

Project2: Machine Learning Cloud Service

Project2: Machine Learning Cloud Service

Overview

Benefits of ML in the Cloud

Getting Started

Creating Instance

Open Jupyter

Create Python3 File

Create Tensorflow File

Using kmeans of Sagemaker to build a model

Prequisites and Preprocessing

Import data

Start training

Deploy

Result

End

Overview

Artificial intelligence and machine learning are steadily making their way into enterprise applications in areas such as customer support, fraud detection, and business intelligence. There is every reason to believe that much of it will happen in the cloud.

Benefits of ML in the Cloud

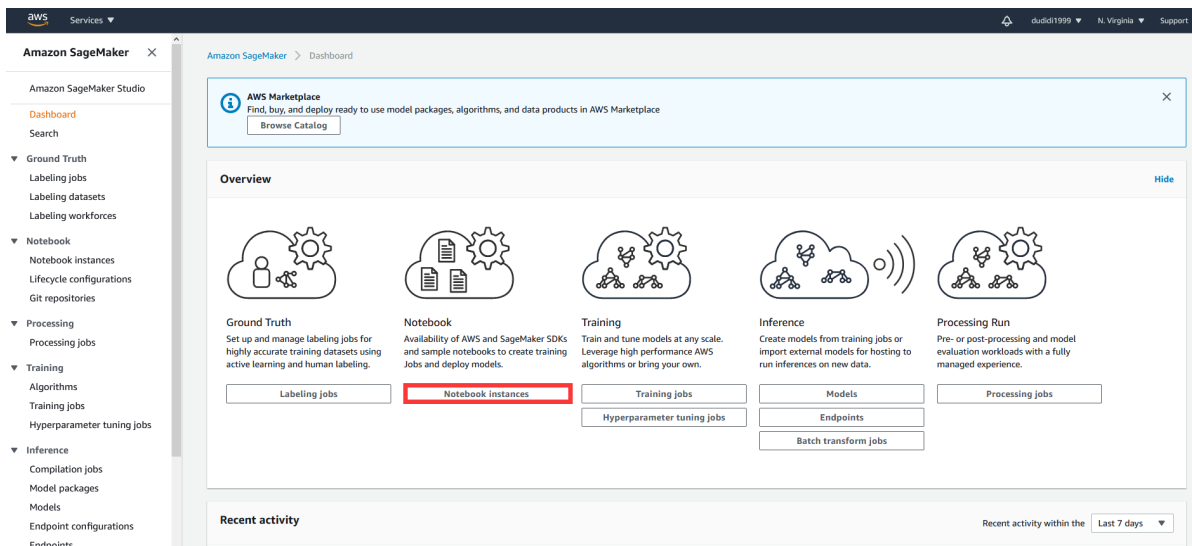
- The cloud's pay-per-use model is good for bursty AI or machine learning workloads.
- The cloud makes it easy for enterprises to experiment with machine learning capabilities and scale up as projects go into production and demand increases.
- The cloud makes intelligent capabilities accessible without requiring advanced skills in artificial intelligence or data science.
- AWS, Microsoft Azure, and Google Cloud Platform offer many machine learning options that don't require deep knowledge of AI, machine learning theory, or a team of data scientists.

Getting Started

Here use AWS as an example. AWS has many free services. Commonly used EC2, S3, RDS, etc. all support a one-year limited free service, and Sagemaker also has a free experience service. Sagemaker is AWS's machine learning training platform, hosting Jupyter Notebook, and many commonly used machine learning algorithms are built-in.

Creating Instance

In dashboard of sagemaker, we can create notebook instances.



Give it a name. Then create a IAMrole, select VPC and subnet. Omit other options.

Notebook instance name

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

Notebook instance type

Elastic Inference [Learn more](#)

► **Additional configuration**

Permissions and encryption

IAM role
Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the [AmazonSageMakerFullAccess](#) IAM policy attached.

