_scale</code>	<p>The range for initialization of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>factors_init_sigma</code>	<p>The standard deviation for initialization of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>

Parameter Name	Description
<code>factors_init_value</code>	<p>The initial value of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>constant</code>.</p> <p>Optional</p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>factors_lr</code>	<p>The learning rate for factorization terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.0001</p>
<code>factors_wd</code>	<p>The weight decay for factorization terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.00001</p>
<code>linear_lr</code>	<p>The learning rate for linear terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>
<code>linear_init_method</code>	<p>The initialization method for linear terms:</p> <ul style="list-style-type: none"> <code>normal</code> Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>linear_init_sigma</code>. <code>uniform</code> Initializes weights with random values uniformly sampled from a range specified by <code>[-linear_init_scale, +linear_init_scale]</code>. <code>constant</code> Initializes the weights to a scalar value specified by <code>linear_init_value</code>. <p>Optional</p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>

Parameter Name	Description
<code>linear_init_scale</code>	<p>Range for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>linear_init_sigma</code>	<p>The standard deviation for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>linear_init_value</code>	<p>The initial value of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>constant</code>.</p> <p>Optional</p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>linear_wd</code>	<p>The weight decay for linear terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>
<code>mini_batch_size</code>	<p>The size of mini-batch used for training.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 1000</p>
<code>rescale_grad</code>	<p>Gradient rescaling optimizer parameter. If set, multiplies the gradient with <code>rescale_grad</code> before updating. Often choose to be 1.0/batch_size.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: None</p>

Tuning 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 [Automatic Model Tuning \(p. 50\)](#).

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
<code>test:rmse</code>	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
<code>test:binary_classification_accuracy</code>	Accuracy	Maximize
<code>test:binary_classification_cross_entropy</code>	Cross Entropy	Minimize
<code>test:binary_f_beta</code>	Beta	Maximize

Tunable 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
<code>bias_init_scale</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==uniform</code>
<code>bias_init_sigma</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==normal</code>

Parameter Name	Parameter Type	Recommended Ranges	Dependency
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
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

Binary classification

```
let response = {
  "predictions": [
    {
      "score": 0.4,
```

```
        "predicted_label": 0
    }
]
```

Regression

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

JSONLINES

Binary classification

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

Regression

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

RECORDIO

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 takes an image as input and classifies it into one of multiple output categories. 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 \(p. 124\)](#)
- [EC2 Instance Recommendation \(p. 125\)](#)
- [Image Classification Sample Notebooks \(p. 126\)](#)
- [How Image Classification Works \(p. 126\)](#)
- [Hyperparameters \(p. 126\)](#)
- [Tuning an Image Classification Model \(p. 131\)](#)

Input/Output Interface

The Amazon SageMaker Image Classification algorithm supports both RecordIO (`application/x-recordio`) and image (`application/x-image`) content types for training. The algorithm supports only `application/x-image` for inference.

Training 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 \(p. 371\)](#) 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`.

Training with Image Format

If you use the Image format for training, specify `train`, `validation`, `train_1st`, and `validation_1st` channels as values for the `InputDataConfig` parameter of the [CreateTrainingJob \(p. 371\)](#) request. Specify the individual image data (`.jpg` or `.png` files) for the `train` and `validation` channels. Specify one `.1st` file in each of the `train_1st` and `validation_1st` channels. Set the content type for all four channels to `application/x-image`.

A `.1st` 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.

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 \(p. 371\)](#) 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 \(p. 7\)](#).

Inference with Image Format

The generated models can be hosted for inference and support encoded .jpg and .png image formats as 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 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 [Using Notebook Instances \(p. 67\)](#). 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.

Hyperparameters

Parameter Name	Description
num_classes	<p>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.</p> <p>Required</p> <p>Valid values: positive integer</p>
num_training_samples	<p>Number of training examples in the input dataset.</p> <p>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.</p> <p>Required</p>

Parameter Name	Description
	Valid values: positive integer
augmentation_type	<p>Data augmentation type. The input images can be augmented in multiple ways as specified below.</p> <ul style="list-style-type: none"> • <code>crop</code>: Randomly crop the image and flip the image horizontally • <code>crop_color</code>: In addition to ‘<code>crop</code>’, three random values in the range [-36, 36], [-50, 50], and [-50, 50] are added to the corresponding Hue-Saturation-Lightness channels respectively • <code>crop_color_transform</code>: In addition to <code>crop_color</code>, 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. <p>Optional</p> <p>Valid values: <code>crop</code>, <code>crop_color</code>, or <code>crop_color_transform</code>.</p> <p>Default value: no default value</p>
beta_1	<p>The beta1 for adam, that is the exponential decay rate for the first moment estimates.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>
beta_2	<p>The beta2 for adam, that is the exponential decay rate for the second moment estimates.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.999</p>
checkpoint_frequency	<p>Period to store model parameters (in number of epochs).</p> <p>Optional</p> <p>Valid values: positive integer no greater than epochs.</p> <p>Default value: None (Save checkpoint at the epoch that has the best validation accuracy.)</p>
epochs	<p>Number of training epochs.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 30</p>

Parameter Name	Description
eps	<p>The epsilon for adam and rmsprop. It is usually set to a small value to avoid division by 0.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 1e-8</p>
gamma	<p>The gamma for rmsprop, the decay factor for the moving average of the squared gradient.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>
image_shape	<p>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.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: '3, 224, 224'</p>
kv_store	<p>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.</p> <p>This parameter is not applicable to single machine training.</p> <ul style="list-style-type: none"> • <code>dist_sync</code>: The gradients are synchronized after every batch with all the workers. With <code>dist_sync</code>, batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then <code>dist_sync</code> behaves like local with batch size $n \times b$ • <code>dist_async</code>: 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. <p>Optional</p> <p>Valid values: <code>dist_sync</code> or <code>dist_async</code></p> <p>Default value: no default value</p>

Parameter Name	Description
<code>learning_rate</code>	<p>Initial learning rate.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.1</p>
<code>lr_scheduler_factor</code>	<p>The ratio to reduce learning rate used in conjunction with the <code>lr_scheduler_step</code> parameter, defined as $lr_{new} = lr_{old} * lr_scheduler_factor$.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.1</p>
<code>lr_scheduler_step</code>	<p>The epochs at which to reduce the learning rate. As explained in the <code>lr_scheduler_factor</code> parameter, the learning rate is reduced by <code>lr_scheduler_factor</code> at these epochs. For example, if the value is set to "10, 20", then the learning rate is reduced by <code>lr_scheduler_factor</code> after 10th epoch and again by <code>lr_scheduler_factor</code> after 20th epoch. The epochs are delimited by ",".</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: no default value</p>
<code>mini_batch_size</code>	<p>The batch size for training. In a single-machine multi-GPU setting, each GPU handles <code>mini_batch_size/num_gpu</code> training samples. For the multi-machine training in <code>dist_sync</code> mode, the actual batch size is <code>mini_batch_size*number of machines</code>. See MXNet docs for more details.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 32</p>
<code>momentum</code>	<p>The momentum for <code>sgd</code> and <code>nag</code>, ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0</p>

Parameter Name	Description
multi_label	<p>Flag to use for multi-label classification where each sample can be assigned multiple labels. Average accuracy across all classes is logged.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
num_layers	<p>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].</p> <p>Optional</p> <p>Valid values: positive integer in [18, 34, 50, 101, 152, 200] or [20, 32, 44, 56, 110]</p> <p>Default value: 152</p>
optimizer	<p>The optimizer type. For more details of the parameters for the optimizers, please refer to MXNet's API.</p> <p>Optional</p> <p>Valid values: One of <code>sgd</code>, <code>adam</code>, <code>rmsprop</code>, or <code>nag</code>.</p> <ul style="list-style-type: none"> • <code>sgd</code>: Stochastic gradient descent • <code>adam</code>: Adaptive momentum estimation • <code>rmsprop</code>: Root mean square propagation • <code>nag</code>: Nesterov accelerated gradient <p>Default value: <code>sgd</code></p>
precision_dtype	<p>The precision of the weights used for training. The algorithm can use either single precision (<code>float32</code>) or half precision (<code>float16</code>) for the weights. Using half-precision for weights results in reduced memory consumption.</p> <p>Optional</p> <p>Valid values: <code>float32</code> or <code>float16</code></p> <p>Default value: <code>float32</code></p>

Parameter Name	Description
<code>resize</code>	<p>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.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: no default value</p>
<code>top_k</code>	<p>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.</p> <p>Optional</p> <p>Valid values: positive integer larger than 1.</p> <p>Default value: no default value</p>
<code>use_pretrained_model</code>	<p>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.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>use_weighted_loss</code>	<p>Flag to use weighted cross-entropy loss for multi-label classification (used only when <code>multi_label = 1</code>), where the weights are calculated based on the distribution of classes.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>weight_decay</code>	<p>The coefficient weight decay for <code>sgd</code> and <code>nag</code>, ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.0001</p>

Tuning 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 \(p. 50\)](#).

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 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 [Hyperparameters \(p. 126\)](#).

Parameter Name	Parameter Type	Recommended Ranges
<code>beta_1</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
<code>beta_2</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
<code>eps</code>	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 1.0
<code>gamma</code>	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 0.999
<code>learning_rate</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.5
<code>mini_batch_size</code>	IntegerParameterRanges	MinValue: 8, MaxValue: 512
<code>momentum</code>	ContinuousParameterRanges	MinValue: 0.0, MaxValue: 0.999
<code>optimizer</code>	CategoricalParameterRanges	[<code>'sgd'</code> , <code>'adam'</code> , <code>'rmsprop'</code> , <code>'nag'</code>]
<code>weight_decay</code>	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 \(p. 133\)](#)
- [EC2 Instance Recommendation \(p. 134\)](#)
- [IP Insights Sample Notebooks \(p. 134\)](#)
- [How IP Insights Works \(p. 135\)](#)
- [IP Insights Hyperparameters \(p. 136\)](#)
- [Tuning an IP Insights Model \(p. 138\)](#)
- [IP Insights Data Formats \(p. 140\)](#)

Input/Output Interface

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 \(p. 140\)](#).

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—Inference \(p. 85\)](#). 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 \(p. 140\)](#).

EC2 Instance Recommendation

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

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

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.

Table 1: EC2 instances based on the value of `vector_dim` (first column) and `num_entity_vectors` (first row).

<code>vector_dim</code> \ <code>num_entity_vectors</code>	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	ml.m5.4xlarge
128	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge	ml.m5.4xlarge	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				

Table 1: EC2 instances based on the value of `vector_dim` (first column) and `num_entity_vectors` (first row).

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 SageMaker IP Insights Algorithm](#). For

instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Using Notebook Instances \(p. 67\)](#). 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 [CreateTransformJob \(p. 376\)](#) 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
<code>num_entity_vectors</code>	<p>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.</p> <p>Required</p> <p>Valid values: 1 ≤ positive integer ≤ 250,000,000</p>
<code>vector_dim</code>	<p>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.</p> <p>Required</p> <p>Valid values: 4 ≤ positive integer ≤ 4096</p>
<code>batch_metrics_publish_interval</code>	<p>The interval (every X batches) at which the Apache MXNet Speedometer function prints the training speed of the network (samples/second).</p> <p>Optional</p> <p>Valid values: positive integer ≥ 1</p> <p>Default value: 1,000</p>

Parameter Name	Description
epochs	<p>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.</p> <p>Optional</p> <p>Valid values: positive integer ≥ 1</p> <p>Default value: 10</p>
learning_rate	<p>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.</p> <p>Optional</p> <p>Valid values: $1\text{e-}6 \leq \text{float} \leq 10.0$</p> <p>Default value: 0.001</p>
mini_batch_size	<p>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 <code>mini_batch_size</code>, the faster the training and the greater the number of possible shuffled-negative-sample combinations. However, with a large <code>mini_batch_size</code>, the training is more likely to converge to a poor local minimum and perform relatively worse for inference.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{positive integer} \leq 500000$</p> <p>Default value: 10,000</p>
num_ip_encoder_layers	<p>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.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{positive integer} \leq 100$</p> <p>Default value: 1</p>

Parameter Name	Description
<code>random_negative_sampling_rate</code>	<p>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 <code>random_negative_sampling_rate</code> (R) and <code>shuffled_negative_sampling_rate</code> (S) must be in the interval: $1 \leq R + S \leq 500$.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{positive integer} \leq 500$</p> <p>Default value: 1</p>
<code>shuffled_negative_sampling_rate</code>	<p>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 <code>random_negative_sampling_rate</code> (R) and <code>shuffled_negative_sampling_rate</code> (S) must be in the interval: $1 \leq R + S \leq 500$.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{positive integer} \leq 500$</p> <p>Default value: 1</p>
<code>weight_decay</code>	<p>The weight decay coefficient. This parameter adds an L2 regularization factor that is required to prevent the model from overfitting the training data.</p> <p>Optional</p> <p>Valid values: $0.0 \leq \text{float} \leq 10.0$</p> <p>Default value: 0.00001</p>

Tuning 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 [Automatic Model Tuning \(p. 50\)](#).

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 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_samples	IntegerParameterRanges	MinValue: 0, MaxValue: 10
shuffled_negative_samples	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 \(p. 140\)](#)
- [IP Insights Inference Data Formats \(p. 140\)](#)

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—Training \(p. 82\)](#). 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 [Common Data Formats—Inference \(p. 85\)](#). However, the Amazon SageMaker IP Insights algorithm does not currently support RecordIO format.

IP Insights Input Request 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
```

INPUT: JSON

JSON data can be provided in different formats. IP Insights follows the common Amazon SageMaker formats. For more information about inference formats, see [Common Data Formats—Inference \(p. 85\)](#).

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

The JSON Lines content type is useful for running batch transform jobs. For more information on Amazon SageMaker inference formats, see [Common Data Formats—Inference \(p. 85\)](#). For more information on running batch transform jobs, see [Getting Inferences by Using Amazon SageMaker Batch Transform \(p. 19\)](#).

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

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 SageMaker IP 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

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 \(p. 143\)](#).

Topics

- [Input/Output Interface \(p. 142\)](#)
- [EC2 Instance Recommendation \(p. 143\)](#)
- [K-Means Sample Notebooks \(p. 143\)](#)
- [How K-Means Clustering Works \(p. 143\)](#)
- [K-Means Hyperparameters \(p. 146\)](#)
- [Tuning a K-Means Model \(p. 148\)](#)
- [k-means Response Formats \(p. 149\)](#)

Input/Output Interface

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 \(p. 149\)](#) for inference and the [K-Means Sample Notebooks \(p. 143\)](#).

EC2 Instance Recommendation

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 [Using Notebook Instances \(p. 67\)](#). 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 MINST 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 \(p. 371\)](#). 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 \(p. 144\)](#)
- [Step 2: Iterate over the Training Dataset and Calculate Cluster Centers \(p. 145\)](#)
- [Step 3: Reduce the Clusters from K to k \(p. 146\)](#)

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 \(p. 371\)](#). For an example, see [Step 3.3.2: Create a Training Job \(p. 34\)](#).

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 $c_1 = 100$ previously assigned points. You added 25 points from the mini-batch in this step.

Cluster $c_2 = 150$ previously assigned points. You added 40 points from the mini-batch in this step.

Cluster $c_3 = 450$ 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 \(p. 371\)](#) 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 \(p. 143\)](#).

Parameter Name	Description
<code>feature_dim</code>	The number of features in the input data. Required Valid values: Positive integer
<code>k</code>	The number of required clusters. Required Valid values: Positive integer
<code>epochs</code>	The number of passes done over the training data. Optional Valid values: Positive integer Default value: 1
<code>eval_metrics</code>	A JSON list of metric types used to report a score for the model. Allowed values are <code>msd</code> for Means Square Error and <code>ssd</code> for Sum of Square Distance. If test data is provided, the score is reported for each of the metrics requested. Optional Valid values: Either <code>[\"msd\"]</code> or <code>[\"ssd\"]</code> or <code>[\"msd\", \"ssd\"]</code> . Default value: <code>[\"msd\"]</code>
<code>extra_center_factor</code>	The algorithm creates K centers = <code>num_clusters * extra_center_factor</code> 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 <code>auto</code> . Default value: <code>auto</code>

Parameter Name	Description
half_life_time_size	<p>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 <code>half_life_time_size</code> points, its weight is 1/2. If set to 0, there is no decay.</p> <p>Optional</p> <p>Valid values: Non-negative integer</p> <p>Default value: 0</p>
init_method	<p>Method by which the algorithm chooses the initial cluster centers. The standard k-means approach chooses them at random. An 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.</p> <p>Optional</p> <p>Valid values: Either <code>random</code> or <code>kmeans++</code>.</p> <p>Default value: <code>random</code></p>
local_lloyd_init_method	<p>The initialization method for Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p>Optional</p> <p>Valid values: Either <code>random</code> or <code>kmeans++</code>.</p> <p>Default value: <code>kmeans++</code></p>
local_lloyd_max_iter	<p>The maximum number of iterations for Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 300</p>
local_lloyd_num_trials	<p>The number of times the Lloyd's expectation-maximization (EM) procedure with the least loss is run when building the final model containing <code>k</code> centers.</p> <p>Optional</p> <p>Valid values: Either a positive integer or <code>auto</code>.</p> <p>Default value: <code>auto</code></p>

Parameter Name	Description
<code>local_lloyd_tol</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Float. Range in [0, 1].</p> <p>Default value: 0.0001</p>
<code>mini_batch_size</code>	<p>The number of observations per mini-batch for the data iterator.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 5000</p>

Tuning 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 \(p. 50\)](#).

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
<code>test:msd</code>	Mean squared distances between each record in the test set and the closest center of the model.	Minimize
<code>test:ssd</code>	Sum of the squared distances between each record in the test set and the closest center of the model.	Minimize

Tunable 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
<code>epochs</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue:10
<code>extra_center_factor</code>	<code>IntegerParameterRanges</code>	MinValue: 4, MaxValue:10
<code>init_method</code>	<code>CategoricalParameterRanges</code>	<code>['kmeans++', 'random']</code>
<code>mini_batch_size</code>	<code>IntegerParameterRanges</code>	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

```
{
    "predictions": [
        {
            "closest_cluster": 1.0,
            "distance_to_cluster": 3.0,
        },
        {
            "closest_cluster": 2.0,
            "distance_to_cluster": 5.0,
        },
        ...
    ]
}
```

JSONLINES

```
{"closest_cluster": 1.0, "distance_to_cluster": 3.0}
{"closest_cluster": 2.0, "distance_to_cluster": 5.0}
```

RECORDIO

```
[
    Record = {
        features = {},
        label = {
            'closest_cluster': {
                keys: [],
                values: [1.0, 2.0] # float32
            },
            'distance_to_cluster': {
                keys: [],
                values: []
            }
        }
    }
]
```

```
        values: [3.0, 5.0] # float32
    },
]
}
```

K-Nearest Neighbors

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 \(p. 150\)](#)
- [kNN Sample Notebooks \(p. 151\)](#)
- [How It Works \(p. 151\)](#)
- [EC2 Instance Recommendation \(p. 152\)](#)
- [K-Nearest Neighbors Hyperparameters \(p. 152\)](#)
- [Tuning a K-Nearest Neighbors Model \(p. 154\)](#)
- [Data Formats for K-Nearest Neighbors Training Input \(p. 155\)](#)
- [K-NN Request and Response Formats \(p. 156\)](#)

Input/Output Interface

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-Nearest Neighbors Training Input \(p. 155\)](#) for training, [K-NN Request and Response Formats \(p. 156\)](#) for inference, and the [kNN Sample Notebooks \(p. 151\)](#).

kNN 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 [Using Notebook Instances \(p. 67\)](#). 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 It Works

Step 1: Sampling

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: 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: Building 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.

Model Serialization

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

Training

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.

Inference

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-Nearest Neighbors Hyperparameters

Parameter Name	Description
<code>feature_dim</code>	The number of features in the input data. Required Valid values: positive integer.
<code>k</code>	The number of nearest neighbors.

Parameter Name	Description
	Required Valid values: positive integer
<code>predictor_type</code>	The type of inference to use on the data labels. Required Valid values: <i>classifier</i> for classification or <i>regressor</i> for regression.
<code>sample_size</code>	The number of data points to be sampled from the training data set. Required Valid values: positive integer
<code>dimension_reduction_target</code>	The target dimension to reduce to. Required when you specify the <code>dimension_reduction_type</code> parameter. Valid values: positive integer greater than 0 and less than <code>feature_dim</code> .
<code>dimension_reduction_type</code>	The type of dimension reduction method. Optional Valid values: <i>sign</i> for random projection or <i>fjlt</i> for the fast Johnson-Lindenstrauss transform. Default value: No dimension reduction
<code>faiss_index_ivf_nlists</code>	The number of centroids to construct in the index when <code>index_type</code> is <i>faiss.IVFFlat</i> or <i>faiss.IVFPQ</i> . Optional Valid values: positive integer Default value: <i>auto</i> , which resolves to <code>sqrt(sample_size)</code> .
<code>faiss_index_pq_m</code>	The number of vector sub-components to construct in the index when <code>index_type</code> is set to <i>faiss.IVFPQ</i> . The Facebook AI Similarity Search (FAISS) library requires that the value of <code>faiss_index_pq_m</code> is a divisor of the data dimension. If <code>faiss_index_pq_m</code> is not a divisor of the data dimension, we increase the data dimension to smallest integer divisible by <code>faiss_index_pq_m</code> . 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 <code>dimension_reduction_target</code> 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

Parameter Name	Description
<code>index_metric</code>	<p>The metric to measure the distance between points when finding nearest neighbors. When training with <code>index_type</code> set to <code>faiss.IVFPQ</code>, the <code>INNER_PRODUCT</code> distance and <code>COSINE</code> similarity are not supported.</p> <p>Optional</p> <p>Valid values: <code>L2</code> for Euclidean-distance, <code>INNER_PRODUCT</code> for inner-product distance, <code>COSINE</code> for cosine similarity.</p> <p>Default value: <code>L2</code></p>
<code>index_type</code>	<p>The type of index.</p> <p>Optional</p> <p>Valid values: <code>faiss.Flat</code>, <code>faiss.IVFFlat</code>, <code>faiss.IVFPQ</code>.</p> <p>Default values: <code>faiss.Flat</code></p>
<code>mini_batch_size</code>	<p>The number of observations per mini-batch for the data iterator.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5000</p>

Tuning a K-Nearest Neighbors 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 [Automatic Model Tuning \(p. 50\)](#).

Metrics Computed by the K-Nearest Neighbors 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
<code>test:accuracy</code>	When <code>predictor_type</code> is set to <i>classifier</i> , 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
<code>test:mse</code>	When <code>predictor_type</code> is set to <i>regressor</i> , 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 Hyperparameters

Tune the Amazon SageMaker k-nearest neighbor model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
<code>k</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 1024
<code>sample_size</code>	<code>IntegerParameterRanges</code>	MinValue: 256, MaxValue: 20000000

Data Formats for K-Nearest Neighbors 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

`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

`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

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

content-type: application/json

```
{
    "instances": [
        {"data": {"features": {"values": [-3, -1, -4, 2]}},  
         {"features": [3.0, 0.1, 0.04, 0.002]}}
    ]
}
```

INPUT: JSONLINES

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

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

accept: application/json

```
{
    "predictions": [
        {"predicted_label": 0.0},
        {"predicted_label": 2.0}
    ]
}
```

OUTPUT: JSONLINES

accept: application/jsonlines

```
{"predicted_label": 0.0}
{"predicted_label": 2.0}
```

OUTPUT: VERBOSE JSON

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

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

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 regressor tasks:

```
[06/08/2018 20:15:33 INFO 140026520049408] #test_score (algo-1) : ('mse', 0.01333333333333334)
```

For classifier tasks:

```
[06/08/2018 20:15:46 INFO 140285487171328] #test_score (algo-1) : ('accuracy',  
0.9866666666666666)
```

Latent Dirichlet Allocation (LDA)

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	<i>eat</i>	<i>sleep</i>	<i>play</i>	<i>meow</i>	<i>bark</i>
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 \(p. 159\)](#)
- [EC2 Instance Recommendation \(p. 160\)](#)
- [LDA Sample Notebooks \(p. 160\)](#)
- [How LDA Works \(p. 160\)](#)
- [LDA Hyperparameters \(p. 162\)](#)
- [Tuning an LDA Model \(p. 163\)](#)

Input/Output Interface

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 example notebooks for more detail on training and inference formats.

EC2 Instance Recommendation

LDA currently only supports single-instance CPU training. CPU instances are recommended for hosting/inference.

LDA Sample 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 [Using Notebook Instances \(p. 67\)](#). 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 $\theta \sim \text{Dirichlet}(\alpha)$.

For each word in the document:

- Choose a topic $z \sim \text{Multinomial}(\theta)$
- Choose the corresponding topic-word distribution β_z .
- Draw a word $w \sim \text{Multinomial}(\beta_z)$.

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 $V \times V \times V$ 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 $V \times V \times V$ tensor is prohibitively large to store in memory.
2. Instead, it uses a $V \times V$ moment matrix, which is the two-dimensional analog of the tensor from step 1, to find a whitening matrix of dimension $V \times k$. This matrix can be used to convert the $V \times V$ moment matrix into a $k \times k$ identity matrix. k is the number of topics in the model.
3. This same whitening matrix can then be used to find a smaller $k \times k \times k$ tensor. When spectrally decomposed, this tensor has components that have a simple relationship with the components of the $V \times V \times V$ tensor.
4. *Alternating Least Squares* is used to decompose the smaller $k \times k \times k$ 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 \(p. 160\)](#).

Parameter Name	Description
<code>num_topics</code>	<p>The number of topics for LDA to find within the data.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>feature_dim</code>	<p>The size of the vocabulary of the input document corpus.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>mini_batch_size</code>	<p>The total number of documents in the input document corpus.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>alpha0</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Positive float</p> <p>Default value: 0.1</p>
<code>max_restarts</code>	<p>The number of restarts to perform during the Alternating Least Squares (ALS) spectral decomposition phase of the algorithm. Can be used to find better quality local minima at the expense of additional computation, but typically should not be adjusted.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 10</p>
<code>max_iterations</code>	<p>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.</p>

Parameter Name	Description
	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

Tuning 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 [Automatic Model Tuning \(p. 50\)](#).

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
<code>test:pwll</code>	Per-word log-likelihood on the test dataset. The likelihood that the test dataset is accurately described by the learned LDA model.	Maximize

Tunable 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

Linear models are supervised learning algorithms used for solving either classification or regression problems. As input the model is given labeled examples (\mathbf{x}, \mathbf{y}) . \mathbf{x} is a high dimensional vector and \mathbf{y} is a numeric label. For binary classification problems, the algorithm expects the label to be either 0 or 1. For multiclass classification problems, the algorithm expects the labels to be from 0 to `num_classes` - 1. For regression problems, \mathbf{y} is a real number. The algorithm learns a linear function, or linear threshold function for classification, mapping a vector \mathbf{x} to an approximation of the label \mathbf{y} .

The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. This allows you to simultaneously explore different training objectives and choose the best solution from a validation set. It also allows you to explore a large number of models and choose the best, which optimizes either continuous objectives—such as mean square error, cross entropy loss, absolute error, and so on—or discrete objectives suited for classification, such as F1 measure, precision@recall, or accuracy. When compared with solutions providing a solution to only continuous objectives, the implementation provides a significant increase in speed over naive hyperparameter optimization techniques and added convenience.

The linear learner expects a data matrix, with rows representing the observations, and columns the dimensions of the features. It also requires an additional column containing 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 \(p. 371\)](#). 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 on which you train. Examples include the number of epochs, regularization, and loss type.

Topics

- [Input/Output Interface \(p. 164\)](#)
- [EC2 Instance Recommendation \(p. 165\)](#)
- [Linear Learner Sample Notebooks \(p. 165\)](#)
- [How It Works \(p. 165\)](#)
- [Linear Learner Hyperparameters \(p. 166\)](#)
- [Tuning a Linear Learner Model \(p. 174\)](#)
- [Linear Learner Response Formats \(p. 176\)](#)

Input/Output Interface

Amazon SageMaker linear learner supports three data channels: train, validation, and test. The validation data channel is optional. If you provide validation data, it should be `FullyReplicated`. The validation loss is logged at every epoch, and a sample of the validation data is used to calibrate and select the best model. If you don't provide validation data, the final model calibration and selection uses a sample of the training data. The test data channel is also optional. If test data is provided, the algorithm logs contain the test score for the final model.

For training, linear learner supports both `recordIO wrapped protobuf` and `csv`. For input type `application/x-recordio-protobuf`, only `Float32` tensors are supported. For input type `text/csv`, 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, linear learner supports the `application/json`, `application/x-recordio-protobuf`, and `text/csv` formats. For binary classification models, both the score and the predicted label are returned. For regression, just the score is returned.

For more information on input and output file formats, see [Linear Learner Response Formats \(p. 176\)](#) for inference and the [Linear Learner Sample Notebooks \(p. 165\)](#).

EC2 Instance Recommendation

Linear learner can be trained on single- or multi-machine CPU and GPU instances. During our testing, we have not found substantial evidence to multi-GPU to be faster than single GPU, but results vary depending on the specific use case.

Linear Learner Sample Notebooks

For a sample notebook that uses the Amazon SageMaker linear learner machine learning 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 how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Using Notebook Instances \(p. 67\)](#). 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 It Works

Note

We assume that the input data is shuffled. If not, for example if the data is ordered by label, the method fails.

Step 1: Preprocessing

If the option is turned on, the algorithm first goes over a small sample of the data to learn its characteristics. For every feature and for the label, you learn the mean value and the standard deviation.

This information is used during training. Based on the configuration, you normalize the data. That is, you shift it to have mean zero and scale it to have unit standard deviation. When the `auto` (default) value is specified to decide the normalization you:

- Shift and scale the label for regression problems, and leave it as is for classification problems
- Always scale the features
- Shift the features only for dense data

Step 2: Training

You train using a distributed implementation of stochastic gradient descent. The input allows you to control specifics of the optimization procedure by choosing the exact optimization algorithm, for example, Adam, Adagrad, SGD, and so on, and their parameters, such as momentum, learning rate, and the learning rate schedule. Without specified details, choose a default option that works for the majority of datasets.

During training, you simultaneously optimize multiple models, each with slightly different objectives: in other words, vary L1 or L2 regularization and try out different optimizer settings.

Step 3: Validation and Setting the Threshold

When the training is done, evaluate the different models on a validation set. For regression problems, output the model obtaining the best score on the validation set. When the objective is classification, use a sample of (raw prediction, label) pairs to tune the threshold for a provided objective. The raw prediction is the output of the trained linear function. Allow classification objectives based on the predicted label, such as F1 measure, accuracy, precision@recall, and so on. Choose the model that achieves the best score on the validation set.

Note

If you don't provide a validation set, the algorithm optimizes over the training set. In such a scenario, avoid exploring different regularization procedures.

Linear Learner Hyperparameters

Parameter Name	Description
<code>feature_dim</code>	<p>Number of features in the input data.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>num_classes</code>	<p>The number of classes for the response variable. The classes are assumed to be labeled 0, ..., num_classes - 1.</p> <p>Required when <code>predictor_type</code> is <code>multiclass_classifier</code>; otherwise ignored.</p> <p>Valid values: integers from 3 to 1,000,000</p>
<code>predictor_type</code>	<p>Specifies the type of target variable as a binary classification, multiclass classification, or regression.</p> <p>Required</p> <p>Valid values: <code>binary_classifier</code>, <code>multiclass_classifier</code>, or <code>regressor</code></p>
<code>accuracy_top_k</code>	<p>The value of k when computing the Top K Accuracy metric for multiclass classification. An example is scored as correct if the model assigns one of the top k scores to the true label.</p> <p>Optional</p> <p>Valid values: positive integers</p> <p>Default value: 3</p>
<code>balance_multiclass_weights</code>	<p>Specifies whether to use class weights which give each class equal importance in the loss function. Only used when <code>predictor_type</code> is <code>multiclass_classifier</code>.</p> <p>Optional</p> <p>Valid values: <code>true</code>, <code>false</code></p>

Parameter Name	Description
	Default value: <code>false</code>
<code>beta_1</code>	<p>Exponential decay rate for first moment estimates. Applies only when the <code>optimizer</code> value is <code>adam</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or float between 0 and 1.0</p> <p>Default value: <code>auto</code></p>
<code>beta_2</code>	<p>Exponential decay rate for second moment estimates. Applies only when the <code>optimizer</code> value is <code>adam</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or float between 0 and 1.0</p> <p>Default value: <code>auto</code></p>
<code>bias_lr_mult</code>	<p>Allows a different learning rate for the bias term. The actual learning rate for the bias is <code>learning_rate * bias_lr_mult</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive float</p> <p>Default value: <code>auto</code></p>
<code>bias_wd_mult</code>	<p>Allows different regularization for the bias term. The actual L2 regularization weight for the bias is <code>wd * bias_wd_mult</code>. By default there is no regularization on the bias term.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or non-negative float</p> <p>Default value: <code>auto</code></p>

Parameter Name	Description
binary_classifier_model_selection_criteria	<p>Selects the model evaluation criteria for the validation dataset (or for the training dataset if a validation dataset is not present) when <code>predictor_type</code> is set to <code>binary_classifier</code>. The following criteria are available:</p> <ul style="list-style-type: none"> • <code>accuracy</code>: model with highest accuracy. • <code>f_beta</code>: model with highest f1 score. The default is F1. • <code>precision_at_target_recall</code>: model with highest precision at a given recall target. • <code>recall_at_target_precision</code>: model with highest recall at a given precision target. • <code>loss_function</code>: model with lowest value of the loss function used in training. <p>Optional</p> <p>Valid values: <code>accuracy</code>, <code>f_beta</code>, <code>precision_at_target_recall</code>, <code>recall_at_target_precision</code>, or <code>loss_function</code></p> <p>Default value: <code>accuracy</code></p>
early_stopping_patience	<p>The number of epochs to wait before ending training if no improvement is made in the relevant metric. The metric is the <code>binary_classifier_model_selection_criteria</code> if provided, otherwise the metric is the same as <code>loss</code>. The metric is evaluated on the validation data. If no validation data is provided, the metric is always the same as <code>loss</code> and is evaluated on the training data. To disable early stopping, set <code>early_stopping_patience</code> to a value larger than <code>epochs</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
early_stopping_tolerance	<p>Relative tolerance to measure an improvement in <code>loss</code>. If the ratio of the improvement in <code>loss</code> divided by the previous best <code>loss</code> is smaller than this value, early stopping considers the improvement to be zero.</p> <p>Optional</p> <p>Valid values: positive float</p> <p>Default value: 0.001</p>
epochs	<p>Maximum number of passes over the training data.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 15</p>

Parameter Name	Description
f_beta	<p>The value of beta to use when calculating F score metrics for binary or multiclass classification. Also used if <code>binary_classifier_model_selection_criteria</code> is <code>f_beta</code>.</p> <p>Optional</p> <p>Valid values: positive float</p> <p>Default value: 1.0</p>
huber_delta	<p>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.</p> <p>Optional</p> <p>Valid values: positive float</p> <p>Default value: 1.0</p>
init_bias	<p>Initial weight for bias term.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0</p>
init_method	<p>Sets the initial distribution function used for model weights.</p> <ul style="list-style-type: none"> <code>uniform</code>: uniformly between (-scale, +scale) <code>normal</code>: normal, with mean 0 and sigma <p>Optional</p> <p>Valid values: <code>uniform</code> or <code>normal</code></p> <p>Default value: <code>uniform</code></p>
init_scale	<p>Scales an initial uniform distribution for model weights. Only applies when <code>init_method</code> is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: positive float</p> <p>Default value: 0.07</p>
init_sigma	<p>The initial standard deviation for the normal distribution. Applies only when <code>init_method</code> is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: positive float</p> <p>Default value: 0.01</p>

Parameter Name	Description
l1	<p>The L1 regularization parameter. Set the value to 0 if you do not want to use L1 regularization.</p> <p>Optional</p> <p>Valid values: auto or non-negative float</p> <p>Default value: auto</p>
learning_rate	<p>Step size used by the optimizer for parameter updates.</p> <p>Optional</p> <p>Valid values: auto or positive float</p> <p>Default value: auto, whose value depends on the <code>optimizer</code> chosen.</p>
loss	<p>Specifies the loss function to use.</p> <p>The loss functions available and their default values depend on the value of <code>predictor_type</code>:</p> <ul style="list-style-type: none"> • If the <code>predictor_type</code> is <code>regressor</code>, the available options are <code>auto</code>, <code>squared_loss</code>, <code>absolute_loss</code>, <code>eps_insensitive_squared_loss</code>, <code>eps_insensitive_absolute_loss</code>, <code>quantile_loss</code>, and <code>huber_loss</code>. The default value for <code>auto</code> is <code>squared_loss</code>. • If <code>predictor_type</code> is <code>binary_classifier</code>, the available options are <code>auto</code>, <code>logistic</code>, and <code>hinge_loss</code>. The default value for <code>auto</code> is <code>logistic</code>. • If <code>predictor_type</code> is <code>multiclass_classifier</code>, the available options are <code>auto</code> and <code>softmax_loss</code>. The default value for <code>auto</code> is <code>softmax_loss</code>. <p>Valid values: <code>auto</code>, <code>logistic</code>, <code>squared_loss</code>, <code>absolute_loss</code>, <code>hinge_loss</code>, <code>eps_insensitive_squared_loss</code>, <code>eps_insensitive_absolute_loss</code>, <code>quantile_loss</code>, or <code>huber_loss</code></p> <p>Optional</p> <p>Default value: auto</p>
loss_insensitivity	<p>Parameter for epsilon insensitive loss type. During training and metric evaluation, any error smaller than this is considered to be zero.</p> <p>Optional</p> <p>Valid values: positive float</p> <p>Default value: 0.01</p>

Parameter Name	Description
<code>lr_scheduler_factor</code>	<p>For every <code>lr_scheduler_step</code>, the learning rate decreases by this quantity. Applies only when the <code>use_lr_scheduler</code> is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive float between 0 and 1</p> <p>Default value: <code>auto</code></p>
<code>lr_scheduler_minimum_lr</code>	<p>The learning rate never decreases to a value lower than <code>lr_scheduler_minimum_lr</code>. Applies only when the <code>use_lr_scheduler</code> is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive float</p> <p>Default value: <code>auto</code></p>
<code>lr_scheduler_step</code>	<p>The number of steps between decreases of the learning rate. Applies only when the <code>use_lr_scheduler</code> is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default value: <code>auto</code></p>
<code>margin</code>	<p>Margin for <code>hinge_loss</code>.</p> <p>Optional</p> <p>Valid values: positive float</p> <p>Default value: 1.0</p>
<code>mini_batch_size</code>	<p>Number of observations per mini batch for the data iterator.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 1000</p>
<code>momentum</code>	<p>Momentum parameter of the <code>sgd</code> optimizer.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or float between 0 and 1.0</p> <p>Default value: <code>auto</code></p>
<code>normalize_data</code>	<p>Normalizes the features before training to have a <code>std_dev</code> of 1.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>true</code></p>

Parameter Name	Description
normalize_label	<p>Normalizes label. For regression, the label is normalized. For classification, it is not normalized. If <code>normalize_label</code> is set to <code>true</code> for classification, this parameter is ignored.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>
num_calibration_samples	<p>Number of observations to use from the validation dataset for model calibration (finding the best threshold).</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default value: <code>auto</code></p>
num_models	<p>Number of models to train in parallel. For the default <code>auto</code>, 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.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default values: <code>auto</code></p>
num_point_for_scaler	<p>Number of data points to use for calculating normalization or unbiasing of terms.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10,000</p>
optimizer	<p>The optimization algorithm to use.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>auto</code>: The default value. • <code>sgd</code>: Stochastic gradient descent • <code>adam</code>: Adaptive momentum estimation • <code>rmsprop</code>: A gradient-based optimization technique due to Geoffrey Hinton that uses a moving average of squared gradients to normalize the gradient. <p>Default value: <code>auto</code> Default setting for <code>auto</code> is <code>adam</code>.</p>

Parameter Name	Description
positive_example_weight	<p>Weight: assigned to positive examples when training a binary classifier. The weight of negative examples is fixed at 1. If <i>balanced</i>, then a weight is selected so that errors in classifying negative vs. positive examples have equal impact on the training loss. If <i>auto</i>, the algorithm attempts to select the weight that optimizes performance.</p> <p>Optional</p> <p>Valid values: <code>balanced</code>, <code>auto</code>, or a positive float</p> <p>Default value: 1.0</p>
quantile	<p>Quantile for quantile loss. For quantile q, the model attempts to produce predictions such that <code>true_label < prediction</code> with probability q.</p> <p>Optional</p> <p>Valid values: float between 0 and 1</p> <p>Default value: 0.5</p>
target_precision	<p>Target precision. If <code>binary_classifier_model_selection_criteria</code> is <code>recall_at_target_precision</code>, then precision is held at this value while recall is maximized.</p> <p>Optional</p> <p>Valid values: float between 0 and 1.0</p> <p>Default value: 0.8</p>
target_recall	<p>Target recall. If <code>binary_classifier_model_selection_criteria</code> is <code>precision_at_target_recall</code>, then recall is held at this value while precision is maximized.</p> <p>Optional</p> <p>Valid values: float between 0 and 1.0</p> <p>Default value: 0.8</p>
unbias_data	<p>Unbiases the features before training so the mean is 0. By default data is unbiased if <code>use_bias</code> is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>
unbias_label	<p>Unbiases labels before training so the mean is 0. Only done for regression if <code>use_bias</code> is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>

Parameter Name	Description
use_bias	<p>Specifies whether the model should include a bias term, which is the intercept term in the linear equation.</p> <p>Optional</p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
use_lr_scheduler	<p>If <code>true</code>, uses a scheduler for the learning rate.</p> <p>Optional</p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
wd	<p>The weight decay parameter, also known as the L2 regularization parameter. Set the value to 0 if you do not want to use L2 regularization.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or non-negative float</p> <p>Default value: <code>auto</code></p>

Tuning 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, which 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 \(p. 50\)](#).

Metrics Computed by the Linear Learner Algorithm

The linear learner algorithm reports five metrics, 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
<code>test:objective_loss</code>	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. You can set the loss to other types with the <code>loss</code> hyperparameter.	Minimize

Metric Name	Description	Optimization Direction
<code>test:binary_classifier_accuracy</code>	Accuracy of the final model on the test dataset.	Maximize
<code>test:binary_f_beta</code>	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
<code>test:precision</code>	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 <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.	Maximize
<code>test:recall</code>	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 <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.	Maximize
<code>validation:objective</code>	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 <code>loss</code> hyperparameter.	Minimize
<code>validation:binary_classifier_accuracy</code>	Accuracy of the final model on the validation dataset.	Maximize
<code>validation:binary_f_beta</code>	F_beta score of the final model on the validation dataset. By default, it is the F1 score, which is the harmonic mean of the <code>validation:precision</code> and <code>validation:recall</code> metrics.	Maximize
<code>validation:precision</code>	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 <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.	Maximize
<code>validation:recall</code>	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 <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.	Maximize

Tuning 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	ContinuousParameterRanges	MinValue: 1e-5, MaxValue: 1e5

Linear Learner Response Formats

JSON

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 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
    }
  ]
}
```

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

JSONLINES

Binary classification

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

Multiclass classification

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

Regression

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

RECORDIO

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)

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 \(p. 178\)](#)
- [EC2 Instance Recommendation \(p. 179\)](#)
- [NTM Sample Notebooks \(p. 179\)](#)
- [NTM Hyperparameters \(p. 179\)](#)
- [Tuning an NTM Model \(p. 182\)](#)
- [NTM Response Formats \(p. 183\)](#)

Input/Output Interface

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 \(p. 183\)](#) for inference and the [NTM Sample Notebooks \(p. 179\)](#).

EC2 Instance Recommendation

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 [Using Notebook Instances \(p. 67\)](#). 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	<p>The vocabulary size of the dataset.</p> <p>Required</p> <p>Valid values: Positive integer (min: 1, max: 1,000,000)</p>

Parameter Name	Description
<code>num_topics</code>	<p>The number of required topics.</p> <p>Required</p> <p>Valid values: Positive integer (min: 2, max: 1000)</p>
<code>batch_norm</code>	<p>Whether to use batch normalization during training.</p> <p>Optional</p> <p>Valid values: <i>true</i> or <i>false</i></p> <p>Default value: <i>false</i></p>
<code>clip_gradient</code>	<p>The maximum magnitude for each gradient component.</p> <p>Optional</p> <p>Valid values: Float (min: 1e-3)</p> <p>Default value: Infinity</p>
<code>encoder_layers</code>	<p>The number of layers in the encoder and the output size of each layer. When set to <i>auto</i>, the algorithm uses two layers of sizes $3 \times \text{num_topics}$ and $2 \times \text{num_topics}$ respectively.</p> <p>Optional</p> <p>Valid values: Comma-separated list of positive integers or <i>auto</i></p> <p>Default value: <i>auto</i></p>
<code>encoder_layers_activation</code>	<p>The activation function to use in the encoder layers.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>sigmoid</code>: Sigmoid function • <code>tanh</code>: Hyperbolic tangent • <code>relu</code>: Rectified linear unit <p>Default value: <code>sigmoid</code></p>
<code>epochs</code>	<p>The maximum number of passes over the training data.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 1)</p> <p>Default value: 50</p>

Parameter Name	Description
<code>learning_rate</code>	<p>The learning rate for the optimizer.</p> <p>Optional</p> <p>Valid values: Float (min: 1e-6, max: 1.0)</p> <p>Default value: 0.001</p>
<code>mini_batch_size</code>	<p>The number of examples in each mini batch.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 1, max: 10000)</p> <p>Default value: 256</p>
<code>num_patience_epochs</code>	<p>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 <code>num_patience_epochs</code> number of epochs. To disable early stopping, set <code>num_patience_epochs</code> to a value larger than <code>epochs</code>.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 1)</p> <p>Default value: 3</p>
<code>optimizer</code>	<p>The optimizer to use for training.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>sgd</code>: Stochastic gradient descent • <code>adam</code>: Adaptive momentum estimation • <code>adagrad</code>: Adaptive gradient algorithm • <code>adadelta</code>: An adaptive learning rate algorithm • <code>rmsprop</code>: Root mean square propagation <p>Default value: <code>adadelta</code></p>
<code>rescale_gradient</code>	<p>The rescale factor for gradient.</p> <p>Optional</p> <p>Valid values: float (min: 1e-3, max: 1.0)</p> <p>Default value: 1.0</p>

Parameter Name	Description
<code>sub_sample</code>	<p>The fraction of the training data to sample for training per epoch.</p> <p>Optional</p> <p>Valid values: Float (min: 0.0, max: 1.0)</p> <p>Default value: 1.0</p>
<code>tolerance</code>	<p>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 <code>num_patience_epochs</code> number of epochs.</p> <p>Optional</p> <p>Valid values: Float (min: 1e-6, max: 0.1)</p> <p>Default value: 0.001</p>
<code>weight_decay</code>	<p>The weight decay coefficient. Adds L2 regularization.</p> <p>Optional</p> <p>Valid values: Float (min: 0.0, max: 1.0)</p> <p>Default value: 0.0</p>

Tuning 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 it 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 \(p. 50\)](#).

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 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 \(p. 179\)](#).

Parameter Name	Parameter Type	Recommended Ranges
encoder_layers_activation	CategoricalParameterRanges	['sigmoid', 'tanh', 'relu']
learning_rate	ContinuousParameterRange	MinValue: 1e-4, MaxValue: 0.1
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

```
{
    "predictions": [
        {"topic_weights": [0.02, 0.1, 0,...]}, 
        {"topic_weights": [0.25, 0.067, 0,...]}
    ]
}
```

JSONLINES

```
{"topic_weights": [0.02, 0.1, 0,...]}  
{"topic_weights": [0.25, 0.067, 0,...]}
```

