

Understanding the Flight Delay Data

Introduction

This Lab uses United States domestic flight data to forecast the arrival delay of upcoming flights. The value being predicted is the arrival delay in minutes. This is an example of a regression problem because what is being predicted is a continuous variable, as opposed to a discrete class. If the task was to predict whether or not an upcoming flight would be delayed on arrival, that would be an example of a classification problem. Amazon SageMaker can handle both types of problems. In this Lab, you will use regression because you want to know how much flights will be delayed not just if they will be delayed.

Data used by Amazon SageMaker must be stored in Amazon S3. SageMaker can directly use comma-separated value (CSV) or [recordIO](#) file formats. This Lab provides CSV flight data in an S3 bucket. [The raw data](#) is collected by the United States Department of Transportation's Bureau of Transportation Statistics. Data over a 12 month period starting Dec. 1, 2016, and ending Nov. 30, 2017, is used. The raw data has been cleansed so that it is readily consumable by Amazon SageMaker. The data cleansing process involved:

- Merging monthly comma-separated value (CSV) files into a single CSV file
- Encoding non-numeric fields as numeric fields (SageMaker CSV files must only contain numeric values)
- Removing flight records with no arrival time. Flights may not arrive due to cancelation or diversion. The regression model can only be trained with records that have a flight arrival delay value.
- Randomly shuffling the records
- Sampling the records to reduce the volume of the data to reduce processing times in the Lab. The original data contained over 5,000,000 records while the sampled data contains just over 370,000 records.
- Separating the data into training and test data sets

In this Lab Step, you will observe the attributes in the raw data to understand what features are included. Then you will briefly review the encoded CSV data that SageMaker can understand.

Instructions

1. Read through the following descriptions of attribute groups in the raw flight data to understand what each included attribute represents:

- There are a group of attributes that represent data information, namely year, quarter, month, day of the month, and day of the week (1 represents Sunday, 2 Monday, ..., 7 Saturday):

YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK
2017	4	10	15	7
2017	4	11	23	4
2017	2	6	23	5
2017	1	3	21	2

These attributes allow the model to capture trends related to seasonality and weekdays vs. weekends, for example.

- There are a group of attributes for flight information that includes the airline (**UNIQUE_CARRIER**), specific airplane identification (**TAIL_NUM**), flight number (**FL_NUM**), origin (**ORIGIN**), and destination (**DEST**):

UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN	DEST
VX	N844VA	1174	SFO	EWR
B6	N529JB	470	FLL	BOS
UA	N19130	1815	IAH	SFO
AA	N183AA	628	CLT	PHX

These attributes can be used by the model to capture trends related to specific airlines and airports.

- The next group relates to departure times: scheduled departure time in 24-hour clock format (**CRS_DEP_TIME**), actual departure time (**DEP_TIME**), departure delay with negative values representing early departures (**DEP_DELAY**), departure delay with zero

representing early departures (**DEP_DELAY_NEW**), a binary value representing if a flight was delayed more than 15 minutes on departure (**DEP_DEL15**), and departure delay groups representing how significant the delay was (**DEP_DELAY_GROUP**):

CRS_DEP_TIME	DEP_TIME	DEP_DELAY	DEP_DELAY_NEW	DEP_DEL15	DEP_DELAY_GROUP
805	833	28	28	1	1
713	706	-7	0	0	-1
1320	1324	4	4	0	0
1300	1257	-3	0	0	-1

Except for the scheduled delay which is known well in advance, the other attributes are only known after the plane departs. These attributes aren't known if you were forecasting arrival delays before boarding a flight. You will compare the model performance when you include and exclude these features later in the Lab.

- The last group includes arrival time and distance attributes: scheduled arrival time (**CRS_ARR_TIME**), arrival delay (**ARR_DELAY**), scheduled elapsed time (**CRS_ELAPSED_TIME**), distance in miles (**DISTANCE**), and a distance group that bins flights of similar distances (**DISTANCE_GROUP**):

CRS_ARR_TIME	ARR_DELAY	CRS_ELAPSED_TIME	DISTANCE	DISTANCE_GROUP
1630	2	325	2565	11
1017	-20	184	1237	5
1526	-3	246	1635	7
1435	-23	275	1773	8

These attributes can indicate arrival delay trends related to the time of day of arrivals and lengths of flights.

2. Consider the following list that defines the schema used for the data in this Lab:

- YEAR**: numeric
- QUARTER**: categorical
- MONTH**: categorical
- DAY_OF_MONTH**: categorical
- DAY_OF_WEEK**: categorical
- UNIQUE_CARRIER**: categorical
- TAIL_NUM**: categorical

- **FL_NUM**: categorical
- **ORIGIN**: categorical
- **DEST**: categorical
- **CRS_DEP_TIME**: numeric
- **DEP_TIME**: numeric
- **DEP_DELAY**: numeric
- **DEP_DELAY_NEW**: numeric
- **DEP_DEL15**: numeric (binary)
- **DEP_DELAY_GROUP**: numeric
- **CRS_ARR_TIME**: numeric
- **ARR_DELAY**: numeric (target)
- **CRS_ELAPSED_TIME**: numeric
- **DISTANCE**: numeric
- **DISTANCE_GROUP**: categorical