Root access - optional

☒ **Enable** - Give users root access to the notebook

☐ **Disable** - Don't give users root access to the notebook
Lifecycle configurations always have root access

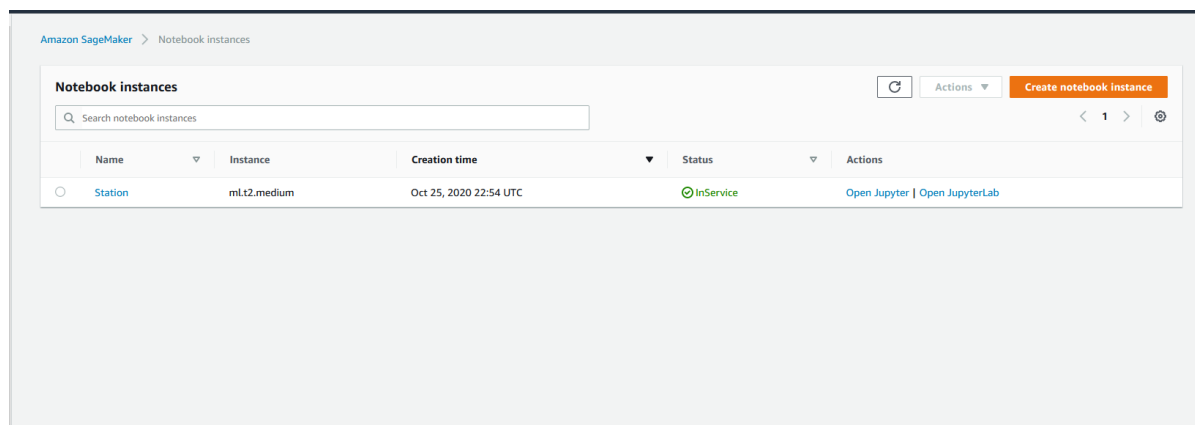
Encryption key - optional
Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.

▼ **Network - optional**

VPC - optional

Subnet
Choose a subnet in an availability zone supported by Amazon SageMaker.

Wait until progress finishes.

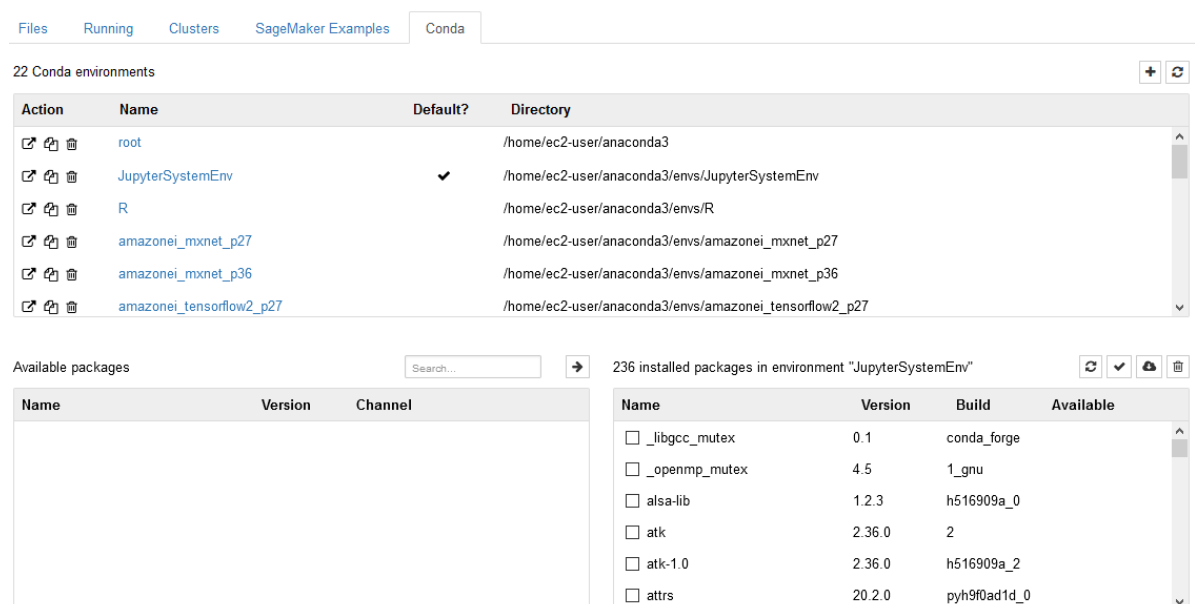


Open Jupyter

Click open Jupyter and it will direct to Jupyter. Here is Jupyter Notebook provided by Sagemaker.