RECORDIO

```
[
```

```
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

Object2Vec 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 such that the semantics of the relationship between pairs of objects in the original space are preserved in the embedding space. The learned embeddings can be used, for example, to efficiently compute nearest neighbors of objects and to visualize natural clusters of related objects in low-dimensional space. The embeddings can also be used as features of the corresponding objects in downstream supervised tasks such as classification or regression.

Topics

- [Object2Vec Sample Notebooks \(p. 184\)](#)
- [Input/Output Interface \(p. 185\)](#)
- [EC2 Instance Recommendation \(p. 185\)](#)
- [How Object2Vec Works \(p. 185\)](#)
- [Object2Vec Hyperparameters \(p. 187\)](#)
- [Tuning an Object2Vec Model \(p. 192\)](#)
- [Data Formats for Object2Vec Training \(p. 194\)](#)
- [Data Formats for Object2Vec Inference \(p. 195\)](#)
- [Encoder Embeddings for Object2Vec \(p. 196\)](#)

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 instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Using Notebook Instances \(p. 67\)](#). 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**.

Input/Output Interface

You can use Object2Vec on many different input data types, such as sentence-sentence pairs, labels-sequence pairs, customer-customer pairs, product-product pairs, and item review user-item pairs. Natively, Object2Vec currently supports two types of input:

- *Discrete tokens*: represented as a list consisting of single integer-id, for example [10].
- *Sequences of discrete tokens*: represented as lists of integer-ids, for example [0,12,10,13].

Pre-processing is required to transform the input data into the supported formats.

The object in each pair can be asymmetric, for example, they can be (token, sequence) pairs or (token, token) pairs or (sequence, sequence) pairs. For token inputs, we support simple embeddings as compatible encoders, while for sequences of token vectors, we support average-pooled embeddings, hierarchical CNNs, as well as multi-layered BiLSTMs as encoders. The input label for each pair can be a categorical label that expresses the relationship between the objects in the pair or it can be a rating/score that expresses the strength of similarity between the two objects. For categorical labels used in classification, we support the cross-entropy loss function. For ratings/score-based labels used in regression, we support the Mean Squared Error loss function. You specify these respective loss functions with the `output_layer` hyperparameter.

EC2 Instance Recommendation

Training

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. Currently, the Object2Vec algorithm is only set up to train on a single machine. But it does however offer support for multiple GPUs.

Inference

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.

How Object2Vec Works

Step 1: Data Processing

The data must be converted to the [JSON Lines](#) text file format specified in the [Data Formats for Object2Vec Training \(p. 194\)](#) section during preprocessing. Also, to ensure obtaining the best accuracy from training, the data needs to be randomly shuffled before feeding it into the model.

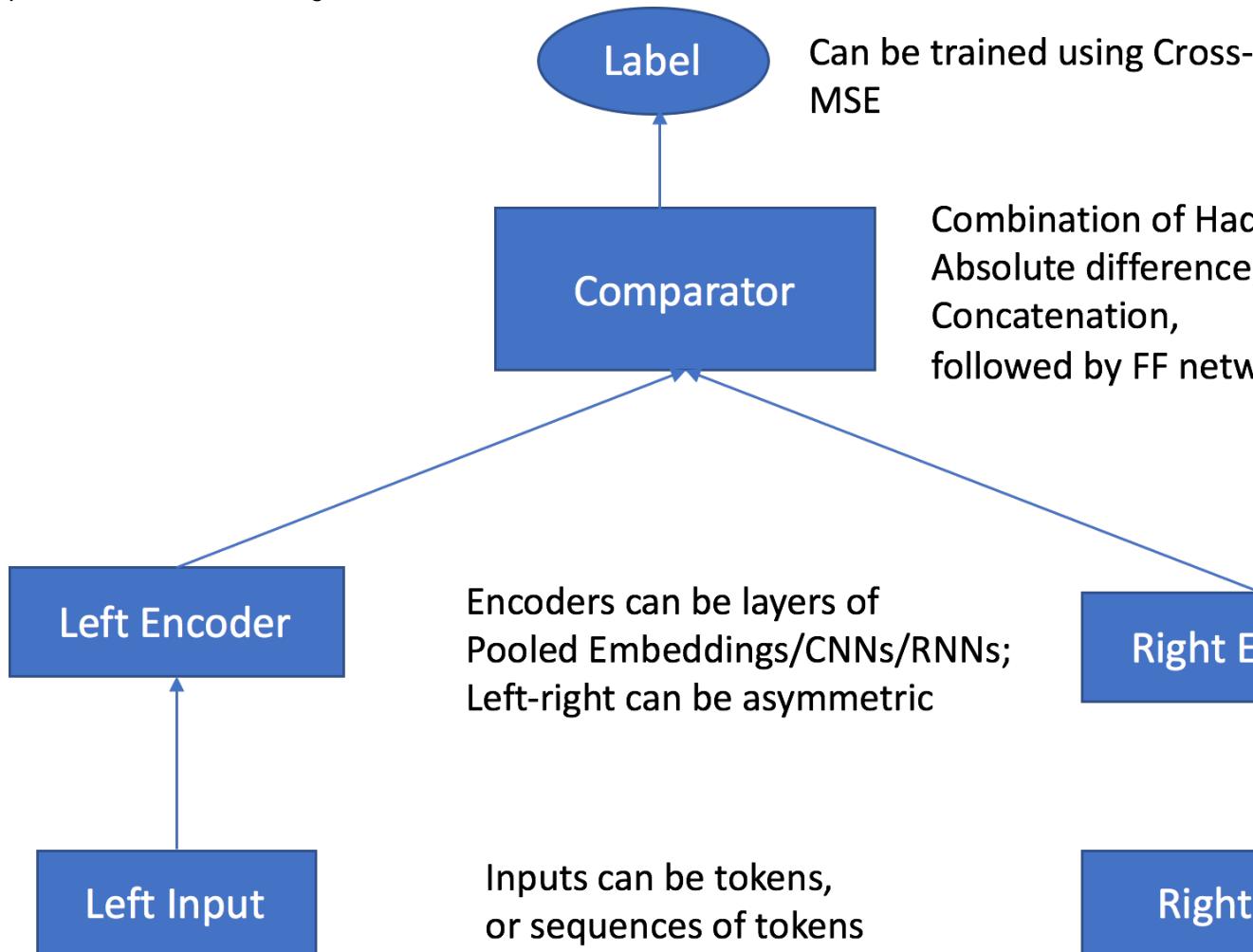
Step 2: Model Training

The architecture of SageMaker Object2Vec consists of following main components:

- *Two input channels*: The 2 input channels take a pair of objects of same or different types as inputs, and pass them to independent and customizable encoders. An example of the input objects could be sequence pairs, tokens pairs, or sequence and tokens pairs.
- *Two encoders*: The `enc0` and `enc1` encoders convert each object into a fixed-length embedding vector. The encoded embeddings of the objects in the pair, which are then passed into a comparator.
- *Comparator*: The comparator compares the embeddings in different ways and outputs scores that indicate the strength of the relationship between the paired objects for each user-specified

relationship-type. The output scores can vary between 1, indicating a strong relationship between a sentence pair, and 0, representing a weak relationship.

At training time, the algorithm accepts pairs of objects and their relationship labels or scores as inputs. The objects in each pair can be of different types. They 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 encoded embeddings of the objects in the pair are then passed into a comparator that compares the embeddings in different ways and outputs scores. The scores correspond to the strength of the relationship between the objects in the pair for each relationship-type specified by the user. The training loss function can then minimize the disagreement between the relationships predicted by the model and those specified by the user-provided labels in the training data.



General Architecture of Object2Vec: The network accepts two inputs and their relationship label at training time. Each of the inputs is passed into a compatible encoder, and the embeddings output by the encoders are passed into a comparator that emits the confidence score corresponding to the relationship label.

Step 3: Inference

After the model is trained, you can use the trained encoder to perform inference in two modes:

- 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 modes is requested based on the input data. To get the embeddings as output, you provide only one input in each instance. To predict the relationship label or score, you provide both inputs in the pair.

Object2Vec Hyperparameters

Parameter Name	Description
enc0_max_seq_len	<p>The maximum sequence length for the enc0 encoder.</p> <p>Required</p> <p>Valid values: $1 \leq \text{integer} \leq 5000$</p>
enc0_vocab_size	<p>The vocabulary size of enc0 tokens.</p> <p>Required</p> <p>Valid values: $2 \leq \text{integer} \leq 3000000$</p>
bucket_width	<p>The allowed difference between data sequence length when bucketing is enabled. Bucketing is enabled when a non-zero value is specified for this parameter.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{integer} \leq 100$</p> <p>Default value: 0 (no bucketing)</p>
dropout	<p>The dropout probability for network layers. Dropout is a form of regularization used in neural networks that reduces overfitting by trimming codependent neurons.</p> <p>Optional</p> <p>Valid values: $0.0 \leq \text{float} \leq 1.0$</p> <p>Default value: 0.0</p>
early_stopping_patience	<p>The number of consecutive epochs without improvement allowed before early stopping is applied. Improvement is defined by the <code>early_stopping_tolerance</code>.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{integer} \leq 5$</p> <p>Default value: 3</p>
early_stopping_tolerance	<p>The reduction in the loss function that an algorithm must achieve between consecutive epochs to avoid early stopping after an <code>early_stopping_patience</code> number of consecutive epochs.</p> <p>Optional</p>

Parameter Name	Description
	<p>Valid values: $0.000001 \leq \text{float} \leq 0.1$</p> <p>Default value: 0.01</p>
enc_dim	<p>The dimension of the output of the embedding layer.</p> <p>Optional</p> <p>Valid values: $4 \leq \text{integer} \leq 10000$</p> <p>Default value: 4096</p>
enc0_network	<p>Network model for the enc0 encoder.</p> <p>Optional</p> <p>Valid values: hcnn, bilstm, or pooled_embedding</p> <ul style="list-style-type: none"> • hcnn: A heterogeneous convolutional neural network • bilstm: A bidirectional long short-term memory network (LSTM), in which the signal propagates backward as well as forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks. • pooled_embedding: Averages the embeddings of all the tokens in the input. <p>Default value: hcnn</p>
enc0_cnn_filter_width	<p>The filter width of the convolutional neural network (CNN) enc0 encoder.</p> <p>Conditional</p> <p>Valid values: $1 \leq \text{integer} \leq 9$</p> <p>Default value: 3</p>
enc0_freeze_pretrained_embedding	<p>Whether to freeze enc0 pretrained embedding weights.</p> <p>Conditional</p> <p>Valid values: True or False</p> <p>Default value: True</p>
enc0_layers	<p>The number of layers in the enc0 encoder.</p> <p>Conditional</p> <p>Valid values: auto or $1 \leq \text{integer} \leq 4$</p> <p>Default value: auto</p>

Parameter Name	Description
enc0_pretrained_embedding_file	<p>The filename of pretrained enc0 token embedding file in the auxiliary data channel.</p> <p>Conditional</p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._]</p> <p>Default value: "" (empty string)</p>
enc0_token_embedding_dim	<p>The output dimension of the enc0 token embedding layer.</p> <p>Conditional</p> <p>Valid values: $2 \leq \text{integer} \leq 1000$</p> <p>Default value: 300</p>
enc0_vocab_file	<p>The vocabulary file for mapping pretrained enc0 token embeddings to vocabulary IDs.</p> <p>Conditional</p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._]</p> <p>Default value: "" (empty string)</p>
enc1_network	<p>The network model for the enc1 encoder. If its value is set to enc0, then enc1 uses the same network model as enc0, including the hyperparameter values. Note that although the enc0 and enc1 encoder networks may have symmetric architecture, shared parameter values for these networks is not supported.</p> <p>Optional</p> <p>Valid values: hcnn, bilstm, or pooled_embedding</p> <ul style="list-style-type: none"> enc0: Network model for the enc0 encoder. hcnn: A heterogeneous convolutional neural network bilstm: A bidirectional LSTM, in which the signal propagates backward as well as forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks. pooled_embedding: Averages the embeddings of all the tokens in the input. <p>Default value: enc0</p>
enc1_cnn_filter_width	<p>The filter width of the CNN enc1 encoder.</p> <p>Conditional</p> <p>Valid values: $1 \leq \text{integer} \leq 9$</p> <p>Default value: 3</p>

Parameter Name	Description
enc1_freeze_pretrained_embedding	Whether to freeze enc1 pretrained embedding weights. Conditional Valid values: <code>True</code> or <code>False</code> Default value: <code>True</code>
enc1_layers	The number of layers in the enc1 encoder. Conditional Valid values: <code>auto</code> or $1 \leq \text{integer} \leq 4$ Default value: <code>auto</code>
enc1_max_seq_len	The maximum sequence length for the enc1 encoder. Conditional Valid values: $1 \leq \text{integer} \leq 5000$
enc1_pretrained_embedding_file	filename of the enc1 pretrained token embedding file in the auxiliary data channel. Conditional Valid values: String with alphanumeric characters, underscore, or period. <code>[A-Za-z0-9\._]</code> Default value: "" (empty string)
enc1_token_embedding_dim	The output dimension of enc1 token embedding layer. Conditional Valid values: $2 \leq \text{integer} \leq 1000$ 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. <code>[A-Za-z0-9\._]</code> Default value: "" (empty string)
enc1_vocab_size	The vocabulary size of enc0 tokens. Conditional Valid values: $2 \leq \text{integer} \leq 3000000$

Parameter Name	Description
epochs	<p>The number of epochs to run for training.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{integer} \leq 100$</p> <p>Default value: 30</p>
learning_rate	<p>The learning rate for training.</p> <p>Optional</p> <p>Valid values: $1.0e-6 \leq \text{float} \leq 1.0$</p> <p>Default value: 0.0004</p>
mini_batch_size	<p>The batch size that the data set is split into for an optimizer during training.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{integer} \leq 10000$</p> <p>Default value: 32</p>
mlp_activation	<p>The type of activation function for the multilayer perceptron (MLP) layer.</p> <p>Optional</p> <p>Valid values: <code>tanh</code>, <code>relu</code>, or <code>linear</code></p> <ul style="list-style-type: none"> • <code>tanh</code>: Hyperbolic tangent • <code>relu</code>: Rectified linear unit (ReLU) • <code>linear</code>: Linear function <p>Default value: <code>linear</code></p>
mlp_dim	<p>The dimension of the output from multilayer perceptron (MLP) layers.</p> <p>Optional</p> <p>Valid values: $2 \leq \text{integer} \leq 10000$</p> <p>Default value: 512</p>
mlp_layers	<p>The number of multilayer perceptron (MLP) layers in the network.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{integer} \leq 10$</p> <p>Default value: 2</p>

Parameter Name	Description
<code>num_classes</code>	<p>The number of classes for classification training. Ignored for regression problems.</p> <p>Optional</p> <p>Valid values: $2 \leq \text{integer} \leq 30$</p> <p>Default value: 2</p>
<code>optimizer</code>	<p>The optimizer type.</p> <p>Optional</p> <p>Valid values: One of <code>adadelta</code>, <code>adagrad</code>, <code>adam</code>, <code>sgd</code>, or <code>rmsprop</code>.</p> <ul style="list-style-type: none"> • <code>adadelta</code>: A per-dimension learning rate method for gradient descent • <code>adagrad</code>: Adaptive gradient algorithm • <code>adam</code>: Adaptive moment estimation algorithm • <code>sgd</code>: Stochastic gradient descent • <code>rmsprop</code>: Root mean square propagation <p>Default value: <code>sgd</code></p>
<code>output_layer</code>	<p>The type of output layer.</p> <p>Optional</p> <p>Valid values: <code>softmax</code> or <code>mean_squared_error</code>:</p> <ul style="list-style-type: none"> • <code>softmax</code>: The Softmax function used for classification. • <code>mean_squared_error</code>: The MSE used for regression. <p>Default value: <code>softmax</code></p>
<code>weight_decay</code>	<p>The weight decay parameter used for optimization.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{float} \leq 10000$</p> <p>Default value: 0</p>

Tuning 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. You choose the objective metric from 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 \(p. 50\)](#).

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
<code>test:mean_squared_error</code>	Root Mean Square Error	Minimize
<code>validation:mean_squared_error</code>	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
<code>test:accuracy</code>	Accuracy	Maximize
<code>test:cross_entropy</code>	Cross-entropy	Minimize
<code>validation:accuracy</code>	Accuracy	Maximize
<code>validation:cross_entropy</code>	Cross-entropy	Minimize

Tunable Hyperparameters

You can tune the following hyperparameters for the Object2Vec algorithm.

Parameter Name	Parameter Type	Recommended Ranges
<code>dropout</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0
<code>early_stopping_patience</code>	IntegerParameterRange	MinValue: 1, MaxValue: 5
<code>early_stopping_tolerance</code>	ContinuousParameterRange	MinValue: 0.001, MaxValue: 0.1
<code>enc_dim</code>	IntegerParameterRange	MinValue: 4, MaxValue: 4096
<code>enc0_cnn_filter_width</code>	IntegerParameterRange	MinValue: 1, MaxValue: 5

Parameter Name	Parameter Type	Recommended Ranges	
enc0_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4	
enc0_token_embedd	IntegerParameterRange	MinValue: 5, MaxValue: 300	
enc1_cnn_filter_w	IntegerParameterRange	MinValue: 1, MaxValue: 5	
enc1_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4	
enc1_token_embedd	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: JSONLINES

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

INPUT: CLASSIFICATION OR REGRESSION

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.

OUTPUT: CLASSIFICATION OR REGRESSION

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 correspond to each of the labels.

Encoder Embeddings for Object2Vec

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 these formats, you specify only one input type, either “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 hyperparameters, which 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 \(p. 197\)](#)
- [EC2 Instance Recommendation \(p. 199\)](#)
- [Object Detection Sample Notebooks \(p. 199\)](#)
- [How Object Detection Works \(p. 199\)](#)
- [Object Detection Hyperparameters \(p. 199\)](#)
- [Tuning an Object Detection Model \(p. 203\)](#)
- [Object Detection Request and Response Formats \(p. 204\)](#)

Input/Output Interface

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. The algorithm supports only image format for inference. 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.

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 example notebooks for more details on data formats.

Training 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 \(p. 371\)](#) 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 \(COCO\)](#).

Training with 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 \(p. 371\)](#) 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

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 build-in object detection model trained in Amazon SageMaker.

To use a pretrained model, in the [CreateTrainingJob \(p. 371\)](#) 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 \(p. 7\)](#).

EC2 Instance Recommendation

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 [Using Notebook Instances \(p. 67\)](#). 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.

Object Detection Hyperparameters

In the [CreateTrainingJob \(p. 371\)](#) 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 \(p. 199\)](#).

Parameter Name	Description
<code>num_classes</code>	<p>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.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>num_training_samples</code>	<p>The number of training examples in the input dataset.</p> <p>Note If there is a mismatch between this value and the number of samples in the training set, then the behavior of the <code>lr_scheduler_step</code> parameter will be undefined and distributed training accuracy may be affected.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>base_network</code>	<p>The base network architecture to use.</p> <p>Optional</p> <p>Valid values: 'vgg-16' or 'resnet-50'</p> <p>Default value: 'vgg-16'</p>
<code>image_shape</code>	<p>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.</p> <p>Optional</p> <p>Valid values: positive integer ≥ 300</p> <p>Default: 300</p>
<code>epochs</code>	<p>The number of training epochs.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default: 30</p>
<code>freeze_layer_pattern</code>	<p>The regular expression (regex) for freezing layers in the base network. For example, if we set <code>freeze_layer_pattern = "^(conv1_ conv2_).*</code>", 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 here and here.</p> <p>Optional</p> <p>Valid values: string</p>

Parameter Name	Description
	Default: No layers frozen.
<code>kv_store</code>	<p>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.</p> <p>Note This parameter is not applicable to single machine training.</p> <p>Optional</p> <p>Valid values: 'dist_sync' or 'dist_async'</p> <ul style="list-style-type: none"> '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. <p>Default: -</p>
<code>label_width</code>	<p>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 <code>label_width</code> should be no smaller than $(10*5 + \text{header information length})$. The header information length is usually 2. We recommend using a slightly larger <code>label_width</code> for the training, such as 60 for this example.</p> <p>Optional</p> <p>Valid values: Positive integer large enough to accommodate the largest annotation information length in the data.</p> <p>Default: 350</p>
<code>learning_rate</code>	<p>The initial learning rate.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.001</p>

Parameter Name	Description
<code>lr_scheduler_factor</code>	<p>The ratio to reduce learning rate. Used in conjunction with the <code>lr_scheduler_step</code> parameter defined as $lr_{new} = lr_{old} * lr_scheduler_factor$.</p> <p>Optional</p> <p>Valid values: float in (0, 1)</p> <p>Default: 0.1</p>
<code>lr_scheduler_step</code>	<p>The epochs at which to reduce the learning rate. The learning rate is reduced by <code>lr_scheduler_factor</code> at epochs listed in a comma-delimited string: "epoch1, epoch2, ...". For example, if the value is set to "10, 20" and the <code>lr_scheduler_factor</code> is set to 1/2, then the learning rate is halved after 10th epoch and then halved again after 20th epoch.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default: -</p>
<code>mini_batch_size</code>	<p>The batch size for training. In a single-machine multi-gpu setting, each gpu handles <code>mini_batch_size/num_gpu</code> training samples. For the multi-machine training in <code>dist_sync</code> mode, the actual batch size is <code>mini_batch_size*number of machines</code>. A large <code>mini_batch_size</code> usually leads to faster training, but it may cause out of memory problem. The memory usage is related to <code>mini_batch_size</code>, <code>image_shape</code>, and <code>base_network</code> architecture. For example, on a single p3.2xlarge instance, the largest <code>mini_batch_size</code> without an out of memory error is 32 with the <code>base_network</code> set to "resnet-50" and an <code>image_shape</code> of 300. With the same instance, you can use 64 as the <code>mini_batch_size</code> with the base network vgg-16 and an <code>image_shape</code> of 300.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default: 32</p>
<code>momentum</code>	<p>The momentum for sgd. Ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.9</p>
<code>nms_threshold</code>	<p>The non-maximum suppression threshold.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.45</p>

Parameter Name	Description
optimizer	<p>The optimizer types. For details on optimizer values, see MXnet's API.</p> <p>Optional</p> <p>Valid values: ['sgd', 'adam', 'rmsprop', 'adadelta']</p> <p>Default: 'sgd'</p>
overlap_threshold	<p>The evaluation overlap threshold.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.5</p>
use_pretrained_model	<p>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.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default: 1</p>
weight_decay	<p>The weight decay coefficient for sgd and rmsprop. Ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float in (0, 1)</p> <p>Default: 0.0005</p>

Tuning 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 [Automatic Model Tuning \(p. 50\)](#).

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 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
<code>learning_rate</code>	ContinuousParameterRange	MinValue: 1e-6, MaxValue: 0.5
<code>mini_batch_size</code>	IntegerParameterRanges	MinValue: 8, MaxValue: 64
<code>momentum</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.999
<code>optimizer</code>	CategoricalParameterRanges	[<code>'sgd'</code> , <code>'adam'</code> , <code>'rmsprop'</code> , <code>'adadelta'</code>]
<code>weight_decay</code>	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,
     0.934526681900244],
    [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 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 [object detection sample notebooks](#).

OUTPUT: JSON

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)

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 \(p. 206\)](#)
- [EC2 Instance Recommendation \(p. 206\)](#)
- [Principal Component Analysis Sample Notebooks \(p. 207\)](#)
- [How PCA Works \(p. 207\)](#)
- [PCA Hyperparameters \(p. 208\)](#)
- [PCA Response Formats \(p. 209\)](#)

Input/Output Interface

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 [Principal Component Analysis Sample Notebooks \(p. 207\)](#) and the .

For more information on input and output file formats, see [PCA Response Formats \(p. 209\)](#) for inference and the [Principal Component Analysis Sample Notebooks \(p. 207\)](#).

EC2 Instance Recommendation

PCA supports both GPU and CPU computation. Which instance type is most performant depends heavily on the specifics of the input data.

Principal Component Analysis 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 [Using Notebook Instances \(p. 67\)](#). 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, \dots, x_n each of dimension $1 \times d$, 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 $1 \times d$ 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 $(\text{num_components} + \text{extra_components}) * b$ matrix that we multiply by each mini-batch, to create a $(\text{num_components} + \text{extra_components}) * d$ matrix. The sum of these matrices is computed by the workers, and the servers perform SVD on the final $(\text{num_components} + \text{extra_components}) * d$ matrix. The top right num_components singular vectors of it are the approximation of the top singular vectors of the input matrix.

Let $\ell = \text{num_components} + \text{extra_components}$. 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, \dots, 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 \(p. 207\)](#).

Parameter Name	Description
<code>feature_dim</code>	<p>Input dimension.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>mini_batch_size</code>	<p>Number of rows in a mini-batch.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>num_components</code>	<p>The number of principal components to compute.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>algorithm_mode</code>	<p>Mode for computing the principal components.</p> <p>Optional</p> <p>Valid values: <i>regular</i> or <i>randomized</i></p> <p>Default value: <i>regular</i></p>

Parameter Name	Description
<code>extra_components</code>	<p>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 <code>num_components</code>. Valid for <i>randomized</i> mode only.</p> <p>Optional</p> <p>Valid values: Non-negative integer or -1</p> <p>Default value: -1</p>
<code>subtract_mean</code>	<p>Indicates whether the data should be unbiased both during training and at inference.</p> <p>Optional</p> <p>Valid values: One of <i>true</i> or <i>false</i></p> <p>Default value: <i>true</i></p>

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

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

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

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

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 \(p. 210\)](#)
- [Instance Recommendations \(p. 211\)](#)
- [Random Cut Forest Sample Notebooks \(p. 211\)](#)
- [How RCF Works \(p. 211\)](#)
- [RCF Hyperparameters \(p. 214\)](#)
- [Tuning a RCF Model \(p. 215\)](#)
- [RCF Response Formats \(p. 216\)](#)

Input/Output Interface

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 "ShardedByS3Key" 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 [Algorithms Provided by Amazon SageMaker: Common Data Formats \(p. 82\)](#) 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 \(p. 216\)](#) for inference and the [Random Cut Forest Sample Notebooks \(p. 211\)](#).

Instance Recommendations

For training, we recommend the m1.m4, m1.c4, and m1.c5 instance families. For inference we recommend using a m1.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.

Randon Cut Forest 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 [Using Notebook Instances \(p. 67\)](#). 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].

Randomly Sampling Data

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 $S = \{S_1, \dots, S_N\}$ 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 $S = \{S_1, \dots, S_N\}$
- Initialize the random sample $X = S_1$
- For each observed sample $S_n, n = 2, \dots, N$:
 - 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 $n = 1, \dots, N$. 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].

Training and Inference

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:



Figure 1. 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.

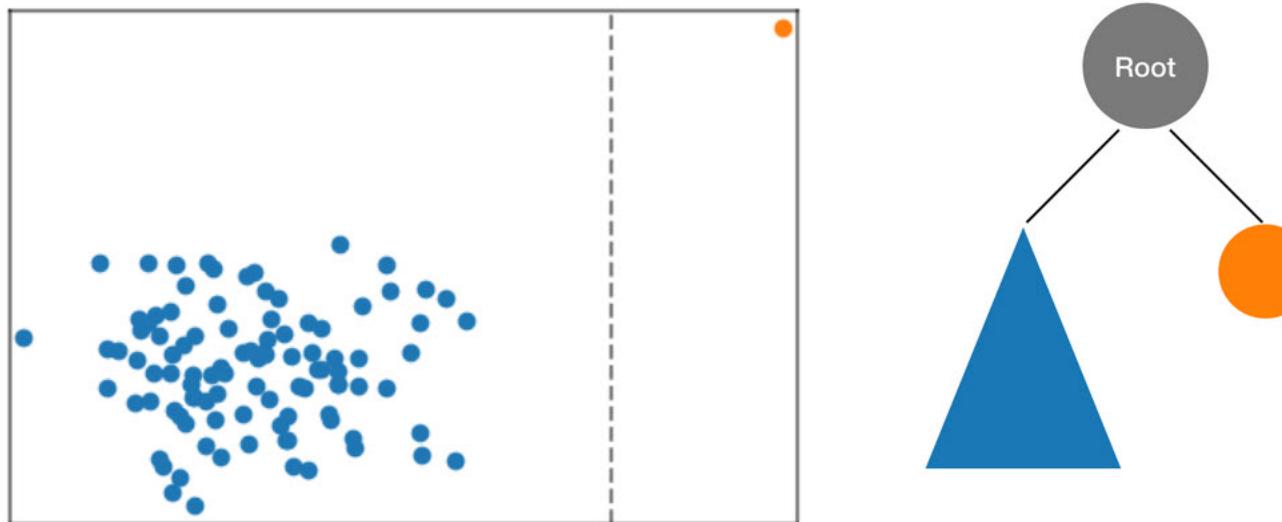


Figure 2. A random cut partitioning the two-dimensional dataset. An anomalous data point is more likely to lie isolated in a bounding box at a smaller tree depth than other points.

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.

Choosing 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 $1/\text{num_samples_per_tree}$ 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 \(p. 211\)](#).

Parameter Name	Description
<code>feature_dim</code>	<p>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.)</p> <p>Required (When the job is run through the console.)</p> <p>Valid values: Positive integer (min: 1, max: 10000)</p>
<code>eval_metrics</code>	A list of metrics used to score a labeled test data set. The following metrics can be selected for output:

Parameter Name	Description
	<ul style="list-style-type: none"> • <code>accuracy</code> - returns fraction of correct predictions. • <code>precision_recall_fscore</code> - returns the positive and negative precision, recall, and F1-scores. <p>Optional</p> <p>Valid values: a list with possible values taken from <code>accuracy</code> or <code>precision_recall_fscore</code>.</p> <p>Default value: Both <code>accuracy</code>, <code>precision_recall_fscore</code> are calculated.</p>
<code>num_samples_per_tree</code>	<p>Number of random samples given to each tree from the training data set.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 1, max: 2048)</p> <p>Default value: 256</p>
<code>num_trees</code>	<p>Number of trees in the forest.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 50, max: 1000)</p> <p>Default value: 100</p>

Tuning a 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 \(p. 50\)](#).

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 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

ACCEPT: application/json.

```
{
  "scores": [
    {"score": 0.02},
    {"score": 0.25}
  ]
}
```

JSONLINES

ACCEPT: application/jsonlines.

```
{"score": 0.02},  
{"score": 0.25}
```

RECORDIO

ACCEPT: application/x-recordio-protobuf.

```
[  
  
    Record = {  
  
        features = {},  
  
        label = {  
  
            'score': {  
  
                keys: [],  
  
                values: [0.25] # float32  
  
            }  
  
        }  
  
    },  
  
    Record = {  
  
        features = {},  
  
        label = {  
  
            'score': {  
  
                keys: [],  
  
                values: [0.02] # float32  
  
            }  
  
        }  
  
    }]
```

```
    keys: [],  
  
    values: [0.23] # float32  
  
}  
  
}  
  
}  
  
]
```

Sequence to Sequence

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 \(p. 218\)](#)
- [EC2 Instance Recommendation \(p. 219\)](#)
- [Sequence to Sequence Sample Notebooks \(p. 220\)](#)
- [How Sequence to Sequence Works \(p. 220\)](#)
- [Sequence to Sequence Hyperparameters \(p. 220\)](#)
- [Tuning a Sequence to Sequence Model \(p. 228\)](#)

Input/Output Interface

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"}
```

Please refer to the notebook for additional details on how to serialize and deserialize the inputs and outputs to specific formats for inference.

EC2 Instance Recommendation

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 [Using Notebook Instances \(p. 67\)](#). 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	<p>Mini batch size for gradient descent.</p> <p>Optional</p> <p>Valid values: positive integer</p>

Parameter Name	Description
	Default value: 64
beam_size	<p>Length of the beam for beam search. Used during training for computing <code>bleu</code> and used during inference.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
bleu_sample_size	<p>Number of instances to pick from validation dataset to decode and compute <code>bleu</code> score during training. Set to -1 to use full validation set (if <code>bleu</code> is chosen as <code>optimized_metric</code>).</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>
bucket_width	<p>Returns <code>(source,target)</code> buckets up to <code>(max_seq_len_source, max_seq_len_target)</code>. The longer side of the data uses steps of <code>bucket_width</code> 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 <code>max_len</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
bucketing_enabled	<p>Set to <code>false</code> to disable bucketing, unroll to maximum length.</p> <p>Optional</p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
checkpoint_frequency_num_batches	<p>Checkpoint and evaluate every x batches.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 1000</p>

Parameter Name	Description
checkpoint_threshold	<p>Maximum number of checkpoints model is allowed to not improve in <code>optimized_metric</code> on validation dataset before training is stopped.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
clip_gradient	<p>Clip absolute gradient values greater than this. Set to negative to disable.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>
cnn_activation_type	<p>The <code>cnn</code> activation type to be used.</p> <p>Optional</p> <p>Valid values: String. One of <code>glu</code>, <code>relu</code>, <code>softrelu</code>, <code>sigmoid</code>, or <code>tanh</code>.</p> <p>Default value: <code>glu</code></p>
cnn_hidden_dropout	<p>Dropout probability for dropout between convolutional layers.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
cnn_kernel_width_decoder	<p>Kernel width for the <code>cnn</code> decoder.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
cnn_kernel_width_encoder	<p>Kernel width for the <code>cnn</code> encoder.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>

Parameter Name	Description
cnn_num_hidden	<p>Number of <code>cnn</code> hidden units for encoder and decoder.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
decoder_type	<p>Decoder type.</p> <p>Optional</p> <p>Valid values: String. Either <code>rnn</code> or <code>cnn</code>.</p> <p>Default value: <code>rnn</code></p>
embed_dropout_source	<p>Dropout probability for source side embeddings.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
embed_dropout_target	<p>Dropout probability for target side embeddings.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
encoder_type	<p>Encoder type. The <code>rnn</code> architecture is based on attention mechanism by Bahdanau et al. and <code>cnn</code> architecture is based on Gehring et al.</p> <p>Optional</p> <p>Valid values: String. Either <code>rnn</code> or <code>cnn</code>.</p> <p>Default value: <code>rnn</code></p>
fixed_rate_lr_half_life	<p>Half life for learning rate in terms of number of checkpoints for <code>fixed_rate_*</code> schedulers.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
learning_rate	<p>Initial learning rate.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0.0003</p>

Parameter Name	Description
<code>loss_type</code>	<p>Loss function for training.</p> <p>Optional</p> <p>Valid values: String. <code>cross-entropy</code></p> <p>Default value: <code>cross-entropy</code></p>
<code>lr_scheduler_type</code>	<p>Learning rate scheduler type. <code>plateau_reduce</code> means reduce the learning rate whenever <code>optimized_metric</code> on validation_accuracy plateaus. <code>inv_t</code> is inverse time decay. $\text{learning_rate}/(1+\text{decay_rate}^t)$</p> <p>Optional</p> <p>Valid values: String. One of <code>plateau_reduce</code>, <code>fixed_rate_inv_t</code>, or <code>fixed_rate_inv_sqrt_t</code>.</p> <p>Default value: <code>plateau_reduce</code></p>
<code>max_num_batches</code>	<p>Maximum number of updates/batches to process. -1 for infinite.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: -1</p>
<code>max_num_epochs</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Positive integer and less than or equal to <code>max_num_epochs</code>.</p> <p>Default value: none</p>
<code>max_seq_len_source</code>	<p>Maximum length for the source sequence length. Sequences longer than this length are truncated to this length.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 100</p>
<code>max_seq_len_target</code>	<p>Maximum length for the target sequence length. Sequences longer than this length are truncated to this length.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 100</p>

Parameter Name	Description
<code>min_num_epochs</code>	<p>Minimum number of epochs the training must run before it is stopped via <code>early_stopping</code> conditions.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 0</p>
<code>momentum</code>	<p>Momentum constant used for <code>sgd</code>. Don't pass this parameter if you are using <code>adam</code> or <code>rmsprop</code>.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: none</p>
<code>num_embed_source</code>	<p>Embedding size for source tokens.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
<code>num_embed_target</code>	<p>Embedding size for target tokens.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
<code>num_layers_decoder</code>	<p>Number of layers for Decoder <code>rnn</code> or <code>cnn</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 1</p>
<code>num_layers_encoder</code>	<p>Number of layers for Encoder <code>rnn</code> or <code>cnn</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 1</p>
<code>optimized_metric</code>	<p>Metrics to optimize with early stopping.</p> <p>Optional</p> <p>Valid values: String. One of <code>perplexity</code>, <code>accuracy</code>, or <code>bleu</code>.</p> <p>Default value: <code>perplexity</code></p>

Parameter Name	Description
optimizer_type	<p>Optimizer to choose from.</p> <p>Optional</p> <p>Valid values: String. One of adam, sgd, or rmsprop.</p> <p>Default value: adam</p>
plateau_reduce_lr_factor	<p>Factor to multiply learning rate with (for plateau_reduce).</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0.5</p>
plateau_reduce_lr_threshold	<p>For plateau_reduce scheduler, multiply learning rate with reduce factor if optimized_metric didn't improve for this many checkpoints.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
rnn_attention_in_upper_layers	<p>Pass the attention to upper layers of rnn, like Google NMT paper. Only applicable if more than one layer is used.</p> <p>Optional</p> <p>Valid values: boolean (true or false)</p> <p>Default value: true</p>
rnn_attention_num_hidden	<p>Number of hidden units for attention layers. defaults to rnn_num_hidden.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: rnn_num_hidden</p>
rnn_attention_type	<p>Attention model for encoders. mlp refers to concat and bilinear refers to general from the Luong et al. paper.</p> <p>Optional</p> <p>Valid values: String. One of dot, fixed, mlp, or bilinear.</p> <p>Default value: mlp</p>

Parameter Name	Description
rnn_cell_type	<p>Specific type of <i>rnn</i> architecture.</p> <p>Optional</p> <p>Valid values: String. Either <code>lstm</code> or <code>gru</code>.</p> <p>Default value: <code>lstm</code></p>
rnn_decoder_state_init	<p>How to initialize <i>rnn</i> decoder states from encoders.</p> <p>Optional</p> <p>Valid values: String. One of <code>last</code>, <code>avg</code>, or <code>zero</code>.</p> <p>Default value: <code>last</code></p>
rnn_first_residual_layer	<p>First <i>rnn</i> layer to have a residual connection, only applicable if number of layers in encoder or decoder is more than 1.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 2</p>
rnn_num_hidden	<p>The number of <i>rnn</i> 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.</p> <p>Optional</p> <p>Valid values: positive even integer</p> <p>Default value: 1024</p>
rnn_residual_connections	<p>Add residual connection to stacked <i>rnn</i>. Number of layers should be more than 1.</p> <p>Optional</p> <p>Valid values: boolean (<code>true</code> or <code>false</code>)</p> <p>Default value: <code>false</code></p>
rnn_decoder_hidden_dropout	<p>Dropout probability for hidden state that combines the context with the <i>rnn</i> hidden state in the decoder.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>

Parameter Name	Description
<code>training_metric</code>	Metrics to track on training on validation data. Optional Valid values: String. Either <code>perplexity</code> or <code>accuracy</code> . Default value: <code>perplexity</code>
<code>weight_decay</code>	Weight decay constant. Optional Valid values: float Default value: 0
<code>weight_init_scale</code>	Weight initialization scale (for <code>uniform</code> and <code>xavier</code> initialization). Optional Valid values: float Default value: 2.34
<code>weight_init_type</code>	Type of weight initialization. Optional Valid values: String. Either <code>uniform</code> or <code>xavier</code> . Default value: <code>xavier</code>
<code>xavier_factor_type</code>	Xavier factor type. Optional Valid values: String. One of <code>in</code> , <code>out</code> , or <code>avg</code> . Default value: <code>in</code>

Tuning 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 \(p. 50\)](#).

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 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
<code>num_layers_encoder</code>	IntegerParameterRange	[1-10]
<code>num_layers_decoder</code>	IntegerParameterRange	[1-10]
<code>batch_size</code>	CategoricalParameterRange	[16,32,64,128,256,512,1024,2048]
<code>optimizer_type</code>	CategoricalParameterRange	['adam', 'sgd', 'rmsprop']
<code>weight_init_type</code>	CategoricalParameterRange	['xavier', 'uniform']
<code>weight_init_scale</code>	ContinuousParameterRange	For the xavier type: MinValue: 2.0, MaxValue: 3.0 For the uniform type: MinValue: -1.0, MaxValue: 1.0
<code>learning_rate</code>	ContinuousParameterRange	MinValue: 0.00005, MaxValue: 0.2
<code>weight_decay</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.1
<code>momentum</code>	ContinuousParameterRange	MinValue: 0.5, MaxValue: 0.9
<code>clip_gradient</code>	ContinuousParameterRange	MinValue: 1.0, MaxValue: 5.0
<code>rnn_num_hidden</code>	CategoricalParameterRange	Applicable only to recurrent neural

Parameter Name	Parameter Type	Recommended Ranges
		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
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_thresold	IntegerParameterRange	[1-5]
fixed_rate_lr_half_life	IntegerParameterRange	[10-30]

XGBoost Algorithm

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.

Topics

- [Input/Output Interface \(p. 231\)](#)
- [EC2 Instance Recommendation \(p. 231\)](#)
- [XGBoost Sample Notebooks \(p. 232\)](#)
- [How XGBoost Works \(p. 232\)](#)
- [XGBoost Hyperparameters \(p. 232\)](#)
- [Tuning a XGBoost Model \(p. 238\)](#)

Input/Output Interface

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.

Amazon SageMaker's 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 *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 index value pairs for features. So each row has the format: <label> <index1>:<value1> <index2>:<value2> ... 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 instance importance use Instance Weight Support

- Amazon SageMaker XGBoost allows customers to differentiate the importance of instances by assigning each instance a weight value. For *text/libsvm* input, customers can assign weight values to 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

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 the Amazon SageMaker XGBoost algorithm to train and host a regression model, see [Regression with 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 [Using Notebook Instances \(p. 67\)](#). 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 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 [Release 0.72 XGBoost Parameters](#).

Parameter Name	Description
num_class	<p>The number of classes.</p> <p>Required if <code>objective</code> is set to <code>multi:softmax</code> or <code>multi:softprob</code>.</p> <p>Valid values: integer</p>
num_round	<p>The number of rounds to run the training.</p> <p>Required</p> <p>Valid values: integer</p>
alpha	L1 regularization term on weights. Increasing this value makes models more conservative.