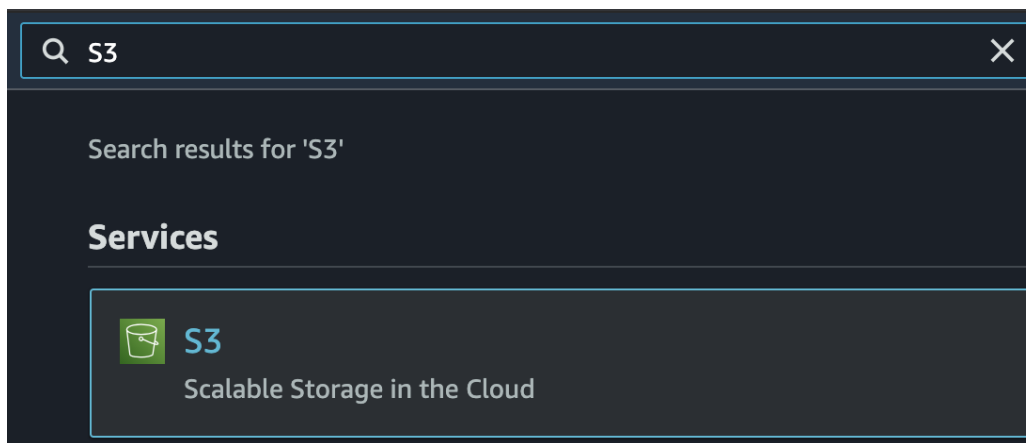
The schema dictates what raw data fields need to be transformed given that SageMaker CSV files only support numeric data. **ARR_DELAY** is the target that you want to predict and must be in the first column of the SageMaker CSV file.

The schema declares several numeric and categorical attributes. There are no text attributes in the data. Deciding if an attribute is numeric or categorical is not obvious in some cases. **ORIGIN** and **DEST** are categorical because there are a limited number of airport values and it doesn't make sense to do mathematical operations on them. For example, it doesn't make sense to add two origins. **DISTANCE** is numeric because it is a numeric quantity and it makes sense to add two distances.

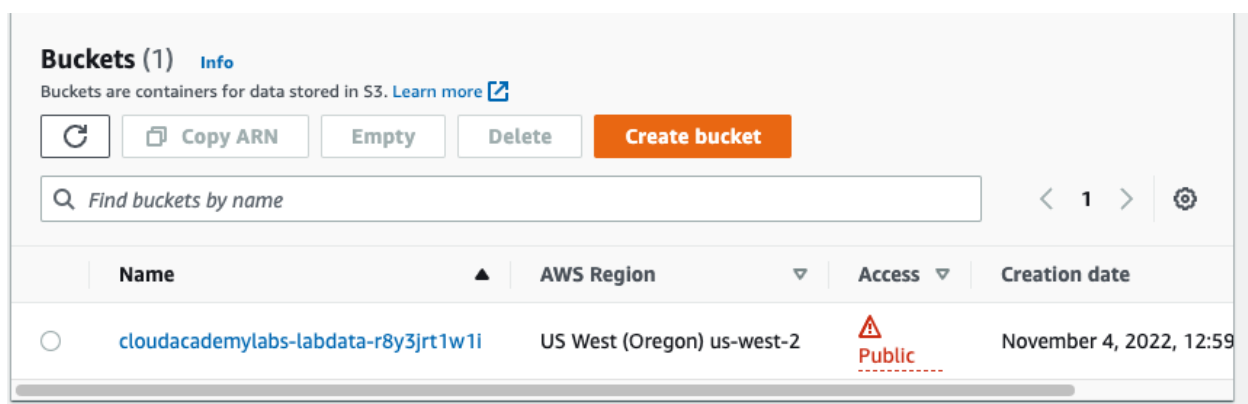
Attributes like **MONTH**, **DAY_OF_WEEK**, and **CRS_ARR_TIME** have an order (ordinal) and it makes some sense to add their values. This supports a numeric attributeType. However, the order is actually cyclical and there are only a finite number of values. For example, **MONTH** can only have twelve values and the month after month 12 is 1, not 13. This has similar properties to a categorical attribute. **MONTH**, **DAY_OF_WEEK**, and **CRS_ARR_TIME** are actually examples of cyclic ordinal attribute types. Amazon Machine Learning doesn't directly support cyclic ordinal attribute types. They can be better represented by transforming their values using a pair of numeric attributes, but to keep the

values easy to interpret they haven't been transformed. The schema has assigned numeric attribute types for times, such as **CRS_ARR_TIME** since there are many possible values. Categorical attribute types have been used for attributes with fewer possible values, such as **QUARTER** and **MONTH**.



3. In the AWS Management Console search bar, enter S3, and click the **S3** result under **Services**:



4. Click on the bucket with a name that begins with **cloudatacademy-labdata**:



5. Observe the two files included in the bucket:

<input type="checkbox"/> Name ▼	Last modified ▼	Size ▼
<input type="checkbox"/>  Flights_test.csv	Apr 23, 2019 10:48:51 AM GMT-0600	155.1 MB
<input type="checkbox"/>  Flights_training.csv	Apr 23, 2019 10:48:54 AM GMT-0600	362.0 MB

The data file that will be used for training a machine learning model is **Flights_training.csv** and the **Flights_test.csv** file is used as an independent test set of data for evaluating the trained model performance. The data files are both over 100MB. To save you from downloading them, the following instruction highlights the differences in the data compared to the raw data.