Its appearance looks same as local Jupyter Notebook. But Sagemaker provides more options. Conda tag is anaconda management page and Sagemaker Example provides online demo by AWS.

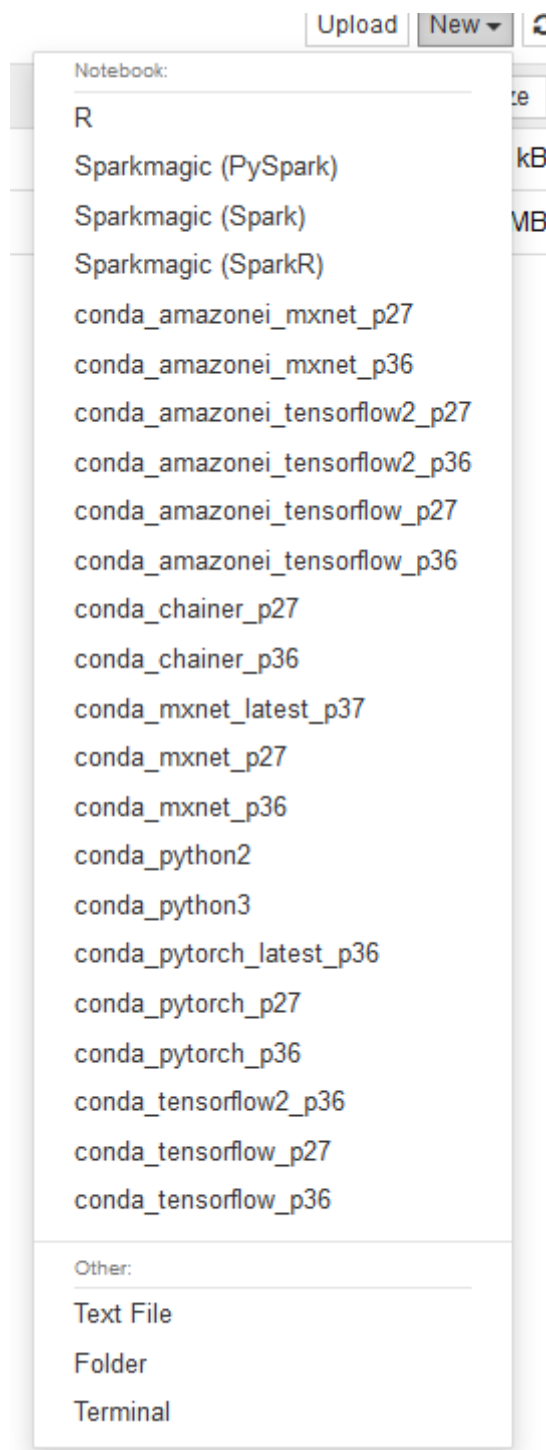


Files	Running	Clusters	SageMaker Examples	Conda
-------	---------	----------	--------------------	-------