Parameter Name	Description
	Optional Valid values: float Default value: 1
<code>base_score</code>	The initial prediction score of all instances, global bias. Optional Valid values: float Default value: 0.5
<code>booster</code>	Which booster to use. The <code>gbtree</code> and <code>dart</code> values use a tree-based model, while <code>gblinear</code> uses a linear function. Optional Valid values: String. One of <code>gbtree</code> , <code>gblinear</code> , or <code>dart</code> . Default value: <code>gbtree</code>
<code>colsample_bylevel</code>	Subsample ratio of columns for each split, in each level. Optional Valid values: Float. Range: [0,1]. Default value: 1
<code>colsample_bytree</code>	Subsample ratio of columns when constructing each tree. Optional Valid values: Float. Range: [0,1]. Default value: 1
<code>csv_weights</code>	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
<code>early_stopping_rounds</code>	The model trains until the validation score stops improving. Validation error needs to decrease at least every <code>early_stopping_rounds</code> to continue training. Amazon SageMaker hosting uses the best model for inference. Optional Valid values: integer Default value: -

Parameter Name	Description
eta	<p>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.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 0.3</p>
eval_metric	<p>Evaluation metrics for validation data. A default metric is assigned according to the objective:</p> <p>para></p> <ul style="list-style-type: none"> • rmse: for regression • error: for classification • map: for ranking <p>For a list of valid inputs, see XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: Default according to objective.</p>
gamma	<p>Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,∞).</p> <p>Default value: 0</p>
grow_policy	<p>Controls the way that new nodes are added to the tree. Currently supported only if <code>tree_method</code> is set to <code>hist</code>.</p> <p>Optional</p> <p>Valid values: String. Either <code>depthwise</code> or <code>lossguide</code>.</p> <p>Default value: <code>depthwise</code></p>
lambda	<p>L2 regularization term on weights. Increasing this value makes models more conservative.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>

Parameter Name	Description
<code>lambda_bias</code>	<p>L2 regularization term on bias.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0</p>
<code>max_bin</code>	<p>Maximum number of discrete bins to bucket continuous features. Used only if <code>tree_method</code> is set to <code>hist</code>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 256</p>
<code>max_delta_step</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Integer. Range: [0,∞).</p> <p>Default value: 0</p>
<code>max_depth</code>	<p>Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfitted. 0 indicates no limit. A limit is required when <code>grow_policy=depth-wise</code>.</p> <p>Optional</p> <p>Valid values: Integer. Range: [0,∞)</p> <p>Default value: 6</p>
<code>max_leaves</code>	<p>Maximum number of nodes to be added. Relevant only if <code>grow_policy</code> is set to <code>lossguide</code>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>min_child_weight</code>	<p>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 <code>min_child_weight</code>, 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.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,∞).</p> <p>Default value: 1</p>

Parameter Name	Description
<code>normalize_type</code>	<p>Type of normalization algorithm.</p> <p>Optional</p> <p>Valid values: Either <i>tree</i> or <i>forest</i>.</p> <p>Default value: <i>tree</i></p>
<code>nthread</code>	<p>Number of parallel threads used to run <i>xgboost</i>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: Maximum number of threads.</p>
<code>objective</code>	<p>Specifies the learning task and the corresponding learning objective. Examples: <code>reg:linear</code>, <code>reg:logistic</code>, <code>multi:softmax</code>. For a full list of valid inputs, please refer to XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: <code>reg:linear</code></p>
<code>one_drop</code>	<p>When this flag is enabled, at least one tree is always dropped during the dropout.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>process_type</code>	<p>The type of boosting process to run.</p> <p>Optional</p> <p>Valid values: String. Either <code>default</code> or <code>update</code>.</p> <p>Default value: <code>default</code></p>
<code>rate_drop</code>	<p>The dropout rate that specifies the fraction of previous trees to drop during the dropout.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>

Parameter Name	Description
<code>refresh_leaf</code>	<p>This is a parameter of the 'refresh' updater plugin. When set to <code>true</code>(1), tree leaves and tree node stats are updated. When set to <code>false</code>(0), only tree node stats are updated.</p> <p>Optional</p> <p>Valid values: 0/1</p> <p>Default value: 1</p>
<code>sample_type</code>	<p>Type of sampling algorithm.</p> <p>Optional</p> <p>Valid values: Either <code>uniform</code> or <code>weighted</code>.</p> <p>Default value: <code>uniform</code></p>
<code>scale_pos_weight</code>	<p>Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: <code>sum(negative cases) / sum(positive cases)</code>.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>
<code>seed</code>	<p>Random number seed.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>silent</code>	<p>0 means print running messages, 1 means silent mode.</p> <p>Valid values: 0 or 1</p> <p>Optional</p> <p>Default value: 0</p>
<code>sketch_eps</code>	<p>Used only for approximate greedy algorithm. This translates into $O(1 / \text{sketch_eps})$ number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy.</p> <p>Optional</p> <p>Valid values: Float, Range: [0, 1].</p> <p>Default value: 0.03</p>

Parameter Name	Description
<code>skip_drop</code>	<p>Probability of skipping the dropout procedure during a boosting iteration.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>
<code>subsample</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 1</p>
<code>tree_method</code>	<p>The tree construction algorithm used in XGBoost.</p> <p>Optional</p> <p>Valid values: One of <code>auto</code>, <code>exact</code>, <code>approx</code>, or <code>hist</code>.</p> <p>Default value: <code>auto</code></p>
<code>tweedie_variance_power</code>	<p>Parameter that controls the variance of the Tweedie distribution.</p> <p>Optional</p> <p>Valid values: Float. Range: (1, 2).</p> <p>Default value: 1.5</p>
<code>updater</code>	<p>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.</p> <p>For a full list of valid inputs, please refer to XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: comma-separated string.</p> <p>Default value: <code>grow_colmaker</code>, <code>prune</code></p>

Tuning a 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.

For more information about model tuning, see [Automatic Model Tuning \(p. 50\)](#).

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.

Metric Name	Description	Optimization Direction
validation:auc	Area under the curve.	Maximize
validation:error	Binary classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
validation:logloss	Negative log-likelihood.	Minimize
validation:mae	Mean absolute error.	You must choose one of them as an objective to optimize when tuning the algorithm with hyperparameter values.>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 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
<code>alpha</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
<code>colsample_bylevel</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
<code>colsample_bytree</code>	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1
<code>eta</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 0.5
<code>gamma</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 5

Parameter Name	Parameter Type	Recommended Ranges
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

Using Your Own Algorithms with Amazon SageMaker

You can easily package your own algorithms for use with Amazon SageMaker, regardless of programming language or framework. Amazon SageMaker is highly flexible. It allows you to do the following:

- Use a suitable algorithm provided by Amazon SageMaker for model training and your own inference code, such as code for embedded applications, or devices, or both.
- Use your own training algorithm and inference code provided by Amazon SageMaker.
- Use your own training algorithm and your own inference code. You package the algorithm and inference code in Docker images, and use the images to train a model and deploy it with Amazon SageMaker.
- Use deep learning containers provided by Amazon SageMaker for model training and your own inference code. You provide a script written for the deep learning framework, such as Apache MXNet or TensorFlow. For more information about training, see [Using Apache MXNet with Amazon SageMaker \(p. 277\)](#) and [Using TensorFlow with Amazon SageMaker \(p. 267\)](#).

Amazon SageMaker algorithms are packaged as Docker images. This gives you the flexibility to use almost any algorithm code with Amazon SageMaker, regardless of implementation language, dependent libraries, frameworks, and so on. For more information on creating Docker images, see [The Dockerfile instructions](#).

You can provide separate Docker images for the training algorithm and inference code, or you can combine them into a single Docker image. 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.

The following sections provide detailed information about how Amazon SageMaker interacts with Docker containers and explain Amazon SageMaker requirements for Docker images. Use this information when creating your own containers. For general information about Docker containers, see [Docker Basics](#) in the [Amazon Elastic Container Service Developer Guide](#).

Topics

- [Using Your Own Training Algorithms \(p. 242\)](#)
- [Using Your Own Inference Code \(p. 246\)](#)

- [Example: Using Your Own Algorithms \(p. 251\)](#)

Using 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 \(p. 242\)](#)
- [How Amazon SageMaker Provides Training Information \(p. 243\)](#)
- [Signalling Algorithm Success and Failure \(p. 245\)](#)
- [How Amazon SageMaker Processes Training Output \(p. 245\)](#)
- [Next Step \(p. 246\)](#)

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 \(p. 371\)](#) API has a stopping condition that directs Amazon SageMaker to stop model training after a specific time.
- The [StopTrainingJob \(p. 457\)](#) 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 \(p. 371\)](#) request to Amazon SageMaker to start model training, you specify the Amazon Elastic Container Registry path of the Docker image that contain 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 \(p. 243\)](#)
- [Environment Variables \(p. 243\)](#)
- [Input Data Configuration \(p. 244\)](#)
- [Training Data \(p. 244\)](#)
- [Distributed Training Configuration \(p. 245\)](#)

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 \(p. 371\)](#) request.

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.

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.
- 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"]  
}
```

Signalling 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.

Next Step

[Using Your Own Inference Code \(p. 246\)](#)

Using 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

- [Using Your Own Inference Code \(Hosting Services\) \(p. 246\)](#)
- [Using Your Own Inference Code \(Batch Transform\) \(p. 249\)](#)

Using Your Own Inference Code (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 \(p. 246\)](#)
- [How Amazon SageMaker Loads Your Model Artifacts \(p. 248\)](#)
- [How Containers Serve Requests \(p. 248\)](#)
- [How Your Container Should Respond to Inference Requests \(p. 248\)](#)
- [How Your Container Should Respond to Health Check \(Ping\) Requests \(p. 248\)](#)

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 \(p. 349\)](#) 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 \(p. 463\)](#) 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 \(p. 461\)](#) 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 container. To stop the 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 \(p. 359\)](#) 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 \(p. 359\)](#) 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, anything else will result in failure to download the file.

Amazon SageMaker stores the model artifact as a single compressed tar file in Amazon S3. Amazon SageMaker uncompresses this tar file into the `/opt/ml/model` directory before your container starts. If you used Amazon SageMaker to train the model, the files will appear just as you left them.

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 \(p. 470\)](#) 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 consistently respond with 200s during the first 30 seconds after startup, the `CreateEndpoint` and `UpdateEndpoint` APIs will fail.

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.

Using Your Own Inference Code (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 in Batch Transform \(p. 249\)](#)
- [How Amazon SageMaker Loads Your Model Artifacts \(p. 250\)](#)
- [How Containers Serve Requests \(p. 250\)](#)
- [How Your Container Should Respond to Health Check \(Ping\) Requests \(p. 251\)](#)

How Amazon SageMaker Runs Your Inference Image in Batch Transform

To configure a container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For batch transform, 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 \(p. 359\)](#) and [CreateTransformJob \(p. 376\)](#) 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 to them then they will not be passed. At which 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 `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. 359) 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, or else it will fail to download the file.

Amazon SageMaker stores the model artifacts as a single compressed tar file in Amazon S3. Amazon SageMaker decompresses this tar file into the `/opt/ml/model` directory before your container starts. If you used Amazon SageMaker to train a model, the tar file will be stored in S3.

How Containers Serve Requests

Containers need to implement a web server that responds to `/invocations` and `/ping` on port 8080.

Containers can optionally provide dynamic information to Amazon SageMaker for better throughput. If you want to provide this information dynamically, implement `/execution-parameters` on port 8080 in addition to `/invocations` and `/ping`. The response for `GET /execution-parameters` should be a simple JSON object with keys for `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB`. Here is an example valid response:

```
{
  "MaxConcurrentTransforms": 8,
  "BatchStrategy": "MULTI_RECORD",
  "MaxPayloadInMB": 6
}
```

Amazon SageMaker will attempt invoking `/execution-parameters` before calling `/invocations`. If the container does not implement `/execution-parameters`, Amazon SageMaker applies these defaults:

Parameter	Possible Values	Default Values	Additional Notes
<code>MaxConcurrentTransforms</code>	1 to 100	1	

Parameter	Possible Values	Default Values	Additional Notes
BatchStrategy	SINGLE_RECORD, MULTI_RECORD MULTI_RECORD		
MaxPayloadInMB	20	6	0 signifies that the payload can be arbitrarily large and is transmitted in HTTP chunked encoding.

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.

Example: Using Your Own Algorithms

An example of packaging scikit-learn for use with Amazon SageMaker is available at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb

Automatically Scaling 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 3.4.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 37\)](#).

Topics

- [Automatic Scaling Components \(p. 252\)](#)
- [Before You Begin \(p. 255\)](#)
- [Related Topics \(p. 255\)](#)
- [Configure Automatic Scaling for a Variant \(p. 255\)](#)
- [Editing a Scaling Policy \(p. 261\)](#)
- [Deleting a Scaling Policy \(p. 262\)](#)
- [Load Testing for Variant Automatic Scaling \(p. 263\)](#)
- [Additional Considerations for Configuring Automatic Scaling \(p. 265\)](#)

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 \(p. 315\)](#).

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

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

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

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

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 3.4.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 37\)](#).

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 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 Variant \(Console\) \(p. 255\)](#)
- [Configure Automatic Scaling for a Variant \(AWS CLI or the Application Auto Scaling API\) \(p. 256\)](#)

Configure Automatic Scaling for a Variant (Console)

To configure automatic scaling 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 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 \(p. 263\)](#).

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 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.

Registering a 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 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 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  
}
```

Defining 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

- [Using a Predefined Metric \(p. 258\)](#)
- [Using a Custom Metric \(p. 258\)](#)
- [Adding a Cooldown Period \(p. 259\)](#)
- [Disabling Scale-in Activity \(p. 259\)](#)

Using 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"  
    }  
}
```

Using 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"  
    }  
}
```

Adding 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  
}
```

Disabling 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  
}
```

Applying 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.

Applying a Scaling Policy (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
```

Applying a Scaling Policy (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 `mscalablepolicy` to a variant named `mscalablevariant`. 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": "mscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 70.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
        }
    }
}
```

Editing a Scaling Policy

You can edit a variant scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Editing 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 Variant \(Console\) \(p. 255\)](#).

Editing 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 [Applying a Scaling Policy to a Production Variant \(p. 260\)](#).

Deleting a Scaling Policy

You can delete a scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Deleting a Scaling Policy (Console)

To delete an automatic scaling policy for a variant (console)

- Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
- In the navigation pane, choose **Endpoints**.
- Choose the endpoint for which you want to delete automatic scaling.
- For **Endpoint runtime settings**, choose the variant that you want to configure.
- Choose **Configure auto scaling**.
- Choose **Deregister auto scaling**.

Deleting 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.

Deleting 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 `mscalablepolicy` from a variant named `mscalablevariant`.

```
aws application-autoscaling delete-scaling-policy \
--policy-name mscalablepolicy \
--resource-id endpoint/MyEndpoint/variant/MyVariant \
--service-namespace sagemaker \
--scalable-dimension sagemaker:variant:DesiredInstanceCount
```

Deleting 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 `mscalablepolicy` from a variant named `mscalablevariant`.

```
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": "mscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount"
}
```

Load Testing for 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 Variant \(p. 264\)](#)
- [Calculate the Target `SageMakerVariantInvocationsPerInstance` \(p. 264\)](#)

Determine the Performance Characteristics of a 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 3.4.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 37\)](#).
2. Use a load testing tool to generate an increasing number of parallel requests, and monitor the RPS and model latency in the output 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.

Additional Considerations for Configuring Automatic Scaling

When configuring automatic scaling, consider the following general guidelines.

Test 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 \(p. 461\)](#) 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.

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

Using TensorFlow with Amazon SageMaker

You can use Amazon SageMaker to train a model using custom TensorFlow code. If you choose to deploy your code using Amazon SageMaker hosting services, you can also provide custom TensorFlow inference code. This section provides guidelines for writing custom TensorFlow code for both model training and inference, and an example that includes sample TensorFlow code and instructions for model training and deployment.

Amazon SageMaker supports using pipe mode in TensorFlow containers. For information, see <https://github.com/aws/sagemaker-tensorflow-extensions/blob/master/README.rst>

For information about TensorFlow supported versions, see <https://github.com/aws/sagemaker-python-sdk#tensorflow-sagemaker-estimators>. The container source code can be found at the GitHub repository at <https://github.com/aws/sagemaker-tensorflow-containers>.

Topics

- [Writing Custom TensorFlow Model Training and Inference Code \(p. 267\)](#)
- [Examples: Using Amazon SageMaker with TensorFlow \(p. 270\)](#)

Writing Custom TensorFlow Model Training and Inference Code

To train a model on Amazon SageMaker using custom TensorFlow code and deploy it on Amazon SageMaker, you need to implement training and inference code interfaces in your code.

Your TensorFlow training script must be a Python 2.7 source file. This training/inference script must contain the following functions:

- `model_fn`: Defines the model that will be trained.
- `train_input_fn`: Preprocess and load training data.
- `eval_input_fn`: Preprocess and load evaluation data.
- `serving_input_fn`: Defines the features to be passed to the model during prediction.

For more information, see <https://github.com/aws/sagemaker-python-sdk#tensorflow-sagemaker-estimators>.

Implement the following training code interface:

```
def model_fn(features, labels, mode, hyperparameters):
    """
    Implement code to do the following:
    1. Configure the model with TensorFlow operations
```

```

2. Define the loss function for training/evaluation
3. Define the training operation/optimizer
4. Generate predictions
5. Return predictions/loss/train_op/eval_metric_ops in EstimatorSpec object

```

For more information on how to create a model_fn, see
https://www.tensorflow.org/extend/estimators#constructing_the_model_fn.

Args:

```

    features: A dict containing the features passed to the model with train_input_fn
in
    training mode, with eval_input_fn in evaluation mode, and with serving_input_fn
    in predict mode.
    labels: Tensor containing the labels passed to the model with train_input_fn in
        training mode and eval_input_fn in evaluation mode. It is empty for
        predict mode.
    mode: One of the following tf.estimator.ModeKeys string values indicating the
context
    in which the model_fn was invoked:
        - TRAIN: The model_fn was invoked in training mode.
        - EVAL: The model_fn was invoked in evaluation mode.
        - PREDICT: The model_fn was invoked in predict mode:

```

```

hyperparameters: The hyperparameters passed to your Amazon SageMaker TrainingJob
that
    runs your TensorFlow training script. You can use this to pass hyperparameters
    to your training script.

```

Returns: An EstimatorSpec, which contains evaluation and loss function.

"""

```
def train_input_fn(training_dir, hyperparameters):
    """

```

Implement code to do the following:

1. Read the **training** dataset files located in training_dir
2. Preprocess the dataset
3. Return 1) a mapping of feature columns to Tensors with
the corresponding feature data, and 2) a Tensor containing labels

For more information on how to create a input_fn, see https://www.tensorflow.org/get_started/input_fn.

Args:

```

    training_dir: Directory where the dataset is located inside the container.
    hyperparameters: The hyperparameters passed to your Amazon SageMaker TrainingJob
that
    runs your TensorFlow training script. You can use this to pass hyperparameters
    to your training script.

```

Returns: (data, labels) tuple

"""

```
def eval_input_fn(training_dir, hyperparameters):
    """

```

Implement code to do the following:

1. Read the **evaluation** dataset files located in training_dir
2. Preprocess the dataset
3. Return 1) a mapping of feature columns to Tensors with
the corresponding feature data, and 2) a Tensor containing labels

For more information on how to create a input_fn, see https://www.tensorflow.org/get_started/input_fn.

Args:

```

    training_dir: The directory where the dataset is located inside the container.
    hyperparameters: The hyperparameters passed to your Amazon SageMaker TrainingJob that
        runs your TensorFlow training script. You can use this to pass hyperparameters
        to your training script.

    Returns: (data, labels) tuple
    """

def serving_input_fn(hyperparameters):
    """
    During training, a train_input_fn() ingests data and prepares it for use by the model.
    At the end of training, similarly, a serving_input_fn() is called to create the model
    that
        will be exported for Tensorflow Serving.

    Use this function to do the following:
        - Add placeholders to the graph that the serving system will feed with inference
        requests.
        - Add any additional operations needed to convert data from the input format into the
            feature Tensors expected by the model.

    The function returns a tf.estimator.export.ServingInputReceiver object, which packages the
    placeholders
        and the resulting feature Tensors together.

    Typically, inference requests arrive in the form of serialized tf.Examples, so the
        serving_input_receiver_fn() creates a single string placeholder to receive them. The
    serving_input_receiver_fn()
        is then also responsible for parsing the tf.Examples by adding a tf.parse_example
    operation to the graph.

    For more information on how to create a serving_input_fn, see
        https://github.com/tensorflow/tensorflow/
blob/18003982ff9c809ab8e9b76dd4c9b9ebc795f4b8/tensorflow/docs_src/programmers_guide/
saved_model.md#preparing-serving-inputs.

    Args:
        hyperparameters: The hyperparameters passed to your TensorFlow Amazon SageMaker
    estimator that
        deployed your TensorFlow inference script. You can use this to pass
    hyperparameters
        to your inference script.

    """

```

Optionally implement the following inference code interface:

```

def input_fn(data, content_type):
    """
    [Optional]
    Prepares data for transformation. Amazon SageMaker invokes your input_fn definition in
    response to
        an InvokeEndpoint operation on an Amazon SageMaker endpoint containing this script.
    Amazon SageMaker passes in the
        data in the InvokeEndpoint request, and the
        InvokeEndpoint ContentType argument.
    If you omit this function, Amazon SageMaker provides a default input_fn for you. The
    default
        input_fn supports protobuf, CSV, or JSON-encoded array data. It returns the input in the
    format expected by
        TensorFlow serving. For more information about the default input_fn, see the Amazon
    SageMaker Python SDK
        GitHub documentation.

```

```
Args:  
    data: An Amazon SageMaker InvokeEndpoint request data  
    content_type: An Amazon SageMaker InvokeEndpoint ContentType value for data.  
Returns:  
    object: A deserialized object that will be used by TensorFlow serving as input. Must  
be in the format  
        defined by the placeholders in your serving_input_fn.  
    """  
  
def output_fn(data, accepts):  
    """  
    [Optional]  
    Serializes the result of prediction in response to an InvokeEndpoint request. This  
function should return a serialized object. This serialized object is returned in the  
response to an  
InvokeEndpoint request. If you omit this function, Amazon SageMaker provides a default  
output_fn for you.  
The default function works with protobuf, CSV, and JSON accept types and serializes data  
to either,  
depending on the specified accepts.  
  
Args:  
    data: A result from TensorFlow Serving  
    accepts: The Amazon SageMaker InvokeEndpoint Accept value. The content type the  
response  
        object should be serialized to.  
Returns:  
    object: The serialized object that will be send to back to the client.
```

Examples: Using Amazon SageMaker with TensorFlow

Topics

- [TensorFlow Example 1: Using the tf.estimator \(p. 271\)](#)

Amazon SageMaker provides the following example notebooks in your Amazon SageMaker notebook instance:

- **tf.estimator** (`iris_dnn_classifier`)—This introductory example shows how to use the Amazon SageMaker `sagemaker.tensorflow.TensorFlow` estimator class. It uses this class to create a model for classifying a flower, based on four numerical features. The custom training code provided for this example uses the `DNNClassifier` to create a neural network with three hidden layers.
- **tf.layers** (`abalone_using_layers`)—This example shows how to use the TensorFlow `layers` module. It uses the module to create a model that predicts the age of a sea snail, based on seven numerical features. The custom code creates a neural network by individually specifying its layers.
- **tf.contrib.keras** (`abalone_using_keras`)—This example shows how to use the TensorFlow Keras library. It uses the library to create a model that predicts the age of a sea snail.
- **distributed TensorFlow** (`distributed_mnist`)—In this example, you explore distributed TensorFlow training in Amazon SageMaker. You use a convolutional neural network (CNN) to classify handwritten numbers and multiple GPU hosts for distributed training.

- **ResNet CIFAR-10 with Tensorboard** (`resnet_cifar10_with_tensorboard`)— In this example, you use Tensorboard with SageMaker. You use a ResNet model to train the CIFAR-10 dataset and evaluate it using TensorBoard.

In the examples, only the TensorFlow custom model training code differs. You interact with Amazon SageMaker the same way in each.

The examples use the high-level Python library provided by Amazon SageMaker. This library is available in the Amazon SageMaker notebook instance that you created in Getting Started. If you use your own terminal, download and install the library using one of the following options:

- Get it from <https://github.com/aws/sagemaker-python-sdk>.
- Use pip to install it:

```
$ pip install sagemaker
```

The following topic explains one TensorFlow example in detail. All of the TensorFlow example notebooks that Amazon SageMaker provides follow the same pattern. They differ only in the custom TensorFlow code that they use for model training.

There are two ways to use this exercise:

- Follow the steps to create, deploy, and validate the model. You create a Jupyter notebook in your Amazon SageMaker notebook instance, and copy code, paste it into the notebook, and run it.
- If you're familiar with using notebooks, open and run the example notebook that Amazon SageMaker provides in the notebook instance.

Topics

- [TensorFlow Example 1: Using the tf.estimator \(p. 271\)](#)

TensorFlow Example 1: Using the tf.estimator

This introductory TensorFlow example demonstrates how to use the Amazon SageMaker `sagemaker.tensorflow.TensorFlow` estimator class. This class provides the `fit` method for model training in Amazon SageMaker and the `deploy` method to deploy the resulting model in Amazon SageMaker. It is included in the Amazon SageMaker high-level Python library.

TensorFlow provides a high-level machine learning API (`tf.estimator`) to easily configure, train, and evaluate a variety of machine learning models. In this exercise, you construct a neural network classifier using this API. You then train it on Amazon SageMaker using the [Iris flower data set](#).

In the example code, you do the following:

1. Train the model. During training, the following occurs:
 - a. Amazon SageMaker loads the Docker image that contains the TensorFlow framework (this starts the Docker container).
 - b. Amazon SageMaker reads training data from an Amazon S3 bucket into the container's file system.
 - c. Your custom training code constructs a neural network classifier, `tf.estimator.DNNClassifier`.
 - d. The Amazon SageMaker code in the container runs training. Your training code reads the training data.
2. Deploy the model using Amazon SageMaker hosting service. Amazon SageMaker returns an endpoint that you send requests to get inferences.

3. Test the model. You send requests to the model to classify flower samples you send in the request.

About the Training Dataset

The Iris flower training dataset used in this exercise contains 150 rows of data, with 50 samples from each of the three related Iris species (Iris setosa, Iris virginica, and Iris versicolor). Each row in the dataset contains the following data for each of the flower samples:

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	0
4.7	3.2	1.3	0.2	0
...
7.0	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
...
6.5	3.0	5.2	2.0	2
6.2	3.4	5.4	2.3	2
5.9	3.0	5.1	1.8	2

For this exercise, the dataset is randomized and split into two .csv files:

- A training dataset of 120 samples (`iris_training.csv`)
- A test dataset of 30 samples (`iris_test.csv`)

Next Step

Step 1: Create a Notebook and Initialize Variables (p. 272)

Step 1: Create a Notebook and Initialize Variables

In this section, you create a Jupyter notebook in your Amazon SageMaker notebook instance and initialize variables.

1. Follow the instructions in [Step 1: Setting Up \(p. 24\)](#) to create the S3 bucket and IAM role.

For simplicity, we suggest that you use one S3 bucket to store both the training code and the results of model training.

2. Create the notebook.
 - a. If you haven't already done so, sign in to the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
 - b. Open the notebook instance by choosing Open next to its name. The Jupyter notebook server page appears.
 - c. To create a notebook, choose **Files**, **New**, and **conda_tensorflow_p36**. This pre-installed environment includes the default Anaconda installation, Python 3, and TensorFlow.
 - d. Name the notebook.
3. To initialize variables, copy, paste, and run the following code in your notebook. Provide the name of the S3 bucket that contains your custom code. The `get_execution_role` function retrieves

the IAM role you created at the time of creating your notebook instance. You can use the bucket that you created in [Step 1: Setting Up \(p. 24\)](#), or create a new bucket.

```
from sagemaker import get_execution_role

#Bucket location to save your custom code in tar.gz format.
custom_code_upload_location = 's3://yourbucket/customcode/tensorflow_iris'

#Bucket location where results of model training are saved.
model_artifacts_location = 's3://yourbucket/artifacts'

#IAM execution role that gives SageMaker access to resources in your AWS account.
role = get_execution_role()
```

Next Step

[Step 2: Train a Model on Amazon SageMaker Using TensorFlow Custom Code \(p. 273\)](#)

Step 2: Train a Model on Amazon SageMaker Using TensorFlow Custom Code

The high-level Python library provides the `TensorFlow` class, which has two methods: `fit` (for training a model) and `deploy` (for deploying a model).

To train a model

1. Create an instance of the `sagemaker.tensorflow.TensorFlow` class by copying, pasting, and running the following code:

```
from sagemaker.tensorflow import TensorFlow

iris_estimator = TensorFlow(entry_point='/home/ec2-user/sample-notebooks/sagemaker-
python-sdk/tensorflow_iris_dnn_classifier_using_estimators/iris_dnn_classifier.py',
                           role=role,
                           output_path=model_artifacts_location,
                           code_location=custom_code_upload_location,
                           train_instance_count=1,
                           train_instance_type='ml.c4.xlarge',
                           training_steps=1000,
                           evaluation_steps=100)
```

Some of these constructor parameters are sent in the `fit` method call for model training in the next step.

Details:

- `entry_point`—The example uses only one source file (`iris_dnn_classifier.py`) and it is already provided for you on your notebook instance. If your custom training code is stored in a single file, specify only the `entry_point` parameter. If it's stored in multiple files, also add the `source_dir` parameter.

Note

Specify only the source file that contains your custom code. The `sagemaker.tensorflow.TensorFlow` object determines which Docker image to use for model training when you call the `fit` method in the next step.

- `output_path`—Identifies the S3 location where you want to save the result of model training (model artifacts).

- `code_location`—S3 location where you want the `fit` method (in the next step) to upload the tar archive of your custom TensorFlow code.
- `role`—Identifies the IAM role that Amazon SageMaker assumes when performing tasks on your behalf, such as downloading training data from an S3 bucket for model training and uploading training results to an S3 bucket.
- `hyperparameters` - Any hyperparameters that you specify to influence the final quality of the model. Your custom training code uses these parameters.
- `train_instance_type` and `train_instance_count`—Identify the type and number of ML Compute instances to launch for model training.

You can also train your model on your local computer by specifying `local` as the value for `train_instance_type` and `1` as the value for `train_instance_count`. For more information about local mode, see <https://github.com/aws/sagemaker-python-sdk#local-mode> in the *Amazon SageMaker Python SDK*.

2. Start model training by copying, pasting, and running the following code:

```
%%time
import boto3

region = boto3.Session().region_name
train_data_location = 's3://sagemaker-sample-data-{} tensorflow/iris'.format(region)

iris_estimator.fit(train_data_location)
```

The `fit` method parameter identifies the S3 location where the training data is stored. The `fit` method sends a [CreateTrainingJob \(p. 371\)](#) request to Amazon SageMaker.

You can get the training job information by calling the [DescribeTrainingJob \(p. 410\)](#) or viewing it in the console. The following is an example response:

```
{
    "InputDataConfig": [
        {
            "ChannelName": "training",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3DataDistributionType": "FullyReplicated",
                    "S3Uri": "s3://sagemaker-sample-data-us-west-2/tensorflow/iris"
                }
            }
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://sagemakermv/artifacts"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 86400
    },
    "TrainingJobName": "sagemaker-tensorflow-py2-cpu-2017-11-18-03-11-11-686",
    "AlgorithmSpecification": {
        "TrainingInputMode": "File",
        "TrainingImage": "142577830533.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
tensorflow-py2-cpu:1.0.5"
    },
    "HyperParameters": {
        "sagemaker_program": "\"iris_dnn_classifier.py\"",
        "checkpoint_path": "\"s3://sagemakermv/artifacts/checkpoints\"",
        "sagemaker_job_name": "\"sagemaker-tensorflow-py2-cpu-2017-11-18-03-11-11-686\""
    }
}
```

```
    "sagemaker_submit_directory": "\"s3://sagemakermv/customcode/tensorflow_iris/sagemaker-tensorflow-py2-cpu-2017-11-18-03-11-686/sourcedir.tar.gz\"",
        "sagemaker_region": "\"us-west-2\"",
        "training_steps": "100",
        "sagemaker_container_log_level": "20"
    },
    "ResourceConfig": {
        "VolumeSizeInGB": 30,
        "InstanceCount": 1,
        "InstanceType": "ml.c4.xlarge"
    },
    "RoleArn": "arn:aws:iam::account-id:role/SageMakerRole"
}
```

Details:

- **TrainingImage**—Amazon SageMaker runs this image to create a container for model training. You don't explicitly identify this image in your request. The `fit` method dynamically chooses the correct image by inspecting the Python version in the interpreter and the GPU capability of the ML compute instance type that you specified when creating the TensorFlow object.
- **Hyperparameters**—The request includes the hyperparameters that you specified when you created the `sagemaker.tensorflow.TensorFlow` object. It also includes the following additional hyperparameters, which have the prefix `sagemaker`. Amazon SageMaker uses these hyperparameters to set up the training environment.
 - `sagemaker_submit_directory`—Identifies the S3 location of the custom training code.

The high-level Python library does several things for you. In this case, the `fit` method creates a gzipped tar archive from the custom code file(s), and uploads the archive to an S3 bucket. You specify this archive in this hyperparameter.

- `sagemaker_program`—Identifies the primary module that your training functions will be loaded from. This is the `entry_point` parameter that you specified when you created the `sagemaker.tensorflow.TensorFlow` object.
- `sagemaker_container_log_level`—Sets the Python logging level.
- `sagemaker_job_name`—Amazon SageMaker uses the job name to publish CloudWatch metrics in your account.
- `sagemaker_checkpoint_path`—In distributed training, TensorFlow uses this S3 location as a shared file system for the ML compute instances running the training.
- `InputDataConfig`—Specifies one channel. A channel is a named input source that the training code consumes.
- `OutputDataConfig`—Identifies the S3 location where you want to save training results (model artifacts).