6. Observe the following encoding of the QUARTER attribute:

QUARTER_1	QUARTER_2	QUARTER_3	QUARTER_4
0	0	1	0
0	0	0	1
0	1	0	0

The image above uses one-hot encoding to encode the categorical QUARTER attribute into four new attributes, one for each possible category. One-hot encoding is used for all of the categorical features in the transformed data with a few exceptions:

- The YEAR attribute is removed since there aren't enough years of data to draw meaningful trends from the year
- The CARRIER, TAIL_NUM, and FL_NUM attributes are dropped to make the file sizes more manageable. These attributes have the most categories which cause thousands of new attributes to be added when using one-hot encoding. There are [other ways to encode](#) these attributes but they are outside of the scope of this Lab.

Note: The header row in the CSV file is removed in the files in S3 because SageMaker requires only numeric values in the file.

Summary

In this Lab Step, you learned about the data that you will use to perform regression and its schema. You also learned some guidelines for determining when to use different attribute types.

Creating a SageMaker Training Job

Introduction

Amazon SageMaker is a fully managed service that covers the entire machine learning workflow to choose an algorithm, train the model, tune and optimize it for deployment, make predictions, and take action. Your models get to production faster with much less effort and lower cost.

Because the data has been prepared for you, the first step is to create a training job to train a model. Training jobs make use of algorithms.

SageMaker has a variety of built-in algorithms that you can use. When those algorithms don't suit your application, you can create your own or find additional algorithms in the AWS Marketplace. For this Lab, the built-in linear learner model is all that you need. It allows you to train a variety of models, including linear regression, which is what you need to forecast the amount of delay flights will have. That is because the delay is a continuous value compared to a value that is in a finite set of possible values. You will create a training job in this Lab Step.

Instructions

1. [Navigate to the Amazon SageMaker Console:](#)

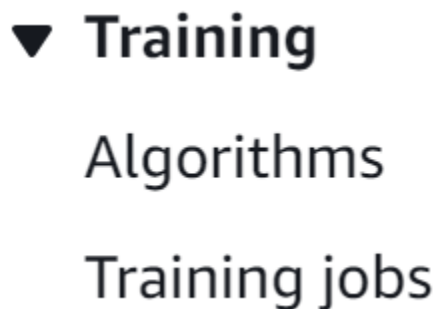
MACHINE LEARNING

Amazon SageMaker

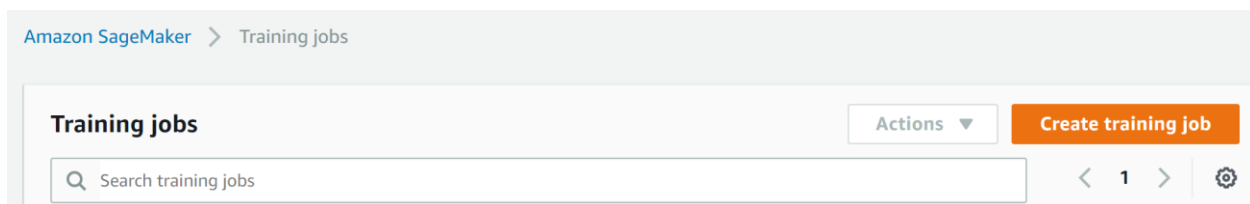
Build, train, and deploy machine learning models at scale

The quickest and easiest way to get ML models from idea to production.

2. Click **Training** > **Training jobs** in the left sidebar:



3. Click **Create training job**:



4. In the **Job settings** section of the **Create a training job** form, set the following values leaving the defaults for the rest:

- **Job name:** *flight-delay-training-#####* where you replace ##### with random digits (Training jobs must have a unique name within an AWS account and region)
- **IAM role: Enter a custom IAM role ARN**
 - **Custom IAM role ARN:** `arn:aws:iam::702751633937:role/sagemaker-role` (The Cloud Academy Lab environment created the role with enough permissions to complete the Lab)
- **Algorithm source:** **Amazon SageMaker built-in algorithm** (You can leverage a variety of existing training algorithms with this setting)
- **Choose an algorithm:** **Tabular-Linear Learner**
- **Input mode:** **File** (The other mode is pipe which can be used for streaming data from S3 for faster training times, but File data is all you need for this Lab)

Job settings

Job name

flight-delay-training-55696

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

IAM role



Amazon SageMaker requires permissions to call other services on your behalf. Choose a role or let us create a role that has the [AmazonSageMakerFullAccess](#) IAM policy attached.

sagemaker-role

Algorithm options

Use an Amazon SageMaker built-in algorithm, your own algorithm, or a third-party algorithm from AWS Marketplace.

▼ Algorithm source

- ☒ Amazon SageMaker built-in algorithm [Learn more](#) 
- ☐ Your own algorithm resource
- ☐ Your own algorithm container in ECR [Learn more](#) 
- ☐ An algorithm subscription from AWS Marketplace

▼ Choose an algorithm

Tabular - Linear Learner

Container

The registry path where the training image is stored in Amazon ECR. [Learn more](#)

174872318107.dkr.ecr.us-west-2.amazonaws.com/linear-learner:1

Input mode

You can provide your training data as a file or pipe.