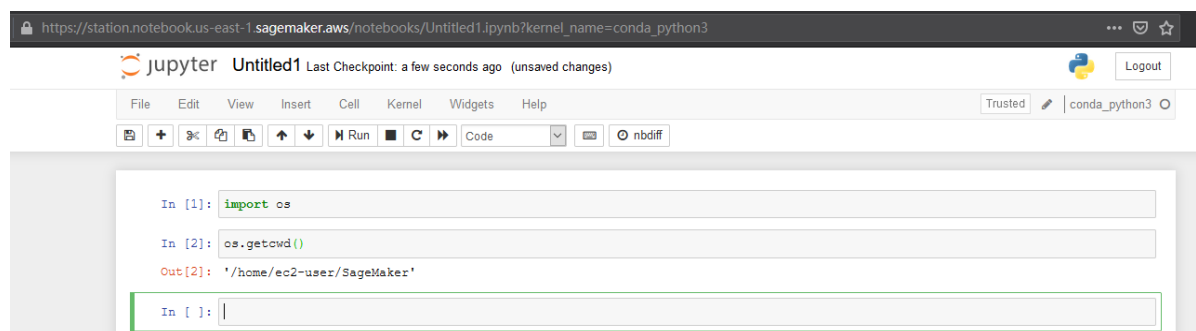
A collection of Amazon SageMaker sample notebooks.

Introduction to Amazon Algorithms		
DeepAR-Electricity.ipynb	Preview	Use
Image-classification-fulltraining-elastic-inference.ipynb	Preview	Use
Image-classification-fulltraining-highlevel.ipynb	Preview	Use
Image-classification-fulltraining.ipynb	Preview	Use
Image-classification-incremental-training-highlevel.ipynb	Preview	Use
Image-classification-lst-format-highlevel.ipynb	Preview	Use
Image-classification-lst-format.ipynb	Preview	Use
Image-classification-multilabel-lst.ipynb	Preview	Use
Image-classification-transfer-learning-highlevel.ipynb	Preview	Use
Image-classification-transfer-learning.ipynb	Preview	Use
LDA-Introduction.ipynb	Preview	Use
Linear_Learner_Regression_csv_format.ipynb	Preview	Use
SageMaker-Seq2Seq-Translation-English-German.ipynb	Preview	Use
blazingtext_hosting_pretrained_fasttext.ipynb	Preview	Use
blazingtext_text_classification_dbpedia.ipynb	Preview	Use
blazingtext_word2vec_subwords_text8.ipynb	Preview	Use
blazingtext_word2vec_text8.ipynb	Preview	Use
deepar_synthetic.ipynb	Preview	Use
factorization_machines_mnist.ipynb	Preview	Use
ipinsights-tutorial.ipynb	Preview	Use

The new button has more file types, which can create Spark, tensorflow and pytorch project files. The concept in Jupyter is called kernel. Adding a Kernel can add a new file type that can be created. It is quite powerful and does not require users. The environment is still very powerful if it depends on its own configuration.