By default, the training job runs synchronously (you see the output in the notebook). If you want it to run asynchronously, set the `wait` value to `false` in the call to the `fit` method or when you create the `sagemaker.tensorflow.TensorFlow` instance.

Next Step

[Step 3: Deploy the Trained Model \(p. 275\)](#)

Step 3: Deploy the Trained Model

To deploy the model, use Amazon SageMaker hosting services. During deployment, Amazon SageMaker launches the ML compute instances and deploys the model (the model artifacts and inference code) on them. In response, you get an endpoint. To get inferences from the model, your application sends requests to the endpoint.

For fast deployment, use the `deploy` method in the `sagemaker.tensorflow.TensorFlow` class. The class is included in the high-level Python library provided by Amazon SageMaker. The `deploy` method does the following:

1. Creates an Amazon SageMaker model by calling the [CreateModel \(p. 359\)](#) API. The model contains important information, such as the location of the model artifacts and the inference code image.
2. Creates an endpoint configuration by calling the [CreateEndpointConfig \(p. 352\)](#) API. This configuration specifies the name of the model (which was created in the preceding step), and the resource configuration (the type and number of ML compute instances to launch for hosting).
3. Creates the endpoint by calling the [CreateEndpoint \(p. 349\)](#) API and specifying the endpoint configuration. Amazon SageMaker launches ML compute instances as specified in the endpoint configuration, and deploys the model on them.

To deploy the model, copy, paste, and run the following code:

```
%time  
  
iris_predictor = iris_estimator.deploy(initial_instance_count=1,  
                                       instance_type='ml.m4.xlarge')
```

When the status of the endpoint is `INSERVICE`, your model had been deployed. The API returns a `TensorFlowPredictor` object. To get inferences, you will use the `predict` method of this object.

Note

You can deploy your model to an endpoint hosted on your local computer by specifying `local` as the value for `train_instance_type` and `1` as the value for `train_instance_count`. For more information about local mode, see <https://github.com/aws/sagemaker-python-sdk#local-mode> in the *Amazon SageMaker Python SDK*.

Next Step

[Step 4: Invoke the Endpoint to Get Inferences \(p. 276\)](#)

Step 4: Invoke the Endpoint to Get Inferences

To get inferences, send requests using the `predict` method of the `TensorFlowPredictor` object. (This object was returned in Step 3). The `predict` method calls the Amazon SageMaker [InvokeEndpoint \(p. 470\)](#).

Run the `predict` method:

```
iris_predictor.predict([6.4, 3.2, 4.5, 1.5]) #expected label to be 1
```

For more information about the input features, see [About the Training Dataset \(p. 272\)](#).

The model predicts that the flower species is Iris versicolor (label 1) with a probability of 97%:

```
{u'result': {u'classifications': [{u'classes': [{u'label': u'0',  
          u'score': 0.009605729021131992},  
          {u'label': u'1', u'score': 0.9699361324310303},  
          {u'label': u'2', u'score': 0.02045818418264389}]}]}}
```

Your Amazon SageMaker notebook instance includes additional examples.

Using Apache MXNet with Amazon SageMaker

You can use Amazon SageMaker to train a model using your own custom Apache MXNet training code. If you choose to use Amazon SageMaker hosting services, you can also provide your own custom Apache MXNet inference code. This section provides guidelines for writing custom Apache MXNet code for both model training and inference and an example that includes sample Apache MXNet code and instructions for model training and deployment.

For information about Apache MXNet supported versions, see <https://github.com/aws/sagemaker-python-sdk#mxnet-sagemaker-estimators>. The container source code can be found at the GitHub repository at <https://github.com/aws/sagemaker-mxnet-containers>.

Topics

- [Writing Custom Apache MXNet Model Training and Inference Code \(p. 277\)](#)
- [Examples: Using Amazon SageMaker with Apache MXNet \(p. 283\)](#)

Writing Custom Apache MXNet Model Training and Inference Code

To train a model on Amazon SageMaker using custom Apache MXNet code and deploy it on Amazon SageMaker, your code must implement the following training code interface and inference code interface.

- Implement the following training code interface:

```
# ----- #  
# Training functions #  
# ----- #  
  
def train(  
    hyperparameters,  
    input_data_config,  
    channel_input_dirs,  
    output_data_dir,  
    model_dir,  
    num_gpus,  
    num_cpus,  
    hosts,  
    current_host,  
    **kwargs):  
  
    """  
    [Required]  
  
    Runs Apache MXNet training. Amazon SageMaker calls this function with information  
    about the training environment. When called, if this function returns an  
    object, that object is passed to a save function. The save function
```

can be used to serialize the model to the Amazon SageMaker training job model directory.

The `**kwargs` parameter can be used to absorb any Amazon SageMaker parameters that your training job doesn't need to use. For example, if your training job doesn't need to know anything about the training environment, your function signature can be as simple as `train(**kwargs)`.

Amazon SageMaker invokes your `train` function with the following python `kwargs`:

Args:

- `hyperparameters`: The Amazon SageMaker Hyperparameters dictionary. A dict of string to string.
- `input_data_config`: The Amazon SageMaker input channel configuration for this job.
- `channel_input_dirs`: A dict of string-to-string maps from the Amazon SageMaker algorithm input channel name to the directory containing files for that input channel. Note, if the Amazon SageMaker training job is run in PIPE mode, this dictionary will be empty.
- `output_data_dir`:
The Amazon SageMaker output data directory. After the function returns, data written to this directory is made available in the Amazon SageMaker training job output location.
- `model_dir`: The Amazon SageMaker model directory. After the function returns, data written to this directory is made available to the Amazon SageMaker training job model location.
- `num_gpus`: The number of GPU devices available on the host this script is being executed on.
- `num_cpus`: The number of CPU devices available on the host this script is being executed on.
- `hosts`: A list of hostnames in the Amazon SageMaker training job cluster.
- `current_host`: This host's name. It will exist in the hosts list.
- `kwargs`: Other keyword args.

Returns:

- `(object)`: Optional. An Apache MXNet model to be passed to the model save function. If you do not return anything (or return `None`), the save function is not called.

"""

pass

```
def save(model, model_dir):
    """
    [Optional]
```

Saves an Apache MXNet model after training. This function is called with the return value of `train`, if there is one. You are free to implement this to perform your own saving operation.

Amazon SageMaker provides a default save function for Apache MXNet models. The default save function serializes 'Apache MXNet Module <<https://mxnet.incubator.apache.org/api/python/module.html>>' models. To rely on the default save function, omit a definition of 'save' from your script. The default save function is discussed in more detail in the Amazon SageMaker Python SDK GitHub documentation.

If you are using the Gluon API, you should provide your own save function, or save your model in the `train` function and let the `train` function complete without returning anything.

Arguments:

- `model (object)`: The return value from `train`.
- `model_dir`: The Amazon SageMaker model directory.

"""

```
    pass
```

- Implement the following inference code interface:

```
# -----
# Hosting functions
# -----



def model_fn(model_dir):
    """
    [Optional]

    Loads a model from disk, reading from model_dir. Called once by each
    inference service worker when it is started.

    If you want to take advantage of Amazon SageMaker's default request handling
    functions, the returned object should be a `Gluon Block
    <https://mxnet.incubator.apache.org/api/python/gluon/gluon.html#mxnet.gluon.Block>
    or MXNet `Module <https://mxnet.incubator.apache.org/api/python/module.html>`,
    described below. If you are providing your own transform_fn,
    then your model_fn can return anything that is compatible with your
    transform_fn.

    Amazon SageMaker provides a default model_fn that works with the serialization
    format used by the Amazon SageMaker default save function, discussed above. If
    you saved your model using the default save function, you do not need to
    provide a model_fn in your hosting script.

    Args:
        - model_dir: The Amazon SageMaker model directory.

    Returns:
        - (object): Optional. The deserialized model.
    """
    pass


def transform_fn(model, input_data, content_type, accept):
    """
    [Optional]

    Transforms input data into a prediction result. Amazon SageMaker invokes your
    transform_fn in response to an InvokeEndpoint operation on an Amazon SageMaker
    endpoint containing this script. Amazon SageMaker passes in the model, previously
    loaded with model_fn, along with the input data, request content type, and accept
    content type from the InvokeEndpoint request.

    The input data is expected to have the given content_type.

    The output returned should have the given accept content type.

    This function should return a tuple of (transformation result, content
    type). In most cases, the returned content type should be the same as the
    accept content type, but it might differ. For example, when your code needs to
    return an error response.

    If you provide a transform_fn in your hosting script, it will be used
    to handle the entire request. You don't need to provide any other
    request handling functions (input_fn, predict_fn, or output_fn).
    If you do provide them, they will be ignored.

    Amazon SageMaker provides default transform_fn implementations that work with
    Gluon and Module models. These support JSON input and output, and for Module
```

```

models, also CSV. To use the default transform_fn, provide a
hosting script without a transform_fn or any other request handling
functions. For more information about the default transform_fn,
see the SageMaker Python SDK GitHub documentation.

Args:
- input_data: The input data from the payload of the
  InvokeEndpoint request.
- content_type: The content type of the request.
- accept: The content type from the request's Accept header.

Returns:
- (object, string): A tuple containing the transformed result
  and its content type
"""
pass

# ----- #
# Request handlers for Gluon models
# ----- #

def input_fn(input_data, content_type):
    """
    [Optional]

    Prepares data for transformation. Amazon SageMaker invokes your input_fn in
    response to an InvokeEndpoint operation on an Amazon SageMaker endpoint that contains
    this script. Amazon SageMaker passes in input data and content type from the
    InvokeEndpoint request.

    The function should return an NDArray that can be passed to the
    predict_fn.

    If you omit this function, Amazon SageMaker provides a default implementation.
    The default input_fn converts JSON-encoded array data into an NDArray.
    For more information about the default input_fn, see the SageMaker
    Python SDK GitHub documentation.

    Args:
        - input_data: The input data from the payload of the
          InvokeEndpoint request.
        - content_type: The content type of the request.

    Returns:
        - (NDArray): An NDArray
"""
pass

def predict_fn(block, array):
    """
    [Optional]

    Performs prediction on an NDArray object. In response to an InvokeEndpoint request,
    Amazon SageMaker invokes your
    predict_fn with the model returned by your model_fn and the result
    of the input_fn.

    The function should return an NDArray containing the prediction results.

    If you omit this function, Amazon SageMaker provides a default implementation.
    The default predict_fn call passes the array to the forward
    method of the Gluon block, for example, block(array).

```

```

Args:
- block (Block): The loaded Gluon model; the result of calling model_fn on this script.
- array (NDArray): The result of a call to input_fn in response to an Amazon SageMaker InvokeEndpoint request.

Returns:
- (NDArray): An NDArray containing the prediction result.
"""
pass

def output_fn(ndarray, accept):
    """
    [Optional]

    Serializes prediction results. Amazon SageMaker invokes your output_fn with the NDArray returned by your predict_fn and the content type from the InvokeEndpoint request's accept header.

    This function should return a tuple of (transformation result, content type). In most cases, the returned content type should be the same as the accept content type, but it might differ; for example, when your code needs to return an error response.

    If you omit this function, Amazon SageMaker provides a default implementation. The default output_fn converts the prediction result into a JSON-encoded array data. For more information about the default input_fn, see the Amazon SageMaker Python SDK GitHub documentation.

    Args:
- ndarray: NDArray. The result of calling predict_fn.
- content_type: string. The content type from the InvokeEndpoint request's Accept header.

    Returns:
- (object, string): A tuple containing the transformed result and its content type
"""
pass

# -----
# Request handlers for Module models
# ----- #

def input_fn(model, input_data, content_type):
    """
    [Optional]

    Prepares data for transformation. Amazon SageMaker invokes your input_fn in response to an InvokeEndpoint operation on an Amazon SageMaker endpoint that contains this script. Amazon SageMaker passes in the MXNet Module returned by your model_fn, along with the input data and content type from the InvokeEndpoint request.

    The function should return an NDArray. Amazon SageMaker wraps the returned NDArray in a DataIter with a batch size that matches your model, and then passes it to your predict_fn.

    If you omit this function, Amazon SageMaker provides a default implementation. The default input_fn converts a JSON or CSV-encoded array data into an NDArray. For more information about the default input_fn, see the Amazon SageMaker Python SDK GitHub documentation.

    Args:

```

```

        - model: A Module; the result of calling model_fn on this script.
        - input_data: The input data from the payload of the
                      InvokeEndpoint request.
        - content_type: The content type of the request.

    Returns:
        - (NDArray): an NDArray
    """
    pass

def predict_fn(module, data):
    """
    [Optional]

    Performs prediction on an MXNet DataIter object. In response to an
    InvokeEndpoint request, Amazon SageMaker invokes your
    predict_fn with the model returned by your model_fn and DataIter
    that contains the result of the input_fn.

    The function should return a list of NDArray or a list of list of NDArray
    containing the prediction results. For more information, see the MXNet Module API
    <https://mxnet.incubator.apache.org/api/python/module.html#mxnet.module.BaseModule.predict>.

    If you omit this function, Amazon SageMaker provides a default implementation.
    The default predict_fn calls module.predict on the input
    data and returns the result.

    Args:
        - module (Module): The loaded MXNet Module; the result of calling
                           model_fn on this script.
        - data (DataIter): A DataIter containing the results of a
                           call to input_fn.

    Returns:
        - (object): A list of NDArray or list of list of NDArray
                     containing the prediction results.
    """
    pass

def output_fn(data, accept):
    """
    [Optional]

    Serializes prediction results. Amazon SageMaker invokes your output_fn with the
    results of predict_fn and the content type from the InvokeEndpoint
    request's accept header.

    This function should return a tuple of (transformation result, content
    type). In most cases, the returned content type should be the same as the
    accept content type, but it might differ. For example, when your code needs to
    return an error response.

    If you omit this function, Amazon SageMaker provides a default implementation.
    The default output_fn converts the prediction result into JSON or CSV-
    encoded array data, depending on the value of the accept header. For more
    information about the default input_fn, see the Amazon SageMaker Python SDK
    GitHub documentation.

    Args:
        - data: A list of NDArray or list of list of NDArray. The result of
                calling predict_fn.
        - content_type: A string. The content type from the InvokeEndpoint
                        request's Accept header.

```

```
    Returns:  
        - (object, string): A tuple containing the transformed result  
            and its content type.  
    """  
    pass
```

Examples: Using Amazon SageMaker with Apache MXNet

Amazon SageMaker provides the following example notebooks in your Amazon SageMaker notebook instance:

- **The Apache MXNet Module API**—In this example, you use the Amazon SageMaker `sagemaker.mxnet.MXNet` estimator class to train a model. The custom Apache MXNet training code trains a multilayer perceptron neural network that predicts the number in images of single-digit handwritten numbers. It uses images of handwritten numbers as training data.
- **The Apache MXNet Gluon API**—This example uses the Gluon API to do the same thing that the Apache MXNet Module API does.

These examples uses the high-level Python library provided by Amazon SageMaker. This library is available on the Amazon SageMaker notebook instance you created as part of Getting Started. For more information, see [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 26\)](#). However, if you choose to use your own terminal, you need to download and install the library using one of the following options:

- Get it from <https://github.com/aws/sagemaker-python-sdk>.
- Install it using pip:

```
$ pip install sagemaker
```

This documentation explains one Apache MXNet example in details. All of the Apache MXNet example notebooks that Amazon SageMaker provides follow the same pattern. They differ only in the custom Apache MXNet code they use in model training.

Topics

- [Apache MXNet Example 1: Using the Module API \(p. 283\)](#)

Apache MXNet Example 1: Using the Module API

This introductory Apache MXNet example demonstrates using Amazon SageMaker `sagemaker.mxnet.MXNet` estimator class, provided as part of Amazon SageMaker high-level Python library. It provides the `fit` method for model training in Amazon SageMaker and `deploy` method to deploy resulting model in Amazon SageMaker.

In this exercise, you construct a neural network classifier using the Apache MXNet Module API. You then train the model using the [The MNIST Database](#) dataset, which Amazon SageMaker provides in an S3 bucket.

In this example, you do the following

1. Train the model. During training, the following occurs:
 - a. Amazon SageMaker loads the Docker image containing the Apache MXNet framework.
 - b. Amazon SageMaker reads training data from the S3 bucket into the container's file system.
 - c. Your custom training code constructs a neural network classifier (using the `mxnet.module.Module` class).
 - d. The Amazon SageMaker code in the container runs training. Your training code reads the training data for model training.
2. Deploy the model using Amazon SageMaker hosting services. Amazon SageMaker returns an endpoint that you send requests to to get inferences.
3. Test the model. The example provides an HTML canvas in the notebook where you can write a single-digit number using your mouse. The image of the number is then sent to the model for inference.

Topics

- [Step 1: Create a Notebook and Initialize Variables \(p. 284\)](#)
- [Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code \(p. 285\)](#)
- [Step 3 : Deploy the Trained Model \(p. 287\)](#)
- [Step 4 : Invoke the Endpoint to Get Inferences \(p. 288\)](#)

Step 1: Create a Notebook and Initialize Variables

In this section, you create a Jupyter notebook in your Amazon SageMaker notebook instance and initialize variables.

1. Follow the instructions in [Step 1: Setting Up \(p. 24\)](#) to create the S3 bucket and IAM role.

For simplicity, we suggest that you use one S3 bucket to store both the training code and the results of model training.
2. Create the notebook.
 - a. If you haven't already done so, sign in to the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
 - b. Open the notebook instance by choosing `Open` next to its name. The Jupyter notebook server page appears.
 - c. To create a notebook, choose **Files**, **New**, and **conda_mxnet_p36**. This pre-installed environment includes the default Anaconda installation, Python 3, and MXNet.
 - d. Name the notebook.
3. To initialize variables, copy, paste, and run the following code in your notebook. Provide the name of the S3 bucket that contains your custom code. The `get_execution_role` function retrieves the IAM role you created at the time of creating your notebook instance. You can use the bucket that you created in [Step 1: Setting Up \(p. 24\)](#), or create a new bucket.

```
from sagemaker import get_execution_role

#Bucket location to save your custom code in tar.gz format.
custom_code_upload_location = 's3://your-bucket-name/customcode/mxnet'

#Bucket location where results of model training are saved.
model_artifacts_location = 's3://your-bucket-name/artifacts'

#IAM execution role that gives Amazon SageMaker access to resources in your AWS account.
#We can use the Amazon SageMaker Python SDK to get the role from our notebook environment.
```

```
role = get_execution_role()
```

Next Step

[Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code \(p. 285\)](#)

Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code

The high-level Python library provides the `MXNet` class with two methods: `fit` (for training a model) and `deploy` (for deploying a model).

To train the model:

1. Create an instance of the `sagemaker.mxnet.MXNet` class by copying, pasting, and running the following code:

```
from sagemaker.mxnet import MXNet

mnist_estimator = MXNet(entry_point='/home/ec2-user/sample-notebooks/sagemaker-python-
sdk/mxnet_mnist/mnist.py',
                       role=role,
                       output_path=model_artifacts_location,
                       code_location=custom_code_upload_location,
                       train_instance_count=1,
                       train_instance_type='ml.m4.xlarge',
                       hyperparameters={'learning_rate': 0.1})
```

Some of these constructor parameters are sent in the `fit` method call for model training in the next step.

Details:

- `entry_point` — The example uses only one source file (`mnist.py`) and it is already provided for you on your notebook instance. If your custom code is in one file, you specify only the `entry_point` parameter. If your training code consists of multiple files, you also add the `source_dir` parameter.

Note

You specify the source of only your custom code. The `sagemaker.mxnet.MXNet` object determines which Docker image to use for model training.

- `role` — The IAM role that Amazon SageMaker assumes when performing tasks on your behalf, such as downloading training data from an S3 bucket for model training and uploading training results to an S3 bucket.
- `code_location` — S3 location where you want the `fit` method (in the next step) to upload the tar archive of your custom Apache MXNet code.
- `output_path` — Identifies S3 location where you want to the result of model training (model artifacts) saved.
- `train_instance_count` and `train_instance_type` — You specify the number and type of instances to use for model training.

You can also train your model on your local computer by specifying `local` as the value for `train_instance_type` and `1` as the value for `train_instance_count`. For more information about local mode, see <https://github.com/aws/sagemaker-python-sdk#local-mode> in the *Amazon SageMaker Python SDK*.

- **Hyperparameters** — Any hyperparameters that you specify to influence the final quality of the model. Your custom training code uses these parameters.
2. Start model training by copying, pasting, and running the following code:

```
%time
import boto3

region = boto3.Session().region_name
train_data_location = 's3://sagemaker-sample-data-{}/mxnet/mnist/train'.format(region)
test_data_location = 's3://sagemaker-sample-data-{}/mxnet/mnist/test'.format(region)

mnist_estimator.fit({'train': train_data_location, 'test': test_data_location})
```

In the `fit` call, you specify the S3 URI strings that identify where the training and test dataset are stored. The `fit` method sends a [CreateTrainingJob \(p. 371\)](#) request to Amazon SageMaker.

You can get the training job information by calling the [DescribeTrainingJob \(p. 410\)](#) or viewing it in the console. The following is an example response:

```
{
  "AlgorithmSpecification": {
    "TrainingImage": "520713654638.dkr.ecr.us-west-2.amazonaws.com/sagemaker-mxnet-py2-cpu:1.0",
    "TrainingInputMode": "File"
  },
  "HyperParameters": {
    "learning_rate": "0.11",
    "sagemaker_submit_directory": "\"s3://sagemakermv/customcode/mxnet_mnist/sagemaker-mxnet-py2-cpu-2017-11-18-02-02-18-586/sourcedir.tar.gz\"",
    "sagemaker_program": "\"mnist.py\"",
    "sagemaker_container_log_level": "20",
    "sagemaker_job_name": "\"sagemaker-mxnet-py2-cpu-2017-11-18-02-02-18-586\"",
    "sagemaker_region": "\"us-west-2\""
  },
  "InputDataConfig": [
    {
      "DataSource": {
        "S3DataSource": {
          "S3DataDistributionType": "FullyReplicated",
          "S3DataType": "S3Prefix",
          "S3Uri": "s3://sagemaker-sample-data-us-west-2/mxnet/mnist/train"
        }
      },
      "ChannelName": "train"
    },
    {
      "DataSource": {
        "S3DataSource": {
          "S3DataDistributionType": "FullyReplicated",
          "S3DataType": "S3Prefix",
          "S3Uri": "s3://sagemaker-sample-data-us-west-2/mxnet/mnist/test"
        }
      },
      "ChannelName": "test"
    }
  ],
  "OutputDataConfig": {
    "S3OutputPath": "s3://sagemakermv/artifacts"
  },
  "TrainingJobName": "sagemaker-mxnet-py2-cpu-2017-11-18-02-02-18-586",
  "StoppingCondition": {
    "MaxRuntimeInSeconds": 86400
  }
}
```

```
        },
        "ResourceConfig": {
            "InstanceCount": 1,
            "InstanceType": "ml.m4.xlarge",
            "VolumeSizeInGB": 30
        },
        "RoleArn": "arn:aws:iam::account-id:role/SageMakerRole"
    }
```

Details:

- **TrainingImage**—Amazon SageMaker runs this image to create a container for model training. You don't explicitly identify this image in your request. Instead, the method dynamically chooses the appropriate image. It inspects the Python version in the interpreter and the GPU capability of the ML compute instance type that you specified when creating the Apache MXNet object.
- **Hyperparameters**—The request includes the hyperparameters that you specified when creating the `sagemaker.mxnet.MXNet` object. In addition, the request includes the following additional hyperparameters (all beginning with the prefix `sagemaker`). The hyperparameters starting with prefix "`sagemaker`" are used by Amazon SageMaker to set up the training environment.
 - `sagemaker_submit_directory`—Identifies the custom training code in S3 location to use for model training.

The high-level Python library does several things for you. In this case, the `fit` method first it creates gzipped tar archive from the custom code file(s), and uploads the archive to S3. This archive is specified in this hyperparameter.

- `sagemaker_program`—Identifies the primary module from which your training functions will be loaded (it is the `entry_point` parameter you specified when creating the `sagemaker.mxnet.MXNet` object).
- `sagemaker_container_log_level` — Sets the Python logging level.
- `InputDataConfig`—Specifies two channels (train and test). Each channel is a named input source the training code consumes.
- `OutputDataConfig`—Identifies the S3 location where you want Amazon SageMaker to save training results (model artifacts).

By default, the training job runs synchronously (you see the output in the notebook).

Next Step

[Step 3 : Deploy the Trained Model \(p. 287\)](#)

Step 3 : Deploy the Trained Model

You can use Amazon SageMaker hosting services to deploy the model. During deployment, Amazon SageMaker launches ML compute instances and deploys the model (model artifacts and inference code) on them. In response, you get an endpoint. To get inferences from the model, your application code can send requests to the endpoint.

The `sagemaker.mxnet.MXNet` class in the high-level Python library provides the `deploy` method for quickly deploying your model. The `deploy` method does the following in order:

1. Creates an Amazon SageMaker model by calling the [CreateModel \(p. 359\)](#) API. The model that you create in Amazon SageMaker holds information such as location of the model artifacts and the inference code image.
2. Creates an endpoint configuration by calling the [CreateEndpointConfig \(p. 352\)](#) API. This configuration holds necessary information including the name of the model (which was created in

the preceding step) and the resource configuration (the type and number of ML compute instances to launch for hosting).

3. Creates the endpoint by calling the [CreateEndpoint \(p. 349\)](#) API and specifying the endpoint configuration created in the preceding step. Amazon SageMaker launches ML compute instances as specified in the endpoint configuration, and deploys the model on them.

Copy, paste, and run the following code:

```
%time

predictor = mnist_estimator.deploy(initial_instance_count=1,
                                    instance_type='ml.m4.xlarge')
```

When the status of the endpoint is INSERVICE the API returns an `MXNetPredictor` object. Use the `predict` method of this object to obtain inferences.

Note

You can deploy a model to an endpoint hosted on your local computer by specifying `local` as the value for `train_instance_type` and `1` as the value for `train_instance_count`. For more information about local mode, see <https://github.com/aws/sagemaker-python-sdk#local-mode> in the *Amazon SageMaker Python SDK*.

Next Step

[Step 4 : Invoke the Endpoint to Get Inferences \(p. 288\)](#)

Step 4 : Invoke the Endpoint to Get Inferences

Your model is now deployed on Amazon SageMaker. To get inferences, send requests using the `predict` method provided by the `MXNetPredictor` object (returned in the preceding section). The method calls the Amazon SageMaker [InvokeEndpoint \(p. 470\)](#).

This example provides an HTML canvas that you can use to draw a number using your mouse. The test code sends this image to the model for inference.

1. Display the canvas by copying, pasting, and running the following code:

```
from IPython.display import HTML
HTML(open("/home/ec2-user/sample-notebooks/sagemaker-python-sdk/mxnet_mnist/
input.html").read())
```

2. Use your mouse to raw a single-digit number on the canvas.
3. Run the `predict` method to get inference from the model.

```
response = predictor.predict(data)
print('Raw prediction result:')
print(response)

labeled_predictions = list(zip(range(10), response[0]))
print('Labeled predictions: ')
print(labeled_predictions)

labeled_predictions.sort(key=lambda label_and_prob: 1.0 - label_and_prob[1])
print('Most likely answer: {}'.format(labeled_predictions[0]))
```

The following is an example of output. It shows the number that was inferred (7) and a number that represents the probability that the inference is correct.

```
Raw prediction result:  
[[1.7489463002839933e-11, 0.006231508683413267, 8.953022916102782e-05,  
0.0872468426823616, 0.0001965702831512317, 1.7784617739380337e-05,  
3.312719196180147e-11, 0.7383657097816467, 0.009811942465603352, 0.15804021060466766]]  
Labeled predictions:  
[(0, 1.7489463002839933e-11), (1, 0.006231508683413267), (2, 8.953022916102782e-05),  
(3, 0.0872468426823616), (4, 0.0001965702831512317), (5, 1.7784617739380337e-05),  
(6, 3.312719196180147e-11), (7, 0.7383657097816467), (8, 0.009811942465603352), (9,  
0.15804021060466766)]  
Most likely answer: (7, 0.7383657097816467)
```

In the result:

- The **Raw prediction result** is a list of 10 probability values that the model returned as inference, corresponding to the digits 0 through 9. From these values, the input digit is 7 based on the highest probability value (0.7383657097816467).
- The values are listed in order, one value for each digit (0 through 9). The model added labels to it and returned **Labeled predictions**.
- Based on the highest probability, our code returned the **Most likely answer** (digit 7).

You can now delete the resources that you created in this exercise. For more information, see [Step 4: Clean up \(p. 48\)](#).

Your Amazon SageMaker notebook instance includes additional examples.

Using Chainer with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Chainer code. Amazon SageMaker provides an open-source container that makes writing a Chainer script and running it in Amazon SageMaker easier. For information about how to build and use the Amazon SageMaker Chainer container, see the GitHub repository at <https://github.com/aws/sagemaker-chainer-container>.

For information about Chainer versions supported by the Amazon SageMaker Chainer container and writing Chainer training scripts and using Chainer estimators with Amazon SageMaker, see <https://github.com/aws/sagemaker-python-sdk#chainer-sagemaker-estimators>.

Using PyTorch with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom PyTorch code. Amazon SageMaker provides an open-source container that makes writing a PyTorch script and running it in Amazon SageMaker easier. For information about how to build and use the Amazon SageMaker PyTorch container, see the GitHub repository at <https://github.com/aws/sagemaker-pytorch-container>.

For information about PyTorch versions supported by the Amazon SageMaker PyTorch container, see <https://github.com/aws/sagemaker-python-sdk#pytorch-sagemaker-estimators>.

For information about writing PyTorch training scripts and using PyTorch estimators with Amazon SageMaker, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/pytorch/README.rst>.

Using 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`s 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.