File

The **Metrics** that are reported to CloudWatch during training are also displayed.

5. In the **Hyperparameters** section, set the following values leaving the defaults for the rest:

- **feature_dim:** 716
- **predictor_type:** regressor (other predictor types are used for creating classification models)

Hyperparameters

You can use hyperparameters to finely control training. We've set default hyperparameters for the algorithm you've chosen. [Learn more](#) 

Key	Value
feature_dim	716
mini_batch_size	1000
epochs	15
predictor_type	regressor ▼

There are many hyperparameters that control the algorithm. When dealing with big data or repetitive training it may be worth spending some time optimizing the algorithm hyperparameters. SageMaker has [hyperparameter tuning jobs](#) to help simplify the process, but you will not need that for this Lab.

6. [Navigate to the S3 Console](#) and copy the name of the S3 bucket that the Cloud Academy Lab environment has created for you:

S3 buckets

[Discover the console](#)

Search for buckets

All access types

Create bucket

Edit public access settings

Empty

Delete

1 Buckets

1 Regions

Bucket name

Access

Region

Date created

cloudacademylabs-labdata-o3t1sn3rc1w

Public

US West
(Oregon)

Apr 18, 2019
11:28:23 PM
GMT-0600

The bucket name will begin with **cloudacademylabs-labdata-** but the last segment of the name varies each time the Lab is started. You need to use the bucket name to tell SageMaker where it can find input data in the next instruction.

7. In the **Input data configuration** section, set the following values leaving the defaults for the rest:

- **Input mode:** File
- **Content type:** `text/csv;label_size=1` (The data has a single target which is the delay so label_size is set to 1)
- **S3 location:** `s3://<bucket_name>/Flights_training.csv` where you replace `<bucket_name>` with the name of the S3 bucket you copied

▼ train

Remove

Channel name

train

Input mode - *optional*

File

Content type - *optional*

text/csv;label_size=1

Choose one of the formats below

- application/x-recordio-protobuf
- text/csv
- text/csv;label_size={Number of label columns}

Compression type

None

Record wrapper

None

Data source

☒ S3

☐ File system

S3 data type

S3Prefix

S3 data distribution type

FullyReplicated

S3 location

s3://cloudacademylabs-labdata-r8y3jrt1w1i/Flights_training.csv

8. Click **Add channel** to create another input source that will be used for testing the trained model.

9. In the new channel section, set the following values leaving the defaults for the rest:

- **Channel name:** *test* (Sagemaker built-in algorithms expect specific channel names so be sure to use the exact name)
- **Input mode:** **File**
- **Content type:** *text/csv;label_size=1*
- **S3 location:** *s3://<bucket_name>/Flights_test.csv* where you replace *<bucket_name>* with the name of the S3 bucket you copied

The test data reserves 30% of the original data for independently testing the model. You can also specify a validation channel, but because SageMaker will automatically use 30% of the training data for validation when using the linear learner model, it is not necessary.

10. In the **Output data configuration** section, set the following values leaving the defaults for the rest:

- **S3 output path:** `s3://<bucket_name>/` where you replace `<bucket_name>` with the name of the S3 bucket you copied

Output data configuration

S3 output path

`s3://cloudacademylabs-labdata-o3tln3rclw/`

The output will be placed in a folder with the job name under the **output path**.

11. Click **Create training job** to create the job and have SageMaker automatically start training a model:

Training jobs				Actions ▼	Create training job
<input type="text" value="Search training jobs"/>				< 1 >	⚙
Name ▼	Creation time ▼	Duration	Status ▼		
flight-delay-training-12345	Apr 18, 2019 19:11 UTC	-	InProgress		

The model takes around twelve minutes to train. You will observe the progress in the remaining instructions of this Lab Step.

12. Click the **Name** of your training job to display its details.

13. In the **Job settings** section, you can click on **View history** under **Status** to see the stages the training job progresses through:

flight-delay-training-23524

Clone

Job settings

Job name

flight-delay-training-23524

ARN

arn:aws:sagemaker:us-west-2:123456789012:training-job/flight-delay-training-23524

Status

InProgress

- Training

[View history](#)

14. Observe the displayed **Status** and **Description** values before closing the dialog box.

It takes a few minutes to prepare the instances for training in the **Starting Status**. The majority of the time is spent in **Training** when the training algorithm is training the model to minimize the difference between the predicted and actual delay. The training job is complete when the **Status** is **Completed**.

15. Scroll down to the **Monitor** section and click **View logs** to open the CloudWatch Log Group for the training job:

Monitor

Access logs for debugging and progress reporting. View metrics to set alarms, send notifications, or take actions. [Learn more](#)

[View algorithm metrics](#)

[View instance metrics](#)

[View logs](#)