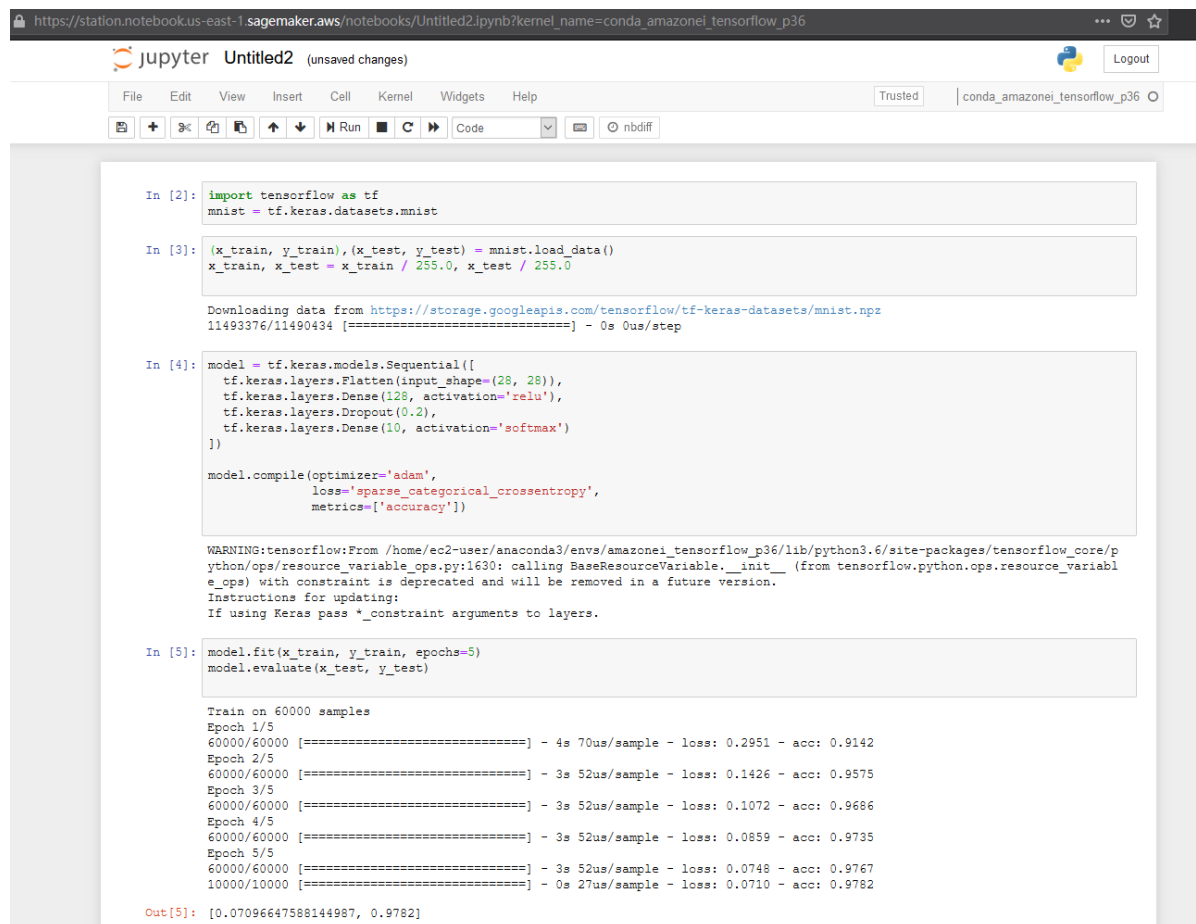


Create Python3 File



Create Tensorflow File

Put the case code on Tensorflow official website to do a simple model training.



```
In [2]: import tensorflow as tf
mnist = tf.keras.datasets.mnist

In [3]: (x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [=====] - 0s 0us/step

In [4]: model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

In [5]: model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

Train on 60000 samples
Epoch 1/5
60000/60000 [=====] - 4s 70us/sample - loss: 0.2951 - acc: 0.9142
Epoch 2/5
60000/60000 [=====] - 3s 52us/sample - loss: 0.1426 - acc: 0.9575
Epoch 3/5
60000/60000 [=====] - 3s 52us/sample - loss: 0.1072 - acc: 0.9686
Epoch 4/5
60000/60000 [=====] - 3s 52us/sample - loss: 0.0859 - acc: 0.9735
Epoch 5/5
60000/60000 [=====] - 3s 52us/sample - loss: 0.0748 - acc: 0.9767
10000/10000 [=====] - 0s 27us/sample - loss: 0.0710 - acc: 0.9782

Out[5]: [0.07096647588144987, 0.9782]
```

Epoch5, the final training accuracy is 97.8%, not bad.

If you simply use Tensorflow, you don't see any advantages, because these operations can also be performed locally, but the speed of downloading the data set is a bit slow.

AWS has many great products, but Sagemaker itself is not very innovative. Many functions are the original IpythonNotebook functions, and they need to be paid. For general users, normal use is still a bit expensive. If there are a lot of training requirements, it is of course very useful. The difference between local computing power and cloud t2, medium is not a little bit different.

Similarly, Google has a tool called Colaboratory, which is actually almost the same as Jupyter Notebook, and the latter is free.

Let's try again using SageMaker's built-in algorithm, and use kmeans to make a simple classification. The following is a brief look at how to use SageMaker for training, this part will see the advantages of cloud machine learning.