Downloading 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](#)

- 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>
```

Integrating 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. 371) 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, to read training data from an S3 bucket and to write model artifacts to an S3 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. 359) request to Amazon SageMaker.
- ii. Sends a [CreateEndpointConfig](#) (p. 352) request to Amazon SageMaker.
- iii. Sends a [CreateEndpoint](#) (p. 349) 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` methods sends a request to the [InvokeEndpoint](#) (p. 470) Amazon SageMaker API to get inferences. The `transform` method appends the inferences to the input `DataFrame`.

Example 1: Using Amazon SageMaker for Training and Inference with Apache Spark

Topics

- [Using Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark \(p. 298\)](#)
- [Using the SageMakerEstimator in a Spark Pipeline \(p. 299\)](#)

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 [Using Apache Spark with Amazon SageMaker \(p. 292\)](#). 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 \(p. 142\)](#).

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, such as requests to create a training job and API calls for deploying the model using Amazon SageMaker hosting services, to Amazon SageMaker.

- 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, 0-9.
 - Identifies that each input image has 784 features (each handwritten number is a 28 x 28-pixel image, making 784 features).
- Call 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, [Integrating Your Apache Spark Application with Amazon SageMaker \(p. 293\)](#). 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.

SageMaker Spark Github Readme provides information on how to run these examples. For more information, see <https://github.com/aws/sagemaker-spark/blob/master/README.md>.

Using Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark

In [Example 1: Using Amazon SageMaker for Training and Inference with Apache Spark \(p. 294\)](#), 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 code sample 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.KMeansProtobufResponseRowDeser

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.

Using 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: Using 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>.

Amazon SageMaker Libraries

The Amazon SageMaker libraries and related information are available at the following locations:

- Amazon SageMaker Apache Spark Library - <https://github.com/aws/sagemaker-spark>
- Amazon SageMaker high-level Python library - <https://github.com/aws/sagemaker-python-sdk>

Authentication and Access Control for Amazon SageMaker

Access to Amazon SageMaker requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon SageMaker notebook instance or an Amazon Elastic Compute Cloud (Amazon EC2) instance. The following sections provide details on how you can use [AWS Identity and Access Management \(IAM\)](#) and Amazon SageMaker to help secure access to your resources.

- [Authentication \(p. 302\)](#)
- [Access Control \(p. 303\)](#)

Authentication

You can access AWS as any of the following types of identities:

- **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 user** – An [IAM user](#) is an identity within your AWS account that has specific custom permissions (for example, permissions to create the in Amazon SageMaker). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](#), [AWS Discussion Forums](#), or the [AWS Support Center](#).

In addition to a user name and password, you can also generate [access keys](#) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several SDKs](#) or by using the [AWS Command Line Interface \(CLI\)](#). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Amazon SageMaker supports *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](#).

- **IAM role** – An [IAM role](#) is an IAM identity that you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:
 - **Federated user access** – Instead of creating an IAM user, you can use existing user 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](#).

- **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. 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 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*.

Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon SageMaker resources. For example, you must have permissions to create an Amazon SageMaker notebook instance.

The following sections describe how to manage permissions for Amazon SageMaker. We recommend that you read the overview first.

- [Overview of Managing Access Permissions to Your Amazon SageMaker Resources \(p. 303\)](#)
- [Using Identity-based Policies \(IAM Policies\) for Amazon SageMaker \(p. 307\)](#)
- [Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference \(p. 312\)](#)

Overview of Managing Access Permissions to Your Amazon SageMaker Resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles). Some services (such as AWS Lambda) also support attaching permissions policies to resources.

Note

An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see [IAM Best Practices](#) in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

- [Amazon SageMaker Resources and Operations \(p. 304\)](#)
- [Understanding Resource Ownership \(p. 304\)](#)
- [Managing Access to Resources \(p. 305\)](#)
- [Specifying Policy Elements: Resources, Actions, Effects, and Principals \(p. 306\)](#)

- [Specifying Conditions in a Policy \(p. 307\)](#)

Amazon SageMaker Resources and Operations

In Amazon SageMaker, the primary resource is a *notebook instance*. Amazon SageMaker also supports additional resource types: *training jobs*, *models*, *endpoint configurations*, *endpoints*, and *tags*. These additional resources are referred to as *subresources*. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to.

Except for *tags*, these resources and subresources have unique ARNs associated with them, as shown in the following table. *Tags* use the ARN of the resource that they are modifying. For example, when used on a model, the `AddTag` action uses the same ARN as the model resource.

Resource Type	ARN Format
Batch Transform Job	<code>arn:aws:sagemaker:region:account-id:transform-job/<i>transformJobName</i></code>
Endpoint	<code>arn:aws:sagemaker:region:account-id:endpoint/<i>endpointName</i></code>
Endpoint Config	<code>arn:aws:sagemaker:region:account-id:endpoint-config/<i>endpointConfigName</i></code>
Hyperparameter Tuning Job	<code>arn:aws:sagemaker:region:account-id:hyper-parameter-tuning-job/<i>hyperParameterTuningJobName</i></code>
Model	<code>arn:aws:sagemaker:region:account-id:model/<i>modelName</i></code>
Notebook Instance	<code>arn:aws:sagemaker:region:account-id:notebook-instance/<i>notebookInstanceName</i></code>
Notebook Instance Lifecycle Configuration	<code>arn:aws:sagemaker:region:account-id:notebook-instance-lifecycle-config/<i>notebookInstanceLifecycleConfigName</i></code>
Training Job	<code>arn:aws:sagemaker:region:account-id:training-job/<i>trainingJobName</i></code>

Amazon SageMaker provides a set of operations to work with Amazon SageMaker resources. For a list of available operations, see the [Amazon SageMaker API Reference \(p. 344\)](#).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the [principal entity](#) (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a notebook instance, your AWS account is the owner of the resource (in Amazon SageMaker, the resource is a notebook instance).
- If you create an IAM user in your AWS account and grant permissions to create a notebook instance to that user, the user can create a notebook instance. However, your AWS account, to which the user belongs, owns the notebook instance resource.
- If you create an IAM role in your AWS account with permissions to create a notebook instance, anyone who can assume the role can create a notebook instance. Your AWS account, to which the user belongs, owns the notebook instance resource.

Managing Access to Resources

A *permissions policy* describes who has access to what. The following section explains the options for creating permissions policies.

Note

This section discusses using IAM in the context of Amazon SageMaker. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see [What Is IAM?](#) in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see [AWS IAM Policy Reference](#) in the *IAM User Guide*.

Permissions policies attached to an IAM identity are referred to as *identity-based* policies (IAM polices). Permissions policies attached to a resource are referred to as *resource-based* policies. Amazon SageMaker supports only identity-based permissions policies.

Topics

- [Identity-Based Policies \(IAM Policies\) \(p. 305\)](#)
- [Resource-Based Policies \(p. 306\)](#)

Identity-Based Policies (IAM Policies)

You can attach permissions policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create a Amazon SageMaker resource, such as a notebook instance, you can attach a permissions policy to a user or to a group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in account A can create a role to grant cross-account permissions to another AWS account (for example, account B) or an AWS service as follows:
 1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in account A.
 2. Account A administrator attaches a trust policy to the role identifying account B as the principal who can assume the role.
 3. Account B administrator can then delegate permissions to assume the role to any users in account B. Doing this allows users in account B to create or access resources in account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

For more information about using IAM to delegate permissions, see [Access Management](#) in the *IAM User Guide*.

Some of the Amazon SageMaker actions (for example, `CreateTrainingJob`) require the user to pass an IAM role to Amazon SageMaker so that the service can assume that role and its permissions. To pass a role (and its permissions) to an AWS service, a user must have permissions to pass the role to the service. To allow a user to pass a role to an AWS service, you must grant permission for the `iam:PassRole` action. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*.

The following is an example permission policy. You attach it to an IAM user.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
"Action": [
    "sagemaker>CreateModel"
],
"Effect": "Allow",
"Resource":
    "arn:aws:sagemaker:region:account-id:model/modelName"
},
{
    "Action": "iam:PassRole",
    "Effect": "Allow",
    "Resource": [
        "arn:aws:iam::account-id:role/role-name"
    ]
}
}
```

For more information about using identity-based policies with Amazon SageMaker, see [Using Identity-based Policies \(IAM Policies\) for Amazon SageMaker \(p. 307\)](#). For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

Resource-Based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon SageMaker doesn't support resource-based policies.

Specifying Policy Elements: Resources, Actions, Effects, and Principals

For each Amazon SageMaker resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon SageMaker defines a set of actions that you can specify in a policy. For example, for the Amazon SageMaker notebook instance resource, the following actions are defined: `CreateNotebookInstance`, `DeleteNotebookInstance`, and `DescribeNotebookInstance`. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see [Amazon SageMaker Resources and Operations \(p. 304\)](#) and [API Reference \(p. 344\)](#).

The following are the most basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the identity-based policy applies to. For more information, see [Amazon SageMaker Resources and Operations \(p. 304\)](#).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, you can use `sagemaker>CreateModel` to add a model to the notebook instance.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. Amazon SageMaker doesn't support resource-based policies.

To learn more about IAM policy syntax and to read policy descriptions, see [AWS IAM Policy Reference](#) in the *IAM User Guide*.

For a list showing all of the Amazon SageMaker API operations and the resources that they apply to, see [Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference \(p. 312\)](#).

Specifying Conditions in a Policy

When you grant permissions, you can use the IAM policy language to specify the conditions under which a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see [Conditions](#) in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon SageMaker. However, there are AWS-wide condition keys that you can use as appropriate. For an example of AWS-wide keys used in an Amazon SageMaker permissions policy, see [Permissions Required to Use the Amazon SageMaker Console \(p. 308\)](#). For a complete list of AWS-wide keys, see [Available Keys for Conditions](#) in the *IAM User Guide*.

Using Identity-based Policies (IAM Policies) for Amazon SageMaker

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on Amazon SageMaker resources.

Important

We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your Amazon SageMaker resources. For more information, see [Overview of Managing Access Permissions to Your Amazon SageMaker Resources \(p. 303\)](#).

Topics

- [Permissions Required to Use the Amazon SageMaker Console \(p. 308\)](#)
- [AWS Managed \(Predefined\) Policies for Amazon SageMaker \(p. 309\)](#)
- [Control Access to Amazon SageMaker Resources by Using Tags \(p. 310\)](#)

The following is an example of a basic permissions policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowCreate-Describe-Delete-Models",  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:CreateModel",  
                "sagemaker:DescribeModel",  
                "sagemaker>DeleteModel"],  
            "Resource": "*"  
        },  
        {  
            "Statement": [  
                {  
                    "Sid": "AdditionalIamPermission",  
                    "Effect": "Allow",  
                    "Action": [  
                        "iam:PassRole"],  
                    "Resource": "arn:aws:iam::account-id:role/role-name"  
                }  
            ]  
        }  
    ]  
}
```

The policy has two statements:

- The first statement grants permission for three Amazon SageMaker actions (`sagemaker:CreateModel`, `sagemaker:DescribeModel`, and `sagemaker>DeleteModel`) within an Amazon SageMaker notebook instance. Using the wildcard character (*) as the resource grants universal permissions for these actions across all AWS Regions and models owned by this account.
 - The second statement grants permission for the `iam:PassRole` action, which is needed for the Amazon SageMaker action `sagemaker:CreateModel`, which is allowed by the first statement.

The policy doesn't specify the `Principal` element because in an identity-based policy you don't specify the principal who gets the permission. When you attach the policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

For a table showing all of the Amazon SageMaker API operations and the resources that they apply to, see [Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference \(p. 312\)](#).

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 \(p. 312\)](#).

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:

```
        "kms>ListAliases"
    ],
    "Resource": "*"
}
// List/create execution roles for Create forms
{
    "Sid": "ListAndCreateExecutionRoles",
    "Effect": "Allow",
    "Action": [
        "iam>ListRoles",
        "iam>CreateRole",
        "iam>CreateRole",
        "iam>CreatePolicy",
        "iam>AttachRolePolicy"
    ],
    "Resource": "*"
},
// PassRole permissions as required by CreateNotebookInstance, CreateTrainingJob,
and CreateModel.
{
    "Sid": "PassRoleForExecutionRoles",
    "Effect": "Allow",
    "Action": [
        "iam>PassRole"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "sagemaker.amazonaws.com"
        }
    }
}
]
```

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.

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. 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 \(p. 347\)](#).
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"
                }
            }
        }
    ]
}
```

```
    ]  
}
```

Using 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 \(p. 355\)](#). 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 \(p. 347\)](#), 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 \(p. 388\)](#), 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 \(p. 445\)](#) 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 API Permissions: Actions, Permissions, and Resources Reference

When you are setting up [Access Control \(p. 303\)](#) 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 \(p. 347\)](#)

Required Permissions (API Action): `sagemaker:AddTags`

Resources: *

API Operation: [CreateEndpoint \(p. 349\)](#)

Required Permissions (API Action): `sagemaker>CreateEndpoint`

Resources: `arn:aws:sagemaker:region:account-id:endpoint/endpointName`

API Operation: [CreateEndpointConfig \(p. 352\)](#)

Required Permissions (API Action): `sagemaker>CreateEndpointConfig`

Resources: `arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName`

API Operation: [CreateModel \(p. 359\)](#)

Required Permissions (API Action): `sagemaker>CreateModel, iam:PassRole`

Resources: `arn:aws:sagemaker:region:account-id:model/modelName`

API Operation: [CreateNotebookInstance \(p. 362\)](#)

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 \(p. 371\)](#)

Required Permissions (API Action): `sagemaker>CreateTrainingJob, iam:PassRole`

Resources: `arn:aws:sagemaker:region:account-id:training-job/trainingJobName`

API Operation: [DeleteEndpoint \(p. 381\)](#)

Required Permissions (API Action): `sagemaker>DeleteEndpoint`

Resources: `arn:aws:sagemaker:region:account-id:endpoint/endpointName`

API Operation: [DeleteEndpointConfig \(p. 383\)](#)

Required Permissions (API Action): `sagemaker>DeleteEndpointConfig`

Resources: `arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName`

API Operation: [DeleteModel \(p. 384\)](#)

Required Permissions (API Action): `sagemaker>DeleteModel`

Resources: `arn:aws:sagemaker:region:account-id:model/modelName`

API Operation: [DeleteNotebookInstance \(p. 385\)](#)

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. 388)

Required Permissions (API Action): sagemaker:DeleteTags

Resources: *

API Operation: [DescribeEndpoint](#) (p. 390)

Required Permissions (API Action): sagemaker:DescribeEndpoint

Resources: arn:aws:sagemaker:*region:account-id:endpoint/endpointName*

API Operation: [DescribeEndpointConfig](#) (p. 393)

Required Permissions (API Action): sagemaker:DescribeEndpointConfig

Resources: arn:aws:sagemaker:*region:account-id:endpoint-config/endpointConfigName*

API Operation: [DescribeModel](#) (p. 400)

Required Permissions (API Action): sagemaker:DescribeModel

Resources: arn:aws:sagemaker:*region:account-id:model/modelName*

API Operation: [DescribeNotebookInstance](#) (p. 403)

Required Permissions (API Action): sagemaker:DescribeNotebookInstance

Resources: arn:aws:sagemaker:*region:account-id:notebook-instance/notebookInstanceName*

API Operation: [DescribeTrainingJob](#) (p. 410)

Required Permissions (API Action): sagemaker:DescribeTrainingJob

Resources: arn:aws:sagemaker:*region:account-id:training-job/trainingJobName*

API Operation: [CreatePresignedNotebookInstanceUrl](#) (p. 369)

Required Permissions (API Action): sagemaker>CreatePresignedNotebookInstanceUrl

Resources: arn:aws:sagemaker:*region:account-id:notebook-instance/notebookInstanceName*

API Operation: [InvokeEndpoint](#) (p. 470)

Required Permissions (API Action): sagemaker:InvokeEndpoint

Resources: arn:aws:sagemaker:*region:account-id:endpoint/endpointName*

API Operation: [ListEndpointConfigs](#) (p. 420)

Required Permissions (API Action): sagemaker>ListEndpointConfigs

Resources: *

API Operation: [ListEndpoints](#) (p. 423)

Required Permissions (API Action): sagemaker>ListEndpoints

Resources: *

API Operation: [ListModels](#) (p. 430)

Required Permissions (API Action): sagemaker>ListModels

Resources: *

API Operation: [ListNotebookInstances \(p. 436\)](#)

Required Permissions (API Action): `sagemaker>ListNotebookInstances`

Resources: *

API Operation: [ListTags \(p. 440\)](#)

Required Permissions (API Action): `sagemaker>ListTags`

Resources: *

API Operation: [ListTrainingJobs \(p. 442\)](#)

Required Permissions (API Action): `sagemaker>ListTrainingJobs`

Resources: *

API Operation: [StartNotebookInstance \(p. 451\)](#)

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: [StopNotebookInstance \(p. 455\)](#)

Required Permissions (API Action): `sagemaker StopNotebookInstance`

Resources: `arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName`

API Operation: [StopTrainingJob \(p. 457\)](#)

Required Permissions (API Action): `sagemaker StopTrainingJob`

Resources: `arn:aws:sagemaker:region:account-id:training-job/trainingJobName`

API Operation: [UpdateEndpoint \(p. 461\)](#)

Required Permissions (API Action): `sagemaker UpdateEndpoints`

Resources: `arn:aws:sagemaker:region:account-id:endpoint/endpointName`

API Operation: [UpdateNotebookInstance \(p. 465\)](#)

Required Permissions (API Action): `sagemaker UpdateNotebookInstance`, `iam PassRole`

Resources: `arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName`

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 \(p. 362\)](#), [CreateHyperParameterTuningJob \(p. 355\)](#), [CreateTrainingJob \(p. 371\)](#), and [CreateModel \(p. 359\)](#).

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 well scoped to the actions you need to perform. For more information see [Using the AWS Managed Permission Policy \(AmazonSageMakerFullAccess\) for an Execution Role \(p. 324\)](#). If you prefer to create custom policies and manage permissions, see the following topics.

For more information about IAM roles, see [IAM Roles](#) in the *IAM User Guide*.

Topics

- [CreateNotebookInstance API: Execution Role Permissions \(p. 316\)](#)
- [CreateHyperParameterTuningJob API: Execution Role Permissions \(p. 319\)](#)
- [CreateTrainingJob API: Execution Role Permissions \(p. 321\)](#)
- [CreateModel API: Execution Role Permissions \(p. 322\)](#)
- [Using the AWS Managed Permission Policy \(AmazonSageMakerFullAccess\) for an Execution Role \(p. 324\)](#)

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",  
                "lambda:InvokeFunction"  
            ],  
            "Resource": "  
                arn:aws:lambda:  
                    <region>:  
                    <account>:  
                    function:  
                        CreateNotebookInstance  
            "  
        }  
    ]  
}
```

```

        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "cloudwatch:PutMetricData",
        "logs>CreateLogGroup",
        "logs>CreateLogStream",
        "logs>DescribeLogStreams",
        "logs>PutLogEvents",
        "logs>GetLogEvents",
        "s3>CreateBucket",
        "s3>ListBucket",
        "s3:GetBucketLocation",
        "s3GetObject",
        "s3:PutObject",
        "s3>DeleteObject"
    ],
    "Resource": "*"
},
{
    "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 `s3GetObject`, `s3PutObject`, and `s3DeleteObject` 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`

`SupplementalContainers.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 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:::mybucket"  
            ]  
        }  
    ]  
}
```

```
        "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 `s3GetObject` and `s3PutObject` 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"  
    ]  
}
```

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 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:::outputbucket"  
            ]  
        }  
    ]  
}
```

```

        "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.AlgorithmSpecifications.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 `s3GetObject` and `s3PutObject` 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"
    ]
}
```

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 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 \(p. 359\)](#) 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"  
    ]  
}
```

Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role

You can create an execution role one of two ways:

- In the Amazon SageMaker console when you create a notebook instance, training job, or model.
- In the AWS Identity and Access Management (IAM) console. You then specify the role as you follow the notebook instance, training job, and model creation workflows in the Amazon SageMaker console.

Regardless of how you create an execution role, you can attach the AWS-managed permission policy (`AmazonSageMakerFullAccess`) to the role.

When attaching the `AmazonSageMakerFullAccess` policy to a role, you must do one of the following to allow Amazon SageMaker to access your S3 bucket:

- Include the string `"SageMaker"` or `"sagemaker"` in the name of the bucket where you store training data, or the model artifacts resulting from model training, or both.
- Include the string `"SageMaker"` or `"sagemaker"` in the object name of the training data object(s).
- Tag the S3 object with `"sagemaker=true"`. The key and value are case sensitive. For more information, see [Object Tagging](#) in the Amazon Simple Storage Service Developer Guide.
- Add a bucket policy that allows access for the execution role. For more information, see [Using Bucket Policies and User Policies](#) in the Amazon Simple Storage Service Developer Guide.

You can attach additional policies that specify the resources for which you want to grant permissions for the `s3:GetObject`, `s3:PutObject`, and `s3>ListBucket` actions. In the IAM console, you can attach a customer managed policy or an inline policy to your execution role(s). Alternatively, when you create a role in the Amazon SageMaker console, you can attach a customer managed policy that specifies the S3 buckets. This resulting execution role has the prefix `"AmazonSageMaker-ExecutionRole-"`.

Monitoring 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](#).

Monitoring 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 \(p. 470\)](#).

Metrics are available at a 1-minute frequency.

Metric	Description
Invocation4XXErrors	<p>The number of <code>InvokeEndpoint</code> requests where the model returned a 4xx HTTP response code. For each 4xx response, 1 is sent; otherwise, 0 is sent.</p> <p>Units: None</p> <p>Valid statistics: Average, Sum</p>
Invocation5XXErrors	The number of <code>InvokeEndpoint</code> requests where the model returned a 5xx HTTP response code. For each 5xx response, 1 is sent; otherwise, 0 is sent.

Metric	Description
	Units: None Valid statistics: Average, Sum
Invocations	The number of <code>InvokeEndpoint</code> 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 <code>InstanceCount</code> in each <code>ProductionVariant</code> . $1/\text{numberOfInstances}$ is sent as the value on each request, where <code>numberOfInstances</code> is the number of active instances for the <code>ProductionVariant</code> 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 <code>ModelLatency</code> . Overhead latency can vary depending on multiple factors, including request and response payload sizes, request frequency, and authentication/authorization of the request. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count

Dimensions for Endpoint Invocation Metrics

Dimension	Description
EndpointName, VariantName	Filters endpoint invocation metrics for a <code>ProductionVariant</code> 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	<p>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, <code>CPUUtilization</code> can range from 0% to 400%.</p> <p>For training jobs, the value is the CPU utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the CPU utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the CPU utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p>
MemoryUtilizaton	<p>The percentage of memory that is used by the containers on an instance. This value can range between 0% and 100%.</p> <p>For training jobs, the value is the memory utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the memory utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the memory utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p>
GPUUtilization	<p>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, <code>GPUUtilization</code> can range from 0% to 400%.</p> <p>For training jobs, the value is the GPU utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the GPU utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the GPU utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p>
GPUMemoryUtilization	<p>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, <code>GPUMemoryUtilization</code> can range from 0% to 400%.</p> <p>For training jobs, the value is the GPU memory utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the GPU memory utilization of the transform container on the instance.</p>

Metric	Description
	<p>For endpoint variants, the value is the sum of the GPU memory utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p>
DiskUtilization	<p>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.</p> <p>For training jobs, the value is the disk space utilization of the algorithm container on the instance.</p> <p>For endpoint variants, the value is the sum of the disk space utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p>

Dimensions for Training Job, Batch Transform Job, and Endpoint Instance Metrics

Dimension	Description
Host	<p>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.</p> <p>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.</p> <p>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.</p>

Logging Amazon SageMaker with Amazon CloudWatch

To help you debug your training jobs, endpoints, 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

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]
/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

Note

The /aws/sagemaker/NotebookInstances log group is made when you create a Notebook Instance with a Lifecycle configuration. For more information, see [Step 2.1: \(Optional\) Customize a Notebook Instance \(p. 29\)](#).

Logging 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 \(p. 470\)](#), 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](#).

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 \(p. 470\)](#), are logged by CloudTrail and are documented in the [Actions \(p. 344\)](#). 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 \(p. 355\)](#), Amazon SageMaker creates training jobs to evaluate various combinations of hyperparameters to find the best result. Similarly, when you call [StopHyperParameterTuningJob \(p. 453\)](#) 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": "ASXIAGXAMPLEQULKNXV",  
        "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":"ASXIAGXAMPLEQULKNXV",
        "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"
}

```

Best Practices

Use this as a reference to quickly find recommendations for maximizing Amazon SageMaker performance.

Topics

- [Deployment \(p. 332\)](#)

Deployment

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.

Security

This section provides guidelines for securing notebook instances, connecting to Amazon SageMaker API through a VPC Interface Endpoint, and allowing training and hosting instances to access data in your VPC.

Topics

- [Notebook Instance Security \(p. 333\)](#)
- [Connect to Amazon SageMaker Through a VPC Interface Endpoint \(p. 334\)](#)
- [Protect Training Jobs by Using an Amazon Virtual Private Cloud \(p. 335\)](#)
- [Protect Endpoints by Using an Amazon Virtual Private Cloud \(p. 337\)](#)
- [Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud \(p. 340\)](#)

Notebook Instance Security

Note the following security considerations for notebook instances.

Topics

- [Notebook Instances Are Internet-Enabled by Default \(p. 333\)](#)

Notebook Instances Are Internet-Enabled by Default

Amazon SageMaker notebook instances are internet-enabled. 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 a NAT gateway and your security groups allow outbound connections. For information about creating a VPC interface endpoint for your notebook instance, see [Connect to a Notebook Instance Through a VPC Interface Endpoint \(p. 69\)](#). 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](#). 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"
}
]
```

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](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](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 \(p. 344\)](#) inside your VPC. The result `AuthorizedUrl` from the [CreatePresignedNotebookInstanceUrl \(p. 369\)](#) 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 \(p. 69\)](#).

To learn more about AWS PrivateLink, see the [AWS PrivateLink documentation](#) and visit the AWS Blog. Refer to [VPC Pricing](#) for the price of VPC Endpoints. To learn more about VPC and Endpoints, see [Amazon VPC](#).

Protect Training Jobs by Using an Amazon Virtual Private Cloud

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*.

Configuring 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 \(p. 371\)](#) 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"  
    ]  
}
```

Configuring 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*.

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](#).

The following policy allows access to S3 buckets. Edit this policy to allow access only the resources that your training job needs.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject",  
                "s3>ListBucket",  
                "s3:GetBucketLocation",  
                "s3>DeleteObject",  
                "s3>ListMultipartUploadParts",  
                "s3:AbortMultipartUpload"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

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*.

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 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](#).

Connecting 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 jobs 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*.

Protect Endpoints by Using an Amazon Virtual Private Cloud

Amazon SageMaker hosts models in an Amazon Virtual Private Cloud by default. However, models access AWS resources—such as the Amazon S3 buckets that you use to store model artifacts—over the internet.

To avoid making your data and model containers accessible over 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 model containers and data because you can configure the VPC so that it isn't 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 VPC configuration when you create a model by specifying subnets and security groups. When you specify your subnets and security groups, Amazon SageMaker creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the specified 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*.

Configuring a Model for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the [CreateModel \(p. 359\)](#) API or when you create a model in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and then attaches 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. It also enables 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"  
    ]  
}
```

Configuring Your Private VPC for Amazon SageMaker Hosting

Use the following guidelines to configure a private VPC for your Amazon SageMaker hosting jobs . For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two available private IP addresses for each instance for a hosted model. 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, models that use that VPC can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint for access to the buckets. We recommend that you create a VPC endpoint with a custom policy

that restricts access to your S3 buckets to only requests from your VPC. For more information, see [Endpoints for Amazon S3](#) in the *Amazon VPC User Guide*.

The following policy allows access to S3 buckets. Edit this policy to allow access only the resources that your model needs.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject",  
                "s3>ListBucket",  
                "s3:GetBucketLocation",  
                "s3>DeleteObject",  
                "s3>ListMultipartUploadParts",  
                "s3:AbortMultipartUpload"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

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*.

Note

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 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/*"
        ]
    }
}
```

Connect 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 on the internet, 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*.

Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud

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 creating a private VPC and configuring it so that it is not connected to the internet. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*.