16. Click the name of the Log Group (**flight-delay-training...**) to display the logs for training:

Filter events		Clear	1m	30m	1h	12h	Custom		
▶	Timestamp	Message							
		There are older events to load. Load more .							
▶	2022-11-04T13:50:07.884-05:00	#metrics {"StartTime": 1667587807.1345673, "EndTime": 1667587807.1345828, "D...							
▶	2022-11-04T13:50:07.884-05:00	#metrics {"StartTime": 1667587807.1346319, "EndTime": 1667587807.1346471, "D...							
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] #quality_metric: host=algo-1, epo...							
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] #early_stopping_criteria_metric: ...							
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] Epoch 10: Loss improved. Updating...							
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] Saving model for epoch: 10							
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] Saved checkpoint to "/tmp/tmpvlu_...							
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] #progress_metric: host=algo-1, co...							
▶	2022-11-04T13:50:07.885-05:00	#metrics {"StartTime": 1667587771.1115475, "EndTime": 1667587807.14519, "Dim...							
▶	2022-11-04T13:50:07.885-05:00	[11/04/2022 18:50:07 INFO 139867255830336] #throughput_metric: host=algo-1, ...							
▶	2022-11-04T13:50:43.895-05:00	#metrics {"StartTime": 1667587843.3425455, "EndTime": 1667587843.342612, "Di...							
▶	2022-11-04T13:50:43.895-05:00	#metrics {"StartTime": 1667587843.3427026, "EndTime": 1667587843.3427174, "D...							
▶	2022-11-04T13:50:43.895-05:00	#metrics {"StartTime": 1667587843.3427684, "EndTime": 1667587843.3427854, "D...							
▶	2022-11-04T13:50:43.895-05:00	#metrics {"StartTime": 1667587843.3428423, "EndTime": 1667587843.34286, "Dim...							

A variety of logs are recorded. The **#metrics** logs allow you to monitor various metrics as the training progresses. For example, in the image above the bottom **#metrics** logs report the **train_mse_objective** metrics, which you can use to monitor the algorithms progress in minimizing the mean squared error

(MSE) loss function. Gradually over time you should see the **max** values decreases as the training improves the model. You will also see model update message as follows:

▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] Epoch 10: Loss improved. Updating...
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] Saving model for epoch: 10
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] Saved checkpoint to "/tmp/tmpvlu_...
▶	2022-11-04T13:50:07.884-05:00	[11/04/2022 18:50:07 INFO 139867255830336] #progress_metric : host=algo-1, co...

When an **Epoch** improves upon the previous best model, a **Loss improved. Updating best model** message will be displayed.

17. Return to the SageMaker Console browser tab and observe the charts further down the **Monitor** section that graphically track the algorithm and instance metrics:



A new data point is added to the charts every five minutes. You can use the data to determine if the instance type used for training is under- or oversized as well as benchmarking different algorithm parameters by comparing throughputs. The ml.m4.xlarge training instance has four vCPUs and 16GiB of memory. The **MemoryUtilization** is below 1% and the **CPUUtilization** is well below the maximum of 400%, so the training process is not limited by memory

or CPU. The throughput (**train:throughput**) is the bottleneck in this case. Throughput measures how many training records the algorithm processes per second. You can improve throughput by experimenting with different algorithm hyperparameters (such as `mini_batch_size`) and using the RecordIO format rather than a CSV file. Lastly, As you saw in the logs, the loss chart (**train:objective_loss**) reduces over time signaling that the training is making progress.

18. Wait until the training completes and the **S3 model artifact** link appears in the **Output** section:

Output


S3 model artifact

<s3://cloudacademylabs-labdata-o3tln3rc1w/flight-delay-training-23524/output/model.tar.gz> 

You use trained model artifacts to create SageMaker models. You will do this in the following Lab Step.

19. Click the **S3 model artifact** link to open the location in S3:

☐ Name ▼

☐  model.tar.gz

Having a browser tab open to the model artifact will help with creating a model in the next Lab Step.

20. Return to the CloudWatch Logs and scroll to the bottom:

```
▼ 2022-11-04T13:53:14.947-05:00 #metrics {"StartTime": 1667587407.2478688, "EndTime": 1667587994.4919555, ...  
#metrics  
{  
  "StartTime": 1667587407.2478688,  
  "EndTime": 1667587994.4919555,  
  "Dimensions": {  
    "Algorithm": "Linear Learner",  
    "Host": "algo-1",  
    "Operation": "training"  
  },  
  "Metrics": {  
    "initialize.time": {  
      "sum": 1838.8800621032715,  
      "count": 1,  
      "min": 1838.8800621032715,  
      "max": 1838.8800621032715  
    }  
  },  
  ...  
}
```

Copy

There is a lot of useful information here including what hyperparameters yielded the best model, the **test_score mse_objective** which measures the model performance on the test data that is not used during training, and time **metrics** for different stages of the training process. The **test mse_objective** is approximately **163**. Since MSE is a squared error, you can interpret the square root of the MSE as a kind of average error. The square root of the MSE is approximately 12.7 minutes in this case.

Summary

In this Lab Step, you trained a model using a SageMaker training job that uses the linear learner built-in algorithm. You used a test data channel to evaluate the model independent of the training data used to train the model. You used CSV files for the training job. CSV files are very common but have some disadvantages. Most models will train faster when using [RecordIO format](#). RecordIO format also supports Pipe mode, as opposed to file mode, which allows for streaming data from S3 for faster training (compared to downloading the entire file locally). However, it is not as easy to find RecordIO data and it requires some programming expertise to convert a CSV to RecordIO format.

Creating a SageMaker Model

Introduction

The SageMaker training job produced model artifacts in an Amazon S3 bucket. Those artifacts make it easy to share training results, but they are not readily usable for inference. You must first create a SageMaker model from the artifacts. You will do just that in this Lab Step.