Using kmeans of SageMaker to build a model

Prerequisites and Preprocessing

Permissions and environment variables

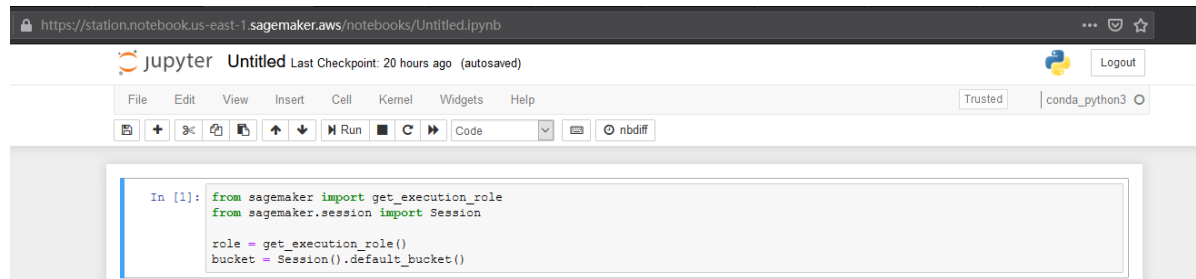
Here we set up the linkage and authentication to AWS services. There are two parts to this:

1. The roles used to give learning and hosting access to your data. Here we extract the role you created earlier for accessing your notebook. See the documentation if you want to specify a different role.

2. The S3 bucket name that you want to use for training and model data. Here we use a default in the form of `sagemaker-{region}-{AWS account Id}`, but you may specify a different one if you wish.

Import data

Obtain execution permissions, and create an s3bucket to store these training data. (AWS provides free 5G s3 space).

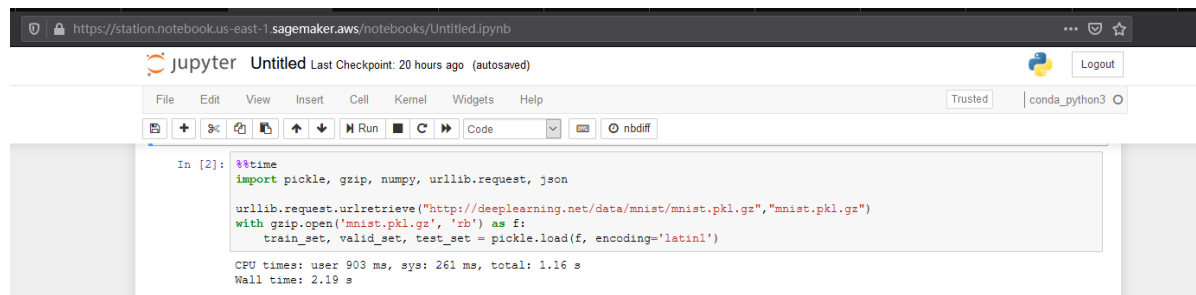


```
In [1]: from sagemaker import get_execution_role
        from sagemaker.session import Session

        role = get_execution_role()
        bucket = Session().default_bucket()
```

The training uses the Minist data set, 70,000 small 28x28 pixel pictures, the pictures are handwritten 0-9 numbers. The next step is data acquisition, data processing and other operations, Sagemaker quickly completed.

The imported data is divided into three categories, one is training data, the other is verification data, and the other is test data.