With a VPC, you can monitor network traffic to and from your model containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

When you specify subnets and security groups, Amazon SageMaker creates elastic network interfaces (ENIs) and associates them with your security groups in one of the subnets. ENIs allow model containers to connect to resources in your VPC. For information about ENIs, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

Configuring 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. 359) API and provide this model when you create a batch 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 batch 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"  
    ]  
}
```

Configuring 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*.

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a batch 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](#).

The following policy allows access to S3 buckets. Edit this policy to allow access only the resources that your batch transform job needs.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject",  
                "s3>ListBucket",  
                "s3:GetBucketLocation",  
                "s3>DeleteObject",  
                "s3>ListMultipartUploadParts",  
                "s3:AbortMultipartUpload"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

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*.

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 them, 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 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](#).

Connecting 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 jobs 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*.

Limits and Supported Regions

For Amazon SageMaker service limits, see [Amazon SageMaker Limits](#).

For a list of the regions supporting Amazon SageMaker, see [Amazon SageMaker Regions](#).

API Reference

This section contains the API Reference documentation.

Topics

- [Actions \(p. 344\)](#)
- [Data Types \(p. 473\)](#)

Actions

The following actions are supported by Amazon SageMaker Service:

- [AddTags \(p. 347\)](#)
- [CreateEndpoint \(p. 349\)](#)
- [CreateEndpointConfig \(p. 352\)](#)
- [CreateHyperParameterTuningJob \(p. 355\)](#)
- [CreateModel \(p. 359\)](#)
- [CreateNotebookInstance \(p. 362\)](#)
- [CreateNotebookInstanceLifecycleConfig \(p. 366\)](#)
- [CreatePresignedNotebookInstanceUrl \(p. 369\)](#)
- [CreateTrainingJob \(p. 371\)](#)
- [CreateTransformJob \(p. 376\)](#)
- [DeleteEndpoint \(p. 381\)](#)
- [DeleteEndpointConfig \(p. 383\)](#)
- [DeleteModel \(p. 384\)](#)
- [DeleteNotebookInstance \(p. 385\)](#)
- [DeleteNotebookInstanceLifecycleConfig \(p. 387\)](#)
- [DeleteTags \(p. 388\)](#)
- [DescribeEndpoint \(p. 390\)](#)
- [DescribeEndpointConfig \(p. 393\)](#)
- [DescribeHyperParameterTuningJob \(p. 395\)](#)
- [DescribeModel \(p. 400\)](#)
- [DescribeNotebookInstance \(p. 403\)](#)
- [DescribeNotebookInstanceLifecycleConfig \(p. 407\)](#)
- [DescribeTrainingJob \(p. 410\)](#)
- [DescribeTransformJob \(p. 416\)](#)
- [ListEndpointConfigs \(p. 420\)](#)
- [ListEndpoints \(p. 423\)](#)
- [ListHyperParameterTuningJobs \(p. 426\)](#)
- [ListModels \(p. 430\)](#)
- [ListNotebookInstanceLifecycleConfigs \(p. 433\)](#)
- [ListNotebookInstances \(p. 436\)](#)
- [ListTags \(p. 440\)](#)

- [ListTrainingJobs \(p. 442\)](#)
- [ListTrainingJobsForHyperParameterTuningJob \(p. 445\)](#)
- [ListTransformJobs \(p. 448\)](#)
- [StartNotebookInstance \(p. 451\)](#)
- [StopHyperParameterTuningJob \(p. 453\)](#)
- [StopNotebookInstance \(p. 455\)](#)
- [StopTrainingJob \(p. 457\)](#)
- [StopTransformJob \(p. 459\)](#)
- [UpdateEndpoint \(p. 461\)](#)
- [UpdateEndpointWeightsAndCapacities \(p. 463\)](#)
- [UpdateNotebookInstance \(p. 465\)](#)
- [UpdateNotebookInstanceLifecycleConfig \(p. 468\)](#)

The following actions are supported by Amazon SageMaker Runtime:

- [InvokeEndpoint \(p. 470\)](#)

Amazon SageMaker Service

The following actions are supported by Amazon SageMaker Service:

- [AddTags \(p. 347\)](#)
- [CreateEndpoint \(p. 349\)](#)
- [CreateEndpointConfig \(p. 352\)](#)
- [CreateHyperParameterTuningJob \(p. 355\)](#)
- [CreateModel \(p. 359\)](#)
- [CreateNotebookInstance \(p. 362\)](#)
- [CreateNotebookInstanceLifecycleConfig \(p. 366\)](#)
- [CreatePresignedNotebookInstanceUrl \(p. 369\)](#)
- [CreateTrainingJob \(p. 371\)](#)
- [CreateTransformJob \(p. 376\)](#)
- [DeleteEndpoint \(p. 381\)](#)
- [DeleteEndpointConfig \(p. 383\)](#)
- [DeleteModel \(p. 384\)](#)
- [DeleteNotebookInstance \(p. 385\)](#)
- [DeleteNotebookInstanceLifecycleConfig \(p. 387\)](#)
- [DeleteTags \(p. 388\)](#)
- [DescribeEndpoint \(p. 390\)](#)
- [DescribeEndpointConfig \(p. 393\)](#)
- [DescribeHyperParameterTuningJob \(p. 395\)](#)
- [DescribeModel \(p. 400\)](#)
- [DescribeNotebookInstance \(p. 403\)](#)
- [DescribeNotebookInstanceLifecycleConfig \(p. 407\)](#)
- [DescribeTrainingJob \(p. 410\)](#)
- [DescribeTransformJob \(p. 416\)](#)
- [ListEndpointConfigs \(p. 420\)](#)

- [ListEndpoints \(p. 423\)](#)
- [ListHyperParameterTuningJobs \(p. 426\)](#)
- [ListModels \(p. 430\)](#)
- [ListNotebookInstanceLifecycleConfigs \(p. 433\)](#)
- [ListNotebookInstances \(p. 436\)](#)
- [ListTags \(p. 440\)](#)
- [ListTrainingJobs \(p. 442\)](#)
- [ListTrainingJobsForHyperParameterTuningJob \(p. 445\)](#)
- [ListTransformJobs \(p. 448\)](#)
- [StartNotebookInstance \(p. 451\)](#)
- [StopHyperParameterTuningJob \(p. 453\)](#)
- [StopNotebookInstance \(p. 455\)](#)
- [StopTrainingJob \(p. 457\)](#)
- [StopTransformJob \(p. 459\)](#)
- [UpdateEndpoint \(p. 461\)](#)
- [UpdateEndpointWeightsAndCapacities \(p. 463\)](#)
- [UpdateNotebookInstance \(p. 465\)](#)
- [UpdateNotebookInstanceLifecycleConfig \(p. 468\)](#)

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, models, 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 \(p. 355\)](#)

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 \(p. 553\)](#).

The request accepts the following data in JSON format.

ResourceArn (p. 347)

The Amazon Resource Name (ARN) of the resource that you want to tag.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

Tags (p. 347)

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 \(p. 535\)](#) 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. 347)

A list of tags associated with the Amazon SageMaker resource.

Type: Array of [Tag \(p. 535\)](#) 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 \(p. 551\)](#).

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 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.

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 in 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[EndpointConfigName \(p. 349\)](#)

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 \(p. 349\)](#)

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. 349)

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 \(p. 535\)](#) 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. 350)

The Amazon Resource Name (ARN) of the endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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": [  
        {  
            "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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[EndpointConfigName \(p. 352\)](#)

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 \(p. 352\)](#)

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.

Required: No

[ProductionVariants \(p. 352\)](#)

An array of `ProductionVariant` objects, one for each model that you want to host at this endpoint.

Type: Array of [ProductionVariant \(p. 523\)](#) objects

Array Members: Minimum number of 1 item.

Required: Yes

[Tags \(p. 352\)](#)

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 \(p. 535\)](#) 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 \(p. 353\)](#)

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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"  
                }  
            ],  
            "IntegerParameterRanges": [  
                {  
                    "MaxValue": "string",  
                    "MinValue": "string",  
                    "Name": "string"  
                }  
            ]  
        },  
        "ResourceLimits": {  
            "MaxNumberOfTrainingJobs": number,  
            "MaxParallelTrainingJobs": number  
        },  
        "Strategy": "string"  
    },  
    "HyperParameterTuningJobName": "string",  
    "Tags": [  
        {  
            "Key": "string",  
            "Value": "string"  
        }  
    ],  
    "TrainingJobDefinition": {  
        "AlgorithmSpecification": {  
            "MetricDefinitions": [  
                {  
                    "Name": "string",  
                    "Regex": "string"  
                }  
            ],  
            "TrainingImage": "string",  
            "TrainingInputMode": "string"  
        },  
        "InputDataConfig": [
```

```
{
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
        "S3DataSource": {
            "S3DataDistributionType": "string",
            "S3DataType": "string",
            "S3Uri": "string"
        }
    },
    "InputMode": "string",
    "RecordWrapperType": "string"
},
],
"OutputDataConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
},
"ResourceConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
},
"RoleArn": "string",
"StaticHyperParameters": {
    "string" : "string"
},
"StoppingCondition": {
    "MaxRuntimeInSeconds": 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[HyperParameterTuningJobConfig \(p. 355\)](#)

The [HyperParameterTuningJobConfig \(p. 503\)](#) 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 \(p. 50\)](#)

Type: [HyperParameterTuningJobConfig \(p. 503\)](#) object

Required: Yes

[HyperParameterTuningJobName \(p. 355\)](#)

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 \(p. 355\)](#)

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 \(p. 535\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

[TrainingJobDefinition \(p. 355\)](#)

The [HyperParameterTrainingJobDefinition \(p. 498\)](#) 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 \(p. 498\)](#) object

Required: Yes

[WarmStartConfig \(p. 355\)](#)

Specifies 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` 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 \(p. 507\)](#) 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 \(p. 357\)](#)

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 \(p. 551\)](#).

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 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

```
{  
    "ExecutionRoleArn": "string",  
    "modelName": "string",  
    "PrimaryContainer": {  
        "ContainerHostname": "string",  
        "Environment": {  
            "string" : "string"  
        },  
        "Image": "string",  
        "ModelDataUrl": "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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[ExecutionRoleArn \(p. 359\)](#)

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-zA-Z-]*:iam::\d{12}:role/[a-zA-Z0-9+=,.@-_/.]+$`

Required: Yes

[ModelName \(p. 359\)](#)

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 \(p. 359\)](#)

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 \(p. 482\)](#) object

Required: Yes

[Tags \(p. 359\)](#)

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 \(p. 535\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

[VpcConfig \(p. 359\)](#)

A [VpcConfig \(p. 551\)](#) 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 \(p. 551\)](#) 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. 360)

The ARN of the model created in Amazon SageMaker.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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).

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

```
{  
    "DirectInternetAccess": "string",  
    "InstanceType": "string",  
    "KmsKeyId": "string",  
    "LifecycleConfigName": "string",  
    "NotebookInstanceName": "string",  
    "RoleArn": "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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[DirectInternetAccess \(p. 362\)](#)

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 you 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 \(p. 362\)](#)

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 \(p. 362\)](#)

If you provide a AWS KMS key ID, Amazon SageMaker uses it to encrypt data at rest on the ML storage volume that is 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.

Required: No

[LifecycleConfigName \(p. 362\)](#)

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 \(p. 362\)](#)

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 \(p. 362\)](#)

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`) permissions 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-zA-Z-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-_/.]+\$

Required: Yes

[SecurityGroupIds \(p. 362\)](#)

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.

Required: No

[SubnetId \(p. 362\)](#)

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.

Required: No

[Tags \(p. 362\)](#)

A list of tags to associate with the notebook instance. You can add tags later by using the `CreateTags` API.

Type: Array of [Tag \(p. 535\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

[VolumeSizeInGB \(p. 362\)](#)

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. 365)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceLifecycleConfigName (p. 366)

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. 366)

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 \(p. 515\)](#) objects

Array Members: Maximum number of 1 item.

Required: No

OnStart (p. 366)

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 \(p. 515\)](#) 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. 367)

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 \(p. 551\)](#).

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 Java](#)
- [AWS SDK for JavaScript](#)

- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

CreatePresignedNotebookInstanceIdUrl

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.

You can restrict access to this API and to the URL that it returns to a list of IP addresses that you specify. To restrict access, attach an IAM policy that denies access to this API unless the call comes from an IP address in the specified list to every AWS Identity and Access Management user, group, or role used to access the notebook instance. 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](#).

Request Syntax

```
{  
    "NotebookInstanceName": "string",  
    "SessionExpirationDurationInSeconds": number  
}
```

Request Parameters

For information about the parameters that are common to all actions, see [Common Parameters \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 369)

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. 369)

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 \(p. 369\)](#)

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 \(p. 551\)](#).

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 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 deep 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 influence the quality of the final model. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see [Algorithms](#).
- **InputDataConfig** - Describes the training dataset and the Amazon S3 location where it is stored.
- **OutputDataConfig** - Identifies the Amazon S3 location 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.
- **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** - Sets a duration for training. Use this parameter to cap model training costs.

For more information about Amazon SageMaker, see [How It Works](#).

Request Syntax

```
{  
    "AlgorithmSpecification": {  
        "MetricDefinitions": [  
            {  
                "Name": "string",  
                "Regex": "string"  
            }  
        ],  
        "TrainingImage": "string",  
        "TrainingInputMode": "string"  
    },  
    "HyperParameters": {  
        "string" : "string"  
    },  
    "InputDataConfig": [  
        {  
            "ChannelName": "string",  
            "CompressionType": "string",  
            "ContentType": "string",  
            "DataSource": {  
                "S3DataSource": {  
                    "S3DataDistributionType": "string",  
                    "S3DataType": "string",  
                    "S3Uri": "string"  
                }  
            },  
            "InputMode": "string",  
            "InputProcessing": {  
                "InputFilter": "string",  
                "InputFormat": "string",  
                "InputType": "string",  
                "OutputFormat": "string",  
                "OutputType": "string",  
                "Processor": "string",  
                "ProcessorVersion": "string",  
                "SamplingRate": "string",  
                "S3Input": {  
                    "ManifestFile": "string",  
                    "ManifestType": "string",  
                    "RecordWrapperType": "string",  
                    "ShardCount": "string",  
                    "ShardSize": "string"  
                }  
            }  
        }  
    },  
    "OutputDataConfig": {  
        "KmsKeyId": "string",  
        "S3OutputPath": "string"  
    },  
    "ResourceConfig": {  
        "ClusterConfig": {  
            "InstanceCount": 123,  
            "InstanceType": "string",  
            "VolumeKmsKeyId": "string"  
        },  
        "InstanceType": "string",  
        "PreferredInstanceType": "string",  
        "VolumeSize": 123  
    },  
    "StoppingCondition": {  
        "MaxRuntimeInSeconds": 123,  
        "MaxTrainingJobs": 123,  
        "MaxTrainingTimeInHours": 123  
    }  
}
```

```

        "RecordWrapperType": "string"
    },
],
"OutputDataConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
},
"ResourceConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
},
"RoleArn": "string",
"StoppingCondition": {
    "MaxRuntimeInSeconds": 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[AlgorithmSpecification \(p. 371\)](#)

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 \(p. 477\)](#) object

Required: Yes

[HyperParameters \(p. 371\)](#)

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.

Value Length Constraints: Maximum length of 256.

Required: No

[InputDataConfig \(p. 371\)](#)

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 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.

Type: Array of [Channel \(p. 480\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: No

[OutputDataConfig \(p. 371\)](#)

Specifies the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: [OutputDataConfig \(p. 519\)](#) object

Required: Yes

[ResourceConfig \(p. 371\)](#)

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 \(p. 527\)](#) object

Required: Yes

[RoleArn \(p. 371\)](#)

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 \(p. 371\)](#)

Sets a duration for training. Use this parameter 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 might use this 120-second window to save the model artifacts.

When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job. This intermediate data is a valid model artifact. You can use it to create a model using the `CreateModel` API.

Type: [StoppingCondition \(p. 534\)](#) object

Required: Yes

[Tags \(p. 371\)](#)

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 \(p. 535\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

[TrainingJobName \(p. 371\)](#)

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 \(p. 371\)](#)

A [VpcConfig \(p. 551\)](#) 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 \(p. 551\)](#) 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 \(p. 374\)](#)

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 \(p. 551\)](#).

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 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 \(p. 359\)](#).
- **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",  
    "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,  
        "Type": "string"  
    }  
}
```

```
    "InstanceType": "string",
    "VolumeKmsKeyId": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see [Common Parameters \(p. 553\)](#).

The request accepts the following data in JSON format.

[BatchStrategy \(p. 376\)](#)

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.

Batch transform will automatically split your input data into whatever payload size is specified if you set `SplitType` to `Line` and `BatchStrategy` to `MultiRecord`. There's no need to split the dataset into smaller files or to use larger payload sizes unless the records in your dataset are very large.

Type: String

Valid Values: `MultiRecord` | `SingleRecord`

Required: No

[Environment \(p. 376\)](#)

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.

Required: No

[MaxConcurrentTransforms \(p. 376\)](#)

The maximum number of parallel requests that can be sent to each instance in a transform job. This is good for algorithms that implement multiple workers on larger instances . The default value is 1. To allow Amazon SageMaker to determine the appropriate number for `MaxConcurrentTransforms`, set the value to 0.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

[MaxPayloadInMB \(p. 376\)](#)

The maximum payload size allowed, in MB. A payload is the data portion of a record (without metadata). The value in `MaxPayloadInMB` must be greater or equal to the size of a single record. You can approximate the size of a record by dividing the size of your dataset by the number of records. Then multiply this value by the number of records you want in a mini-batch. We recommend

to enter a slightly larger value than this to ensure the records fit within the maximum payload size. 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 only works in supported algorithms. Currently, Amazon SageMaker built-in algorithms do not support this feature.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

[ModelName \(p. 376\)](#)

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 \(p. 376\)](#)

An array of key-value pairs. Adding tags is optional. For more information, see [Using Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*.

Type: Array of [Tag \(p. 535\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

[TransformInput \(p. 376\)](#)

Describes the input source and the way the transform job consumes it.

Type: [TransformInput \(p. 541\)](#) object

Required: Yes

[TransformJobName \(p. 376\)](#)

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 \(p. 376\)](#)

Describes the results of the transform job.

Type: [TransformOutput \(p. 545\)](#) object

Required: Yes

[TransformResources \(p. 376\)](#)

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: [TransformResources \(p. 547\)](#) 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 \(p. 379\)](#)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

EndpointName (p. 381)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

EndpointConfigName ([p. 383](#))

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[ModelName \(p. 384\)](#)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 385)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceLifecycleConfigName (p. 387)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

ResourceArn (p. 388)

The Amazon Resource Name (ARN) of the resource whose tags you want to delete.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

TagKeys (p. 388)

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: ^((?!aws:)[\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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

EndpointName (p. 390)

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 \(p. 390\)](#)

A timestamp that shows when the endpoint was created.

Type: Timestamp

[EndpointArn \(p. 390\)](#)

The Amazon Resource Name (ARN) of the endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

[EndpointConfigName \(p. 390\)](#)

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 \(p. 390\)](#)

Name of the endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

[EndpointStatus \(p. 390\)](#)

The status of the endpoint.

- **OutOfService**: Endpoint is not available to take incoming requests.
- **Creating**: [CreateEndpoint \(p. 349\)](#) is executing.
- **Updating**: [UpdateEndpoint \(p. 461\)](#) or [UpdateEndpointWeightsAndCapacities \(p. 463\)](#) 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 \(p. 463\)](#) call or when the [UpdateEndpointWeightsAndCapacities \(p. 463\)](#) operation is called explicitly.
- **InService**: Endpoint is available to process incoming requests.
- **Deleting**: [DeleteEndpoint \(p. 381\)](#) is executing.
- **Failed**: Endpoint could not be created, updated, or re-scaled. Use [DescribeEndpointFailureReason \(p. 392\)](#) for information about the failure. [DeleteEndpoint \(p. 381\)](#) 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 \(p. 390\)](#)

If the status of the endpoint is Failed, the reason why it failed.

Type: String

Length Constraints: Maximum length of 1024.

[LastModifiedTime \(p. 390\)](#)

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

[ProductionVariants \(p. 390\)](#)

An array of [ProductionVariantSummary \(p. 525\)](#) objects, one for each model hosted behind this endpoint.

Type: Array of [ProductionVariantSummary \(p. 525\)](#) objects

Array Members: Minimum number of 1 item.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[EndpointConfigName \(p. 393\)](#)

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": [  
        {  
            "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 \(p. 393\)](#)

A timestamp that shows when the endpoint configuration was created.

Type: Timestamp

[EndpointConfigArn \(p. 393\)](#)

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

[EndpointConfigName \(p. 393\)](#)

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 \(p. 393\)](#)

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.

[ProductionVariants \(p. 393\)](#)

An array of `ProductionVariant` objects, one for each model that you want to host at this endpoint.

Type: Array of [ProductionVariant \(p. 523\)](#) objects

Array Members: Minimum number of 1 item.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 395)

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"
        }
    ],
    "IntegerParameterRanges": [
        {
            ".MaxValue": "string",
            ".MinValue": "string",
            "Name": "string"
        }
    ]
},
"ResourceLimits": {
    "MaxNumberOfTrainingJobs": number,
    "MaxParallelTrainingJobs": number
},
"Strategy": "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": {
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },

```

```

    "InputDataConfig": [
        {
            "ChannelName": "string",
            "CompressionType": "string",
            "ContentType": "string",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "string",
                    "S3DataType": "string",
                    "S3Uri": "string"
                }
            },
            "InputMode": "string",
            "RecordWrapperType": "string"
        }
    ],
    "OutputDataConfig": {
        "KmsKeyId": "string",
        "S3OutputPath": "string"
    },
    "ResourceConfig": {
        "InstanceCount": number,
        "InstanceType": "string",
        "VolumeKmsKeyId": "string",
        "VolumeSizeInGB": number
    },
    "RoleArn": "string",
    "StaticHyperParameters": {
        "string" : "string"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 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. 395)

A [TrainingJobSummary \(p. 538\)](#) object that describes the training job that completed with the best current [HyperParameterTuningJobObjective \(p. 504\)](#).

Type: [HyperParameterTrainingJobSummary \(p. 500\)](#) object

CreationTime (p. 395)

The date and time that the tuning job started.

Type: Timestamp

FailureReason (p. 395)

If the tuning job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

HyperParameterTuningEndTime (p. 395)

The date and time that the tuning job ended.

Type: Timestamp

HyperParameterTuningJobArn (p. 395)

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. 395)

The [HyperParameterTuningJobConfig \(p. 503\)](#) object that specifies the configuration of the tuning job.

Type: [HyperParameterTuningJobConfig \(p. 503\)](#) object

HyperParameterTuningJobName (p. 395)

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. 395)

The status of the tuning job: InProgress, Completed, Failed, Stopping, or Stopped.

Type: String

Valid Values: Completed | InProgress | Failed | Stopped | Stopping

LastModifiedTime (p. 395)

The date and time that the status of the tuning job was modified.

Type: Timestamp

ObjectiveStatusCounters (p. 395)

The [ObjectiveStatusCounters \(p. 518\)](#) object that specifies the number of training jobs, categorized by the status of their final objective metric, that this tuning job launched.

Type: [ObjectiveStatusCounters \(p. 518\)](#) object

OverallBestTrainingJob (p. 395)

If the hyperparameter tuning job is an warm start tuning job with a `WarmStartType` of `IDENTICAL_DATA_AND_ALGORITHM`, this is the [TrainingJobSummary \(p. 538\)](#) 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 \(p. 500\)](#) object

TrainingJobDefinition (p. 395)

The [HyperParameterTrainingJobDefinition \(p. 498\)](#) object that specifies the definition of the training jobs that this tuning job launches.

Type: [HyperParameterTrainingJobDefinition \(p. 498\)](#) object

TrainingJobStatusCounters (p. 395)

The [TrainingJobStatusCounters \(p. 536\)](#) object that specifies the number of training jobs, categorized by status, that this tuning job launched.

Type: [TrainingJobStatusCounters \(p. 536\)](#) object

WarmStartConfig (p. 395)

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 \(p. 507\)](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[ModelName \(p. 400\)](#)

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

```
{  
    "CreationTime": number,  
    "ExecutionRoleArn": "string",  
    "ModelArn": "string",  
    "ModelName": "string",  
    "PrimaryContainer": {  
        "ContainerHostname": "string",  
        "Environment": {  
            "string" : "string"  
        },  
        "Image": "string",  
        "ModelDataURL": "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.

[CreationTime \(p. 400\)](#)

A timestamp that shows when the model was created.

Type: Timestamp

ExecutionRoleArn (p. 400)

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-zA-Z-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/-/_/]+\$

ModelArn (p. 400)

The Amazon Resource Name (ARN) of the model.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

ModelName (p. 400)

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. 400)

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 \(p. 482\)](#) object

VpcConfig (p. 400)

A [VpcConfig \(p. 551\)](#) 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 \(p. 551\)](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 403)

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

```
{  
    "CreationTime": number,  
    "DirectInternetAccess": "string",  
    "FailureReason": "string",  
    "InstanceType": "string",  
    "KmsKeyId": "string",  
    "LastModifiedTime": number,  
    "NetworkInterfaceId": "string",  
    "NotebookInstanceArn": "string",  
    "NotebookInstanceLifecycleConfigName": "string",  
    "NotebookInstanceName": "string",  
    "NotebookInstanceStatus": "string",  
    "RoleArn": "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.

CreationTime (p. 403)

A timestamp. Use this parameter to return the time when the notebook instance was created

Type: Timestamp

[DirectInternetAccess \(p. 403\)](#)

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 \(p. 403\)](#)

If status is failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

[InstanceType \(p. 403\)](#)

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 \(p. 403\)](#)

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.

[LastModifiedTime \(p. 403\)](#)

A timestamp. Use this parameter to retrieve the time when the notebook instance was last modified.

Type: Timestamp

[NetworkInterfaceId \(p. 403\)](#)

Network interface IDs that Amazon SageMaker created at the time of creating the instance.

Type: String

[NotebookInstanceArn \(p. 403\)](#)

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

[NotebookInstanceLifecycleConfigName \(p. 403\)](#)

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 \(p. 403\)](#)

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 \(p. 403\)](#)

The status of the notebook instance.

Type: String

Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting | Updating

[RoleArn \(p. 403\)](#)

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-zA-Z-]*:iam::\d{12}:role/[a-zA-Z_0-9+=,.@\\-_]+\$

[SecurityGroups \(p. 403\)](#)

The IDs of the VPC security groups.

Type: Array of strings

Array Members: Maximum number of 5 items.

Length Constraints: Maximum length of 32.

[SubnetId \(p. 403\)](#)

The ID of the VPC subnet.

Type: String

Length Constraints: Maximum length of 32.

[Url \(p. 403\)](#)

The URL that you use to connect to the Jupyter notebook that is running in your notebook instance.

Type: String

[VolumeSizeInGB \(p. 403\)](#)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceLifecycleConfigName (p. 407)

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 \(p. 407\)](#)

A timestamp that tells when the lifecycle configuration was created.

Type: Timestamp

[LastModifiedTime \(p. 407\)](#)

A timestamp that tells when the lifecycle configuration was last modified.

Type: Timestamp

[NotebookInstanceLifecycleConfigArn \(p. 407\)](#)

The Amazon Resource Name (ARN) of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 256.

[NotebookInstanceLifecycleConfigName \(p. 407\)](#)

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

[OnCreate \(p. 407\)](#)

The shell script that runs only once, when you create a notebook instance.

Type: Array of [NotebookInstanceLifecycleHook \(p. 515\)](#) objects

Array Members: Maximum number of 1 item.

[OnStart \(p. 407\)](#)

The shell script that runs every time you start a notebook instance, including when you create the notebook instance.

Type: Array of [NotebookInstanceLifecycleHook \(p. 515\)](#) objects

Array Members: Maximum number of 1 item.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

TrainingJobName (p. 410)

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": {  
        "MetricDefinitions": [  
            {  
                "Name": "string",  
                "Regex": "string"  
            }  
        ],  
        "TrainingImage": "string",  
        "TrainingInputMode": "string"  
    },  
    "CreationTime": number,  
    "FailureReason": "string",  
    "FinalMetricDataList": [  
        {  
            "MetricName": "string",  
            "Timestamp": number,  
            "Value": number  
        }  
    ],  
    "HyperParameters": {  
        "string" : "string"  
    },  
    "InputDataConfig": [  
        {  
            "ChannelName": "string",  
            "CompressionType": "string",  
            "ContentType": "string",  
            "DataSource": "string",  
            "HeaderConfig": {  
                "FileHeaderInfo": "string",  
                "RecordHeaderInfo": "string"  
            },  
            "InputFormat": "string",  
            "LabelType": "string",  
            "RecordWrapperFormat": "string",  
            "S3Input": {  
                "ManifestFileLocation": "string",  
                "RecordWrapperFormat": "string",  
                "SplitType": "string",  
                "S3Uri": "string"  
            },  
            "S3Uri": "string",  
            "SmsInput": {  
                "SmsRoleArn": "string",  
                "SmsTopicArn": "string",  
                "SmsType": "string"  
            },  
            "SmsWindowInput": {  
                "SmsRoleArn": "string",  
                "SmsTopicArn": "string",  
                "SmsWindowType": "string",  
                "SmsWindowWindowDuration": number  
            },  
            "SplitType": "string",  
            "SseSpecification": {  
                "KmsKeyArn": "string",  
                "SseType": "string"  
            },  
            "SseType": "string",  
            "SmsWindowType": "string",  
            "SmsWindowWindowDuration": number  
        }  
    ],  
    "ModelArtifacts": {  
        "ModelDataUrl": "string",  
        "ModelName": "string",  
        "ModelType": "string",  
        "SupportedContentTypes": ["string"],  
        "SupportedResponseContentTypes": ["string"]  
    },  
    "OutputDataConfig": {  
        "KmsKeyArn": "string",  
        "S3Output": {  
            "S3Uri": "string",  
            "S3UriPrefix": "string",  
            "S3UriRegion": "string",  
            "S3UriSuffix": "string"  
        },  
        "SmsWindowOutput": {  
            "SmsRoleArn": "string",  
            "SmsTopicArn": "string",  
            "SmsWindowType": "string",  
            "SmsWindowWindowDuration": number  
        }  
    },  
    "TrainingJobStatus": "string",  
    "TrainingJobType": "string",  
    "TrainingJobArn": "string",  
    "LastModified": "string",  
    "LastModifiedBy": "string",  
    "LastModifiedAt": "string",  
    "LastModifiedAtEpochTime": number  
}
```

```

    "DataSource": {
        "S3DataSource": {
            "S3DataDistributionType": "string",
            "S3DataType": "string",
            "S3Uri": "string"
        }
    },
    "InputMode": "string",
    "RecordWrapperType": "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
},
"TrainingEndTime": number,
"TrainingJobArn": "string",
"TrainingJobName": "string",
"TrainingJobStatus": "string",
"TrainingStartTime": number,
"TrainingJobArn": "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 \(p. 410\)](#)

Information about the algorithm used for training, and algorithm metadata.

Type: [AlgorithmSpecification \(p. 477\)](#) object

[CreationTime \(p. 410\)](#)

A timestamp that indicates when the training job was created.

Type: Timestamp

FailureReason (p. 410)

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

FinalMetricDataList (p. 410)

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 \(p. 510\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

HyperParameters (p. 410)

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

InputDataConfig (p. 410)

An array of `Channel` objects that describes each data input channel.

Type: Array of [Channel \(p. 480\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

LastModifiedTime (p. 410)

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

ModelArtifacts (p. 410)

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: [ModelArtifacts \(p. 512\)](#) object

OutputDataConfig (p. 410)

The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.

Type: [OutputDataConfig \(p. 519\)](#) object

ResourceConfig (p. 410)

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: [ResourceConfig \(p. 527\)](#) object

RoleArn (p. 410)

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-zA-Z-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-_/.]+\$

[SecondaryStatus \(p. 410\)](#)

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 \(p. 532\)](#).

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

[SecondaryStatusTransitions \(p. 410\)](#)

A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of [SecondaryStatusTransition \(p. 532\)](#) objects

[StoppingCondition \(p. 410\)](#)

The condition under which to stop the training job.

Type: [StoppingCondition \(p. 534\)](#) object

[TrainingEndTime \(p. 410\)](#)

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 \(p. 410\)](#)

The Amazon Resource Name (ARN) of the training job.

Type: `String`

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-zA-Z-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:training-job/.*`

[TrainingJobName \(p. 410\)](#)

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 \(p. 410\)](#)

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 \(p. 410\)](#)

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`

[TuningJobArn \(p. 410\)](#)

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. 410)

A [VpcConfig \(p. 551\)](#) 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 \(p. 551\)](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

ResourceNotFound

Resource being accessed 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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

TransformJobName (p. 416)

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,  
    "Environment": {  
        "string" : "string"  
    },  
    "FailureReason": "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"  
    }  
}
```

```
        "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 \(p. 416\)](#)

SingleRecord means only one record was used per a batch. MultiRecord means batches contained as many records that could possibly fit within the MaxPayloadInMB limit.

Type: String

Valid Values: MultiRecord | SingleRecord

[CreationTime \(p. 416\)](#)

A timestamp that shows when the transform Job was created.

Type: Timestamp

[Environment \(p. 416\)](#)

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.

[FailureReason \(p. 416\)](#)

If the transform job failed, the reason that it failed.

Type: String

Length Constraints: Maximum length of 1024.

[MaxConcurrentTransforms \(p. 416\)](#)

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 \(p. 416\)](#)

The maximum payload size , in MB used in the transform job.

Type: Integer

Valid Range: Minimum value of 0.

[ModelName \(p. 416\)](#)

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 \(p. 416\)](#)

Indicates when the transform job is Completed, Stopped, or Failed. You are billed for the time interval between this time and the value of `TransformStartTime`.

Type: Timestamp

[TransformInput \(p. 416\)](#)

Describes the dataset to be transformed and the Amazon S3 location where it is stored.

Type: [TransformInput \(p. 541\)](#) object

[TransformJobArn \(p. 416\)](#)

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 \(p. 416\)](#)

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 \(p. 416\)](#)

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 \(p. 416\)](#)

Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.

Type: [TransformOutput \(p. 545\)](#) object

[TransformResources \(p. 416\)](#)

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: [TransformResources \(p. 547\)](#) object

[TransformStartTime \(p. 416\)](#)

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 \(p. 551\)](#).

ResourceNotFound

Resource being accessed 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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 420\)](#)

A filter that returns only endpoint configurations created after the specified time (timestamp).

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 420\)](#)

A filter that returns only endpoint configurations created before the specified time (timestamp).

Type: Timestamp

Required: No

[MaxResults \(p. 420\)](#)

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 \(p. 420\)](#)

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

[NextToken \(p. 420\)](#)

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.

Required: No

[SortBy \(p. 420\)](#)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name` | `CreationTime`

Required: No

[SortOrder \(p. 420\)](#)

The sort order for results. The default is `Ascending`.

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.

[EndpointConfigs \(p. 421\)](#)

An array of endpoint configurations.