Instructions

1. Click **Inference** > **Models** in the left sidebar menu:

▼ Inference

Compilation jobs

Model packages

Models

2. Click **Create model**:



3. In the **Model settings** section of the **Create model** form, set the following values leaving the defaults for the rest:

- **Model name:** *flight-delays*
- **IAM role:** Enter a custom IAM role ARN
 - **Custom IAM role ARN:** `arn:aws:iam::702751633937:role/sagemaker-role`

Model settings

Model name

flight-delays

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

IAM role


Amazon SageMaker requires permissions to call other services on your behalf. Choose a role or let us create a role that has the [AmazonSageMakerFullAccess](#) IAM policy attached.

sagemaker-role

4. In the **Container definition 1** section, set the following values leaving the defaults for the rest:

- **Provide model artifacts and inference image**
 - **Location of inference code image:** `174872318107.dkr.ecr.us-west-2.amazonaws.com/linear-learner:1` (This is the same image as the image used for training)
 - **Location of model artifacts:** Copy the location of the model artifact you opened in another browser tab by selecting it and copying the **Object URL** link and pasting it in this field:

☒ Name ▼

☒  model.tar.gz

Overview

Key	model.tar.gz
Size	3.6 KB
Expiration date	N/A
Expiration rule	N/A
ETag	70af1dd08da0bf3f32d81b473cd9b23a
Last modified	Apr 19, 2019 12:53:42 PM GMT-0600
Object URL	https://s3-us-west-2.amazonaws.com/174872318107/output/model.tar.gz

Properties

Storage class	Standard
Encryption	AWS-KMS

Open link in new tab

Open link in new window

Open link in incognito

Save link as...

Copy link address

Container definition 1

▼ Container input options

☒ **Provide model artifacts and inference image location**

Use this for models trained using built-in algorithms, BYO algorithms, or models trained outside Amazon SageMaker.

☐ **Use a model package resource**

Use this for model packages that contain inference images and artifacts from AWS Marketplace subscribed algorithms.

☐ **Use a model package subscription from AWS Marketplace**

Use this for model packages published by vendors from AWS Marketplace.

▼ Provide model artifacts and inference image options

☒ **Use a single model**

Use this to host a single model in this container.

☐ **Use multiple models**

Use this to host multiple models in this container.

Location of inference code image

Type the registry path where the inference code image is stored in Amazon ECR.

174872318107.dkr.ecr.us-west-2.amazonaws.com/linear-learner:1

Location of model artifacts - *optional*

Type the URL where model artifacts are stored in S3.

s3://cloudacademylabs-labdata-r8y3jrt1w1i/flight-delay-training-33445/output/r

The path must point to a single gzip compressed tar archive (.tar.gz suffix).

Container host name - *optional*

Type the DNS host name for the container.

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

This section makes it clear that SageMaker uses Docker containers under the hood. If you were to create your own algorithms or inference code, you would need to package them as Docker containers for SageMaker to use them.

5. Click **Create model**.

The model is shown in a table:

Models			Create endpoint	Create endpoint configuration	Actions ▼	Create model
<input type="text" value="Search models"/>			< 1 >			⚙️
	Name ▼	ARN	Creation time ▼			
<input type="radio"/>	flight-delays	arn:aws:sagemaker:us-west-2:123456789012:model/flight-delays	Apr 19, 2019 19:07 UTC			

Summary

In this Lab Step, you created a SageMaker model using the built-in linear learner image and the model artifacts created by the training job. You still cannot directly use a SageMaker model for inference. You must create endpoints or transform jobs that combine the model with compute infrastructure to serve model requests. You will create an endpoint for the model in the next Lab Step.

Creating a SageMaker Endpoint

Introduction

There are two methods for obtaining inferences from SageMaker models. SageMaker endpoints are for generating real-time inferences with models. When you need to get inferences for a large set of data, you can instead use batch transform jobs.

You will create a SageMaker endpoint using your flight delay model in this Lab Step. The endpoint encapsulates the compute resources needed to host your model in an endpoint configuration. The resources are fully managed by AWS, so you don't need to worry about managing the servers.

Instructions

1. Click **Inference > Endpoints** in the sidebar menu:

▼ **Inference**

Compilation jobs

Model packages

Models

Endpoint configurations

Endpoints

2. Click **Create Endpoint**.

3. In the **Endpoint** section of the **Create and configure endpoint** form, set the following value:

- **Endpoint name:** *flight-delays*

4. In the **Attach endpoint configuration** section, select **Create a new endpoint configuration**:

Attach endpoint configuration

☐ **Use an existing endpoint configuration**
Use an existing endpoint configuration or clone an endpoint configuration.

☒ **Create a new endpoint configuration**
Add models and configure the instance and initial weight for each model.

5. In the **New endpoint configuration** section, set the following values leaving the defaults for the rest:

- **Endpoint configuration name:** *flight-delays*
- **Type of endpoint:** *Serverless*
- **Production variants:** Click **Add model**, then **Save** to add and select the **flight-delays** model (endpoints can serve multiple models and balance traffic between them which can be useful when deploying new versions)

New endpoint configuration

Endpoint configuration

Endpoint configuration name

flight-delays

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Type of endpoint

☐ Provisioned

☒ Serverless

Encryption key - *optional*

Encrypt your data. Choose an existing KMS key or enter a key's ARN.