Type: Array of [EndpointConfigSummary \(p. 493\)](#) objects

[NextToken \(p. 421\)](#)

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.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 423\)](#)

A filter that returns only endpoints that were created after the specified time (timestamp).

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 423\)](#)

A filter that returns only endpoints that were created before the specified time (timestamp).

Type: Timestamp

Required: No

[LastModifiedTimeAfter \(p. 423\)](#)

A filter that returns only endpoints that were modified after the specified timestamp.

Type: Timestamp

Required: No

[LastModifiedTimeBefore \(p. 423\)](#)

A filter that returns only endpoints that were modified before the specified timestamp.

Type: Timestamp

Required: No

[MaxResults \(p. 423\)](#)

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 \(p. 423\)](#)

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 \(p. 423\)](#)

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.

Required: No

[SortBy \(p. 423\)](#)

Sorts the list of results. The default is `CreationTime`.

Type: String

Valid Values: `Name` | `CreationTime` | `Status`

Required: No

[SortOrder \(p. 423\)](#)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending` | `Descending`

Required: No

[StatusEquals \(p. 423\)](#)

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 \(p. 424\)](#)

An array or endpoint objects.

Type: Array of [EndpointSummary \(p. 494\)](#) objects

[NextToken \(p. 424\)](#)

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.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 505\)](#) 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 426\)](#)

A filter that returns only tuning jobs that were created after the specified time.

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 426\)](#)

A filter that returns only tuning jobs that were created before the specified time.

Type: Timestamp

Required: No

[LastModifiedTimeAfter \(p. 426\)](#)

A filter that returns only tuning jobs that were modified after the specified time.

Type: Timestamp

Required: No

[LastModifiedTimeBefore \(p. 426\)](#)

A filter that returns only tuning jobs that were modified before the specified time.

Type: Timestamp

Required: No

[MaxResults \(p. 426\)](#)

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 \(p. 426\)](#)

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 \(p. 426\)](#)

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.

Required: No

[SortBy \(p. 426\)](#)

The field to sort results by. The default is `Name`.

Type: String

Valid Values: `Name` | `Status` | `CreationTime`

Required: No

[SortOrder \(p. 426\)](#)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending` | `Descending`

Required: No

[StatusEquals \(p. 426\)](#)

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 \(p. 427\)](#)

A list of [HyperParameterTuningJobSummary \(p. 505\)](#) objects that describe the tuning jobs that the `ListHyperParameterTuningJobs` request returned.

Type: Array of [HyperParameterTuningJobSummary \(p. 505\)](#) objects

[NextToken \(p. 427\)](#)

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.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 430\)](#)

A filter that returns only models created after the specified time (timestamp).

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 430\)](#)

A filter that returns only models created before the specified time (timestamp).

Type: Timestamp

Required: No

[MaxResults \(p. 430\)](#)

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 \(p. 430\)](#)

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 \(p. 430\)](#)

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.

Required: No

[SortBy \(p. 430\)](#)

Sorts the list of results. The default is `CreationTime`.

Type: String

Valid Values: `Name` | `CreationTime`

Required: No

[SortOrder \(p. 430\)](#)

The sort order for results. The default is `Ascending`.

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 \(p. 431\)](#)

An array of `ModelSummary` objects, each of which lists a model.

Type: Array of [ModelSummary \(p. 513\)](#) objects

[NextToken \(p. 431\)](#)

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.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 366\)](#) 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 433\)](#)

A filter that returns only lifecycle configurations that were created after the specified time (timestamp).

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 433\)](#)

A filter that returns only lifecycle configurations that were created before the specified time (timestamp).

Type: Timestamp

Required: No

[LastModifiedTimeAfter \(p. 433\)](#)

A filter that returns only lifecycle configurations that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

[LastModifiedTimeBefore \(p. 433\)](#)

A filter that returns only lifecycle configurations that were modified before the specified time (timestamp).

Type: Timestamp

Required: No

[MaxResults \(p. 433\)](#)

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 \(p. 433\)](#)

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 \(p. 433\)](#)

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.

Required: No

[SortBy \(p. 433\)](#)

Sorts the list of results. The default is `CreationTime`.

Type: String

Valid Values: `Name` | `CreationTime` | `LastModifiedTime`

Required: No

[SortOrder \(p. 433\)](#)

The sort order for results.

Type: String

Valid Values: `Ascending` | `Descending`

Required: No

Response Syntax

```
{  
    "NextToken": "string",  
    "NotebookInstanceLifecycleConfigs": [  
        {  
            "CreationTime": number,  
            "LastModifiedTime": number,  
            "Name": "string",  
            "Status": "Active" | "InService" | "Pending" | "Failed" | "Deleting",  
            "Type": "string"  
        }  
    ]  
}
```

```
        "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 \(p. 434\)](#)

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.

[NotebookInstanceLifecycleConfigs \(p. 434\)](#)

An array of `NotebookInstanceLifecycleConfiguration` objects, each listing a lifecycle configuration.

Type: Array of [NotebookInstanceLifecycleConfigSummary \(p. 514\)](#) objects

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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

```
{  
    "CreationTimeAfter": number,  
    "CreationTimeBefore": number,  
    "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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 436\)](#)

A filter that returns only notebook instances that were created after the specified time (timestamp).

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 436\)](#)

A filter that returns only notebook instances that were created before the specified time (timestamp).

Type: Timestamp

Required: No

[LastModifiedTimeAfter \(p. 436\)](#)

A filter that returns only notebook instances that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

[LastModifiedTimeBefore \(p. 436\)](#)

A filter that returns only notebook instances that were modified before the specified time (timestamp).

Type: Timestamp

Required: No

[MaxResults \(p. 436\)](#)

The maximum number of notebook instances to return.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

[NameContains \(p. 436\)](#)

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 \(p. 436\)](#)

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.

Required: No

[NotebookInstanceLifecycleConfigNameContains \(p. 436\)](#)

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 \(p. 436\)](#)

The field to sort results by. The default is `Name`.

Type: String

Valid Values: `Name` | `CreationTime` | `Status`

Required: No

[SortOrder \(p. 436\)](#)

The sort order for results.

Type: String

Valid Values: Ascending | Descending

Required: No

[StatusEquals \(p. 436\)](#)

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": [  
        {  
            "CreationTime": number,  
            "InstanceType": "string",  
            "LastModifiedTime": number,  
            "NotebookInstanceArn": "string",  
            "NotebookInstanceLifecycleConfigName": "string",  
            "NotebookInstanceName": "string",  
            "NotebookInstanceState": "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 \(p. 438\)](#)

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.

[NotebookInstances \(p. 438\)](#)

An array of `NotebookInstanceSummary` objects, one for each notebook instance.

Type: Array of [NotebookInstanceSummary \(p. 516\)](#) objects

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[MaxResults \(p. 440\)](#)

Maximum number of tags to return.

Type: Integer

Valid Range: Minimum value of 50.

Required: No

[NextToken \(p. 440\)](#)

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.

Required: No

[ResourceArn \(p. 440\)](#)

The Amazon Resource Name (ARN) of the resource whose tags you want to retrieve.

Type: String

Length Constraints: Maximum length of 256.

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 \(p. 440\)](#)

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.

[Tags \(p. 440\)](#)

An array of Tag objects, each with a tag key and a value.

Type: Array of [Tag \(p. 535\)](#) 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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 442\)](#)

A filter that returns only training jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 442\)](#)

A filter that returns only training jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

[LastModifiedTimeAfter \(p. 442\)](#)

A filter that returns only training jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

[LastModifiedTimeBefore \(p. 442\)](#)

A filter that returns only training jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

[MaxResults \(p. 442\)](#)

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 \(p. 442\)](#)

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 \(p. 442\)](#)

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.

Required: No

[SortBy \(p. 442\)](#)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name` | `CreationTime` | `Status`

Required: No

[SortOrder \(p. 442\)](#)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending` | `Descending`

Required: No

[StatusEquals \(p. 442\)](#)

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 \(p. 443\)](#)

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.

[TrainingJobSummaries \(p. 443\)](#)

An array of `TrainingJobSummary` objects, each listing a training job.

Type: Array of [TrainingJobSummary \(p. 538\)](#) objects

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 538\)](#) 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[HyperParameterTuningJobName \(p. 445\)](#)

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 \(p. 445\)](#)

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 \(p. 445\)](#)

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.

Required: No

[SortBy \(p. 445\)](#)

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 \(p. 445\)](#)

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: `Ascending` | `Descending`

Required: No

[StatusEquals \(p. 445\)](#)

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 \(p. 446\)](#)

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.

[TrainingJobSummaries \(p. 446\)](#)

A list of [TrainingJobSummary \(p. 538\)](#) objects that describe the training jobs that the `ListTrainingJobsForHyperParameterTuningJob` request returned.

Type: Array of [HyperParameterTrainingJobSummary \(p. 500\)](#) objects

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[CreationTimeAfter \(p. 448\)](#)

A filter that returns only transform jobs created after the specified time.

Type: Timestamp

Required: No

[CreationTimeBefore \(p. 448\)](#)

A filter that returns only transform jobs created before the specified time.

Type: Timestamp

Required: No

[LastModifiedTimeAfter \(p. 448\)](#)

A filter that returns only transform jobs modified after the specified time.

Type: Timestamp

Required: No

[LastModifiedTimeBefore \(p. 448\)](#)

A filter that returns only transform jobs modified before the specified time.

Type: Timestamp

Required: No

[MaxResults \(p. 448\)](#)

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 \(p. 448\)](#)

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 \(p. 448\)](#)

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.

Required: No

[SortBy \(p. 448\)](#)

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: `Name` | `CreationTime` | `Status`

Required: No

[SortOrder \(p. 448\)](#)

The sort order for results. The default is `Descending`.

Type: String

Valid Values: `Ascending` | `Descending`

Required: No

[StatusEquals \(p. 448\)](#)

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 \(p. 449\)](#)

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.

[TransformJobSummaries \(p. 449\)](#)

An array of `TransformJobSummary` objects.

Type: Array of [TransformJobSummary \(p. 543\)](#) objects

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 451)

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 453)

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 \(p. 551\)](#).

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 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.

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 \(p. 553\)](#).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 455)

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 \(p. 551\)](#).

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 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.

Training algorithms provided by Amazon SageMaker save the intermediate results of a model training job. This intermediate data is a valid model artifact. You can use the model artifacts that are saved when Amazon SageMaker stops a training job to create a model.

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 \(p. 553\)](#).

The request accepts the following data in JSON format.

TrainingJobName ([p. 457](#))

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 \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

TransformJobName (p. 459)

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 \(p. 551\)](#).

ResourceNotFoundException

Resource being accessed 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 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 cannot update an endpoint with the current `EndpointConfig`. 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[EndpointConfigName \(p. 461\)](#)

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 \(p. 461\)](#)

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. 461)

The Amazon Resource Name (ARN) of the endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

DesiredWeightsAndCapacities (p. 463)

An object that provides new capacity and weight values for a variant.

Type: Array of [DesiredWeightAndCapacity \(p. 492\)](#) objects

Array Members: Minimum number of 1 item.

Required: Yes

EndpointName (p. 463)

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 \(p. 463\)](#)

The Amazon Resource Name (ARN) of the updated endpoint.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Errors

For information about the errors that are common to all actions, see [Common Errors \(p. 551\)](#).

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 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. You can also update the VPC security groups.

Request Syntax

```
{  
    "DisassociateLifecycleConfig": boolean,  
    "InstanceType": "string",  
    "LifecycleConfigName": "string",  
    "NotebookInstanceName": "string",  
    "RoleArn": "string",  
    "VolumeSizeInGB": number  
}
```

Request Parameters

For information about the parameters that are common to all actions, see [Common Parameters \(p. 553\)](#).

The request accepts the following data in JSON format.

[DisassociateLifecycleConfig \(p. 465\)](#)

Set to true to remove the notebook instance lifecycle configuration currently associated with the notebook instance.

Type: Boolean

Required: No

[InstanceType \(p. 465\)](#)

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 \(p. 465\)](#)

The name of a lifecycle configuration to associate with the notebook instance. For information about lifecycle 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 \(p. 465\)](#)

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 \(p. 465\)](#)

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-zA-Z-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-_/_]+\$

Required: No

[VolumeSizeInGB \(p. 465\)](#)

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 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 \(p. 551\)](#).

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 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 \(p. 366\)](#) 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 \(p. 553\)](#).

The request accepts the following data in JSON format.

[NotebookInstanceLifecycleConfigName \(p. 468\)](#)

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 \(p. 468\)](#)

The shell script that runs only once, when you create a notebook instance

Type: Array of [NotebookInstanceLifecycleHook \(p. 515\)](#) objects

Array Members: Maximum number of 1 item.

Required: No

[OnStart \(p. 468\)](#)

The shell script that runs every time you start a notebook instance, including when you create the notebook instance.

Type: Array of [NotebookInstanceLifecycleHook \(p. 515\)](#) 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 \(p. 551\)](#).

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 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 \(p. 470\)](#)

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 \(p. 470\)](#)

The desired MIME type of the inference in the response.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

[ContentType \(p. 470\)](#)

The MIME type of the input data in the request body.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

[CustomAttributes \(p. 470\)](#)

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. 470)

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. 470)

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. 471)

The MIME type of the inference returned in the response body.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

CustomAttributes (p. 471)

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). 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 \(p. 471\)](#)

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 \(p. 471\)](#)

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 \(p. 551\)](#).

InternalFailure

An internal failure occurred.

HTTP Status Code: 500

ModelError

Model (owned by the customer in the container) returned an error 500.

HTTP Status Code: 424

ServiceUnavailable

The service is unavailable. Try your call again.

HTTP Status Code: 503

ValidationErrorResponse

Inspect your request and try again.

HTTP Status Code: 400

Example

Pass a trace ID in the `CustomAttributes` of a request and return it in the `CustomAttributes` 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,
# 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 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 \(p. 477\)](#)
- [CategoricalParameterRange \(p. 479\)](#)

- [Channel \(p. 480\)](#)
- [ContainerDefinition \(p. 482\)](#)
- [ContinuousParameterRange \(p. 484\)](#)
- [DataSource \(p. 485\)](#)
- [DeployedImage \(p. 486\)](#)
- [DescribeTrainingJobResponse \(p. 487\)](#)
- [DesiredWeightAndCapacity \(p. 492\)](#)
- [EndpointConfigSummary \(p. 493\)](#)
- [EndpointSummary \(p. 494\)](#)
- [FinalHyperParameterTuningJobObjectiveMetric \(p. 496\)](#)
- [HyperParameterAlgorithmSpecification \(p. 497\)](#)
- [HyperParameterTrainingJobDefinition \(p. 498\)](#)
- [HyperParameterTrainingJobSummary \(p. 500\)](#)
- [HyperParameterTuningJobConfig \(p. 503\)](#)
- [HyperParameterTuningJobObjective \(p. 504\)](#)
- [HyperParameterTuningJobSummary \(p. 505\)](#)
- [HyperParameterTuningJobWarmStartConfig \(p. 507\)](#)
- [IntegerParameterRange \(p. 509\)](#)
- [MetricData \(p. 510\)](#)
- [MetricDefinition \(p. 511\)](#)
- [ModelArtifacts \(p. 512\)](#)
- [ModelSummary \(p. 513\)](#)
- [NotebookInstanceLifecycleConfigSummary \(p. 514\)](#)
- [NotebookInstanceLifecycleHook \(p. 515\)](#)
- [NotebookInstanceSummary \(p. 516\)](#)
- [ObjectiveStatusCounters \(p. 518\)](#)
- [OutputDataConfig \(p. 519\)](#)
- [ParameterRanges \(p. 521\)](#)
- [ParentHyperParameterTuningJob \(p. 522\)](#)
- [ProductionVariant \(p. 523\)](#)
- [ProductionVariantSummary \(p. 525\)](#)
- [ResourceConfig \(p. 527\)](#)
- [ResourceLimits \(p. 529\)](#)
- [S3DataSource \(p. 530\)](#)
- [SecondaryStatusTransition \(p. 532\)](#)
- [StoppingCondition \(p. 534\)](#)
- [Tag \(p. 535\)](#)
- [TrainingJobStatusCounters \(p. 536\)](#)
- [TrainingJobSummary \(p. 538\)](#)
- [TransformDataSource \(p. 540\)](#)
- [TransformInput \(p. 541\)](#)
- [TransformJobSummary \(p. 543\)](#)
- [TransformOutput \(p. 545\)](#)
- [TransformResources \(p. 547\)](#)
- [TransformS3DataSource \(p. 549\)](#)
- [VpcConfig \(p. 551\)](#)

The following data types are supported by Amazon SageMaker Runtime:

Amazon SageMaker Service

The following data types are supported by Amazon SageMaker Service:

- [AlgorithmSpecification \(p. 477\)](#)
- [CategoricalParameterRange \(p. 479\)](#)
- [Channel \(p. 480\)](#)
- [ContainerDefinition \(p. 482\)](#)
- [ContinuousParameterRange \(p. 484\)](#)
- [DataSource \(p. 485\)](#)
- [DeployedImage \(p. 486\)](#)
- [DescribeTrainingJobResponse \(p. 487\)](#)
- [DesiredWeightAndCapacity \(p. 492\)](#)
- [EndpointConfigSummary \(p. 493\)](#)
- [EndpointSummary \(p. 494\)](#)
- [FinalHyperParameterTuningJobObjectiveMetric \(p. 496\)](#)
- [HyperParameterAlgorithmSpecification \(p. 497\)](#)
- [HyperParameterTrainingJobDefinition \(p. 498\)](#)
- [HyperParameterTrainingJobSummary \(p. 500\)](#)
- [HyperParameterTuningJobConfig \(p. 503\)](#)
- [HyperParameterTuningJobObjective \(p. 504\)](#)
- [HyperParameterTuningJobSummary \(p. 505\)](#)
- [HyperParameterTuningJobWarmStartConfig \(p. 507\)](#)
- [IntegerParameterRange \(p. 509\)](#)
- [MetricData \(p. 510\)](#)
- [MetricDefinition \(p. 511\)](#)
- [ModelArtifacts \(p. 512\)](#)
- [ModelSummary \(p. 513\)](#)
- [NotebookInstanceLifecycleConfigSummary \(p. 514\)](#)
- [NotebookInstanceLifecycleHook \(p. 515\)](#)
- [NotebookInstanceSummary \(p. 516\)](#)
- [ObjectiveStatusCounters \(p. 518\)](#)
- [OutputDataConfig \(p. 519\)](#)
- [ParameterRanges \(p. 521\)](#)
- [ParentHyperParameterTuningJob \(p. 522\)](#)
- [ProductionVariant \(p. 523\)](#)
- [ProductionVariantSummary \(p. 525\)](#)
- [ResourceConfig \(p. 527\)](#)
- [ResourceLimits \(p. 529\)](#)
- [S3DataSource \(p. 530\)](#)
- [SecondaryStatusTransition \(p. 532\)](#)
- [StoppingCondition \(p. 534\)](#)
- [Tag \(p. 535\)](#)
- [TrainingJobStatusCounters \(p. 536\)](#)

- [TrainingJobSummary \(p. 538\)](#)
- [TransformDataSource \(p. 540\)](#)
- [TransformInput \(p. 541\)](#)
- [TransformJobSummary \(p. 543\)](#)
- [TransformOutput \(p. 545\)](#)
- [TransformResources \(p. 547\)](#)
- [TransformS3DataSource \(p. 549\)](#)
- [VpcConfig \(p. 551\)](#)

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

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 \(p. 511\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 20 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](#).

Type: String

Length Constraints: Maximum length of 255.

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 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.

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.

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 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.

Required: No

DataSource

The location of the channel data.

Type: [DataSource \(p. 485\)](#) 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 \(p. 477\)](#) 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 which 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

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 Java](#)
- [AWS SDK for Ruby V2](#)

ContainerDefinition

Service: Amazon SageMaker Service

Describes the container, as part of model definition.

Contents

ContainerHostname

The DNS host name for the container after Amazon SageMaker deploys it.

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.

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

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).

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*.

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 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.

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.

Required: Yes

Name

The name of the continuous hyperparameter to tune.

Type: String

Length Constraints: Maximum length of 256.

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 Java](#)
- [AWS SDK for Ruby V2](#)

DataSource

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: [S3DataSource \(p. 530\)](#) 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 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 \(p. 523\)](#).

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 Java](#)
- [AWS SDK for Ruby V2](#)

DescribeTrainingJobResponse

Service: Amazon SageMaker Service

Contents

AlgorithmSpecification

Information about the algorithm used for training, and algorithm metadata.

Type: [AlgorithmSpecification \(p. 477\)](#) object

Required: Yes

CreationTime

A timestamp that indicates when the training job was created.

Type: Timestamp

Required: Yes

FailureReason

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

FinalMetricDataList

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 \(p. 510\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

HyperParameters

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

Required: No

InputDataConfig

An array of `Channel` objects that describes each data input channel.

Type: Array of [Channel \(p. 480\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

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 \(p. 512\)](#) object

Required: Yes

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 \(p. 519\)](#) object

Required: No

ResourceConfig

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: [ResourceConfig \(p. 527\)](#) object

Required: Yes

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-zA-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 on the secondary status of the training job, see [StatusMessage](#) under [SecondaryStatusTransition \(p. 532\)](#).

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

Required: Yes

SecondaryStatusTransitions

A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of [SecondaryStatusTransition \(p. 532\)](#) objects

Required: No

StoppingCondition

The condition under which to stop the training job.

Type: [StoppingCondition \(p. 534\)](#) object

Required: Yes

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: Yes

TrainingJobName

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])*

Required: Yes

TrainingJobStatus

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`

Required: Yes

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-zA-Z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

Required: No

VpcConfig

A [VpcConfig \(p. 551\)](#) 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 \(p. 551\)](#) 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 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 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.

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 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.

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 \(p. 349\)](#) is executing.
- **Updating**: [UpdateEndpoint \(p. 461\)](#) or [UpdateEndpointWeightsAndCapacities \(p. 463\)](#) is executing.
- **SystemUpdating**: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This mainenance 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 \(p. 463\)](#) call or when the [UpdateEndpointWeightsAndCapacities \(p. 463\)](#) operation is called explicitly.
- **InService**: Endpoint is available to process incoming requests.
- **Deleting**: [DeleteEndpoint \(p. 381\)](#) is executing.
- **Failed**: Endpoint could not be created, updated, or re-scaled. Use [DescribeEndpoint:FailureReason \(p. 392\)](#) for information about the failure. [DeleteEndpoint \(p. 381\)](#) is the only operation that can be performed on a failed endpoint.

To get a list of endpoints with a specified status, use the [ListEndpoints>StatusEquals \(p. 424\)](#) 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 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 \(p. 503\)](#).

Contents

MetricName

The name of the objective metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

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 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

MetricDefinitions

An array of [MetricDefinition \(p. 511\)](#) objects that specify the metrics that the algorithm emits.

Type: Array of [MetricDefinition \(p. 511\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 20 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](#).

Type: String

Length Constraints: Maximum length of 255.

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 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 \(p. 497\)](#) object that specifies the algorithm to use for the training jobs that the tuning job launches.

Type: [HyperParameterAlgorithmSpecification \(p. 497\)](#) object

Required: Yes

InputDataConfig

An array of [Channel \(p. 480\)](#) objects that specify the input for the training jobs that the tuning job launches.

Type: Array of [Channel \(p. 480\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 8 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 \(p. 519\)](#) 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 \(p. 527\)](#) 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.

Value Length Constraints: Maximum length of 256.

Required: No

StoppingCondition

Sets a maximum duration for the training jobs that the tuning job launches. Use this parameter to limit model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal. This delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.

When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job.

Type: [StoppingCondition \(p. 534\)](#) object

Required: Yes

VpcConfig

The [VpcConfig \(p. 551\)](#) 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 \(p. 551\)](#) 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 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 \(p. 496\)](#) object that specifies the value of the objective metric of the tuning job that launched this training job.

Type: [FinalHyperParameterTuningJobObjectiveMetric \(p. 496\)](#) 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

The date and time that the training job ended.

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.

Value Length Constraints: Maximum length of 256.

Required: Yes

TuningJobName

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 Java](#)
- [AWS SDK for Ruby V2](#)

HyperParameterTuningJobConfig

Service: Amazon SageMaker Service

Configures a hyperparameter tuning job.

Contents

HyperParameterTuningJobObjective

The [HyperParameterTuningJobObjective \(p. 504\)](#) object that specifies the objective metric for this tuning job.

Type: [HyperParameterTuningJobObjective \(p. 504\)](#) object

Required: Yes

ParameterRanges

The [ParameterRanges \(p. 521\)](#) object that specifies the ranges of hyperparameters that this tuning job searches.

Type: [ParameterRanges \(p. 521\)](#) object

Required: Yes

ResourceLimits

The [ResourceLimits \(p. 529\)](#) object that specifies the maximum number of training jobs and parallel training jobs for this tuning job.

Type: [ResourceLimits \(p. 529\)](#) object

Required: Yes

Strategy

Specifies the search strategy for hyperparameters. Currently, the only valid value is `Bayesian`.

Type: String

Valid Values: `Bayesian`

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 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.

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 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 \(p. 518\)](#) object that specifies the numbers of training jobs, categorized by objective metric status, that this tuning job launched.

Type: [ObjectiveStatusCounters \(p. 518\)](#) object

Required: Yes

ResourceLimits

The [ResourceLimits \(p. 529\)](#) object that specifies the maximum number of training jobs and parallel training jobs allowed for this tuning job.

Type: [ResourceLimits \(p. 529\)](#) 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`

Required: Yes

TrainingJobStatusCounters

The [TrainingJobStatusCounters \(p. 536\)](#) object that specifies the numbers of training jobs, categorized by status, that this tuning job launched.

Type: [TrainingJobStatusCounters \(p. 536\)](#) 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 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 \(p. 522\)](#) 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 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.

Required: Yes

.MinValue

The minimum value of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

Name

The name of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

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 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.

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 Java](#)
- [AWS SDK for Ruby V2](#)

MetricDefinition

Service: Amazon SageMaker Service

Specifies a metric that the training algorithm writes to `stderr` or `stdout`. Amazon SageMaker hyperparameter 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.

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.

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 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 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.

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 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 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.

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 Java](#)
- [AWS SDK for Ruby V2](#)

NotebookInstanceSummary

Service: Amazon SageMaker Service

Provides summary information for an Amazon SageMaker notebook instance.

Contents

CreationTime

A timestamp that shows when the notebook instance was created.

Type: Timestamp

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 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 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 don't provide the 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 *Amazon Simple Storage Service Developer Guide*.

Note

The KMS key policy must grant permission to the IAM role that you specify in your `CreateTrainingJob` request. [Using Key Policies in AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

Type: String

Length Constraints: Maximum length of 2048.

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 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 \(p. 479\)](#) objects that specify ranges of categorical hyperparameters that a hyperparameter tuning job searches.

Type: Array of [CategoricalParameterRange \(p. 479\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

ContinuousParameterRanges

The array of [ContinuousParameterRange \(p. 484\)](#) objects that specify ranges of continuous hyperparameters that a hyperparameter tuning job searches.

Type: Array of [ContinuousParameterRange \(p. 484\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

IntegerParameterRanges

The array of [IntegerParameterRange \(p. 509\)](#) objects that specify ranges of integer hyperparameters that a hyperparameter tuning job searches.

Type: Array of [IntegerParameterRange \(p. 509\)](#) 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 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 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

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 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 [DeployedImage](#) (p. 486) 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 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

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.

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 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 Java](#)
- [AWS SDK for Ruby V2](#)

S3DataSource

Service: Amazon SageMaker Service

Describes the S3 data source.

Contents

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 with 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.

Type: String

Valid Values: `ManifestFile` | `S3Prefix`

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 constitutes the input data for the channel for this datasource. The object that each s3urIs points to must 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 Java](#)
- [AWS SDK for Ruby V2](#)

SecondaryStatusTransition

Service: Amazon SageMaker Service

An array element of [DescribeTrainingJob:SecondaryStatusTransitions \(p. 413\)](#). 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`

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 [DescribeTrainingJobResponse \(p. 487\)](#), 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 Java](#)
- [AWS SDK for Ruby V2](#)

StoppingCondition

Service: Amazon SageMaker Service

Specifies how long model training can run. When model training reaches the limit, Amazon SageMaker ends the training job. Use this API to cap model training cost.

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 training is not lost.

Training algorithms provided by Amazon SageMaker automatically saves the intermediate results of a model training job (it is best effort case, as model might not be ready to save as some stages, for example training just started). This intermediate data is a valid model artifact. You can use it to create a model (`CreateModel`).

Contents

MaxRuntimeInSeconds

The maximum length of time, in seconds, that the training job can run. If model training does not complete during this time, Amazon SageMaker ends the job. If value is not specified, default value is 1 day. Maximum value is 5 days.

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 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: `^((?!aws:)[\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 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 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-zA-Z-]*:sagemaker:[a-zA-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: `^[\w\-\._]+(\-*[\w\-\._]+)*$`

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 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 \(p. 549\)](#) 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 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

Compressing data helps save on storage space. 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.

Required: No

DataSource

Describes the location of the channel data, meaning the S3 location of the input data that the model can consume.

Type: [TransformDataSource \(p. 540\)](#) object

Required: Yes

SplitType

The method to use to split the transform job's data into smaller batches. The default value is `None`. If you don't want to split the data, specify `None`. If you want to split records on a newline character boundary, specify `Line`. To split records according to the RecordIO format, specify `RecordIO`.

Amazon SageMaker will send maximum number of records per batch in each request up to the `MaxPayloadInMB` limit. For more information, see [RecordIO data format](#).

Note

For information about the RecordIO format, see [Data Format](#).

Type: String

Valid Values: `None` | `Line` | `RecordIO`

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 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 calling [ListTransformJobs \(p. 448\)](#).

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 Java](#)
- [AWS SDK for Ruby V2](#)

TransformOutput

Service: Amazon SageMaker Service

Describes the results of a transform job output.

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.

Required: No

AssembleWith

Defines how to assemble the results of the transform job as a single S3 object. You should select 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 `CreateTransformJob` 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.

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, the transformed data is stored in a corresponding subfolder in the location under the output prefix. For example, the input data s3://bucket-name/input-name-prefix/dataset01/data.csv will have the transformed data stored at s3://bucket-name/key-name-prefix/dataset01/, based on the original name, as a series of .part files (.part0001, part0002, etc).

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 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, provide 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. For using built-in algorithms to transform moderately sized datasets, `ml.m4.xlarge` or `ml.m5.large` should suffice. There is no default value for `InstanceType`.

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.

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 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.

Type: String

Valid Values: `ManifestFile` | `S3Prefix`

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 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-xxxxxxx. 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.

Required: Yes

Subnets

The ID of the subnets in the VPC to which you want to connect your training job or model.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 16 items.

Length Constraints: Maximum length of 32.

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 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

ValidationException

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
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. 556)	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 <code>KmsKeyId</code> request parameter in a call to <code>CreateEndpointConfig</code> . You can specify an AWS KMS key that Amazon SageMaker uses to encrypt training model artifacts at rest by setting the <code>KmsKeyId</code> field of the <code>OutputDataConfig</code> 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*.