No Custom Encryption ▼



Production variants

Model name	Training job	Variant name	Memory Size	Max Concurrency	Actions
flight-delays	test1	variant-name-1	1 GB	20	Edit Remove

Create endpoint configuration

6. Click **Create endpoint configuration** followed by **Create Endpoint**.

The endpoint is displayed in a table:

	flight-delays	arn:aws:sagemaker:us-west-2:440464731830:endpoint/flight-delays	Nov 04, 2022 19:06 UTC	 Creating
---	----------------------	---	---------------------------	--

It takes a few minutes to provision the compute resources and load the model.

7. Click **flight-delays** to view the endpoint details.

Here you can find the **URL** to send inference requests to. There is also a **Monitor** section similar to training jobs where you can view metrics and logs for the endpoint (you may need to refresh the page after a few minutes to view the charts). After a few minutes the **Status** reaches **InService** and the endpoint is ready to use:

Status

 **InService**

Summary

In this Lab Step, you created a SageMaker endpoint and an endpoint configuration to describe the resources required for hosting your model.

Using the SageMaker Model for Inferences

Introduction

You can generate inferences by [invoking a SageMaker endpoint](#) in a variety of ways:

- REST API,
- AWS software development kits (SDKs),
- [the SageMaker-specific SDK](#), and

- AWS command line interface (CLI).

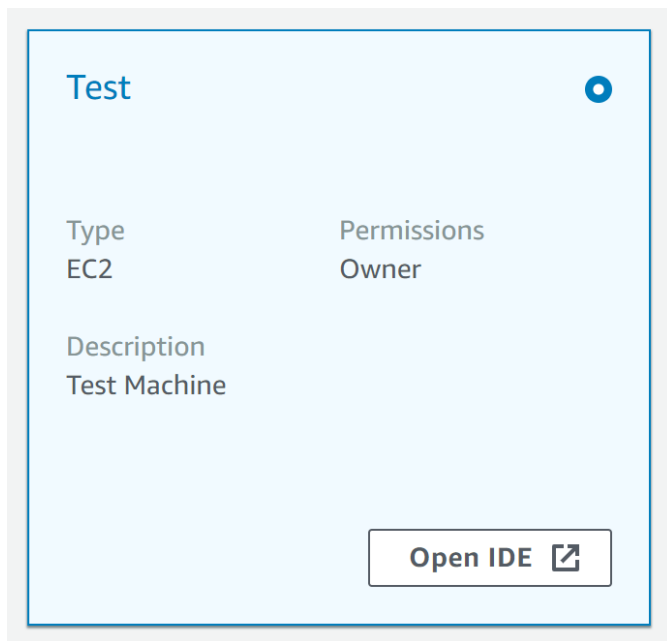
You will use the AWS CLI to invoke the endpoint in this Lab. You will create a shell script that will tell you how much delay to expect for a given flight. For the sake of this Lab Step, assume that you are picking up a friend at Indianapolis International Airport and they are flying from San Francisco International Airport. You would like to estimate the delay so you can decide when to leave for the airport.

You will use an AWS Cloud9 environment for editing the script and running commands from within your web browser. SageMaker endpoints are not publicly accessible and can only be accessed from within AWS.

The `sagemaker:InvokeEndpoint` IAM permission is required to use the endpoint. The Cloud9 instance has the required permission to use the endpoint. It is worth mentioning that although a SageMaker endpoint cannot be directly accessed via the internet you can configure an API Gateway and Lambda function to invoke the SageMaker endpoint allowing for access over the internet.

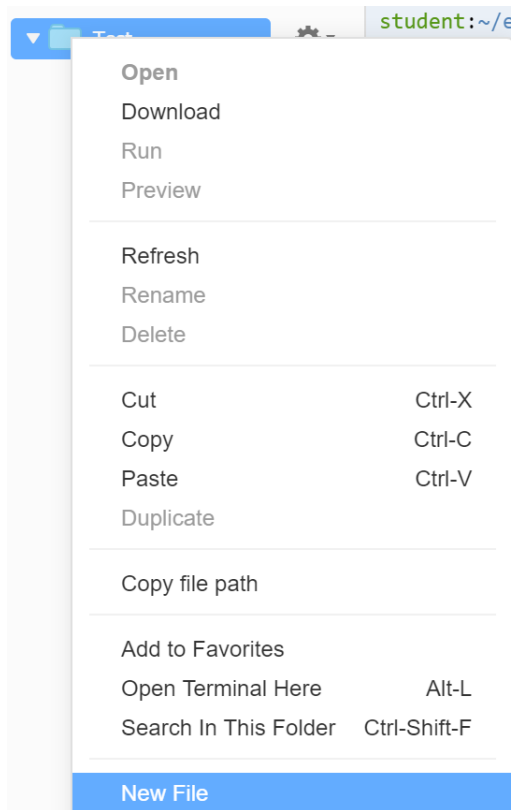
Instructions

1. [Navigate to the AWS Cloud9 environments view.](#)
2. Click **Open IDE** in the **Test** environment card:

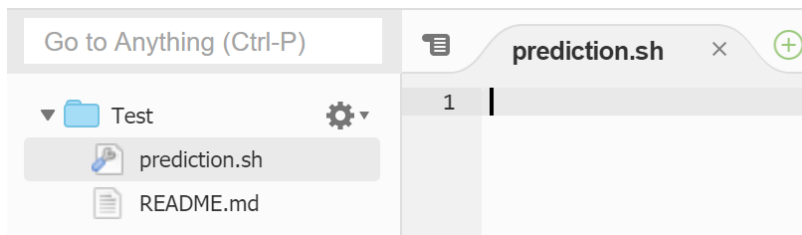


Wait until any progress messages disappear and the welcome tab is visible.

3. Right-click the **Test** folder in the left **Environment** tab and click **New File**:



4. Enter *prediction.sh* as the file name and double-click it to open an editor tab for it:



You will add some commands to the prediciton.sh script to output a predicted flight delay for a given input feature vector.

5. Paste the following into the prediction.sh file and save the file (**File > Save**):

[Copy code](#)

[illegible]

```
# Extract delay prediction from JSON output
predicted_delay=$(grep -Eo "[-0-9.]+" delay_prediction.json)

echo "Flight forecasted to be $predicted_delay minutes delayed"
```

Read through the comments in the code to understand how it works.

The `flight_vector` is unwieldy with so many features, however, the first 9 features are easy to interpret. The vector is best generated by machines and would require some programming experience. That is why the CSV feature vector is provided for you. [SageMaker Models support pipelines](#) of containers to encapsulate any data transformations that need to take place, avoiding the need for any external processing. In this example, a pipeline could be used to perform the one-hot encoding of raw data that is easier for humans to understand in the first stage. Then the second stage would compute the inference using the transformed data. This Lab omits the pipeline for simplicity.

6. In the terminal tab at the bottom of the window, issue the following command to run the script:

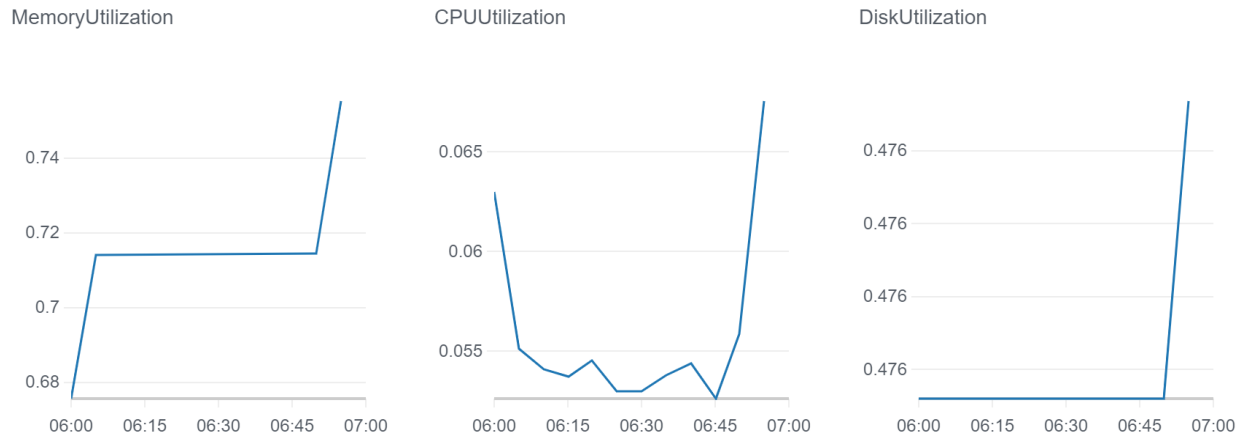
[Copy code](#)

```
bash prediction.sh
```

```
Flight forecasted to be 8.520586967468262 minutes delayed
```

The actual delay for that flight was 14 minutes.

7. Return to the SageMaker endpoint tab and observe the charts in the **Monitor** section:



8. Follow the **View logs** link and click the log group to view the endpoint logs.

The last message corresponds to when the script invoked the endpoint:

```
► 06:59:35 #metrics {"Metrics": {"invocations.count": {"count": 1, "max": 1, "sum": 1.0, "min": 1}}, "EndTime": 1555743574.878947,
```

9. Return to Cloud9 and try editing values of the flight vector to see how the delay prediction changes.

For example, increase the distance by changing `1943` in column 9 to `2943` and see how the predictions change. You can start to reason about how the model works by adjusting the flight vector.

Summary

In this Lab Step, you invoked the SageMaker endpoint from within a script using the AWS CLI and obtained real-time flight delay predictions. You also observed the built-in metrics for SageMaker endpoints. You can use the metrics to understand when you need to adjust your endpoint configuration to increase/decrease the endpoints resources.

Challenge (Optional)

Try creating another (similar) model that is trained using different training job hyperparameters. and compare the test set objective loss (search for **mse_objective** in the CloudWatch Logs) and also create an endpoint to test the model using the `flight_vector` in this Lab Step. For reference, the best model performance with the default hyperparameters was obtained using:

- `learning_rate`: 0.005
- `optimizer`: adam
- `wd`: 0.0001
- `l1`: 0
- `lr_scheduler_minimum_lr`: 0.0001
- `lr_scheduler_factor`: 0.99
- `lr_scheduler_step`: 10

The [descriptions of all the linear learner hyperparameters are here](#).