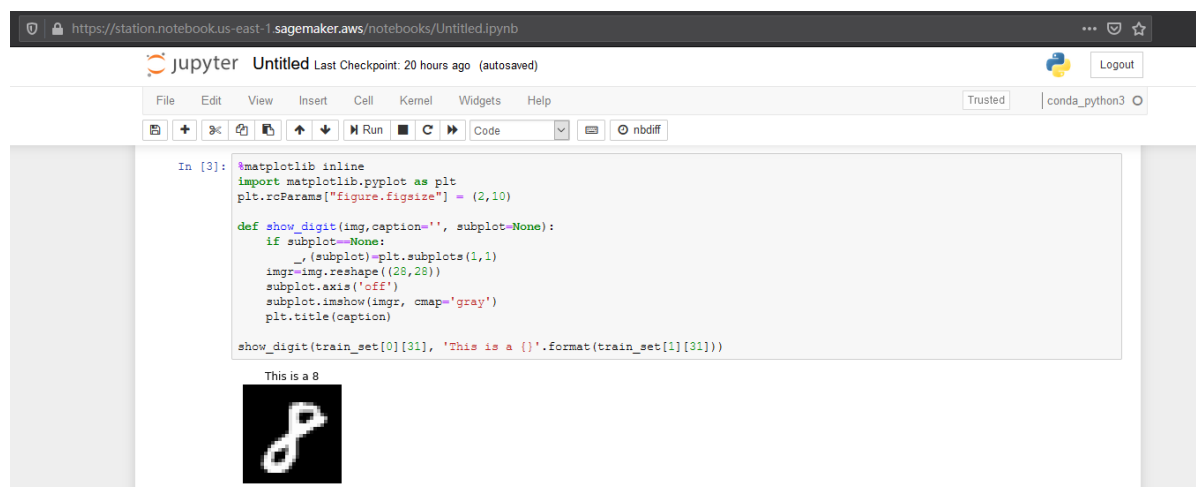


```
In [2]: %time
import pickle, gzip, numpy, urllib.request, json

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')

CPU times: user 903 ms, sys: 261 ms, total: 1.16 s
Wall time: 2.19 s
```

The data is imported successfully, and then in the next step, look at the composition of the data and see what the picture looks like.




```
In [3]: %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')
    plt.title(caption)

show_digit(train_set[0][31], 'This is a {}'.format(train_set[1][31]))

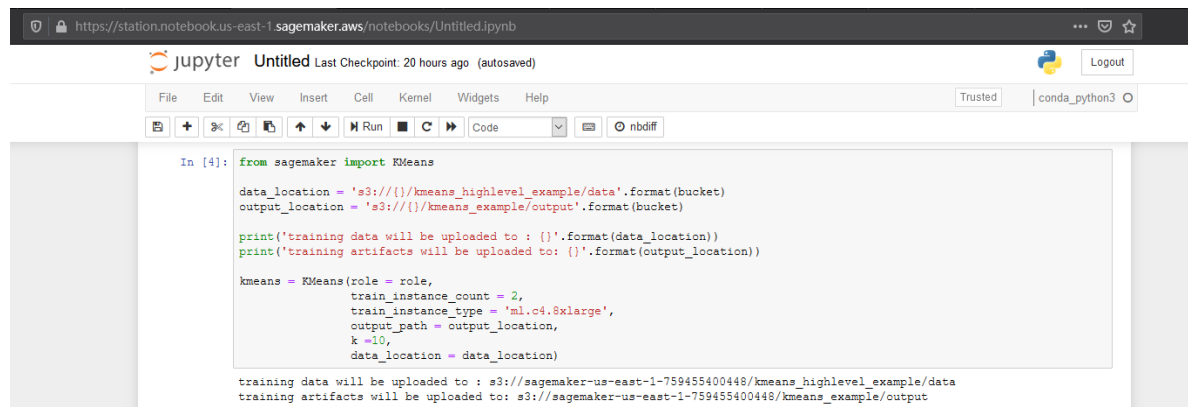
This is a 8
```



The above steps are usually the work of machine learning data processing, because the data itself is very standardized, so our technology did not do anything, because it is a clustering classification algorithm, so there is no need to tag.

Start training

Sagemake of AWS implements its own kmeans. In this operation step, the basic framework of kmeans is constructed, two instances for training are created, and the input data and output data path are set. The value of k is set to 10. The value of k is very critical, which means that the trained model is divided into 10 clusters, and all data is in these ten clusters.



```
In [4]: from sagemaker import KMeans

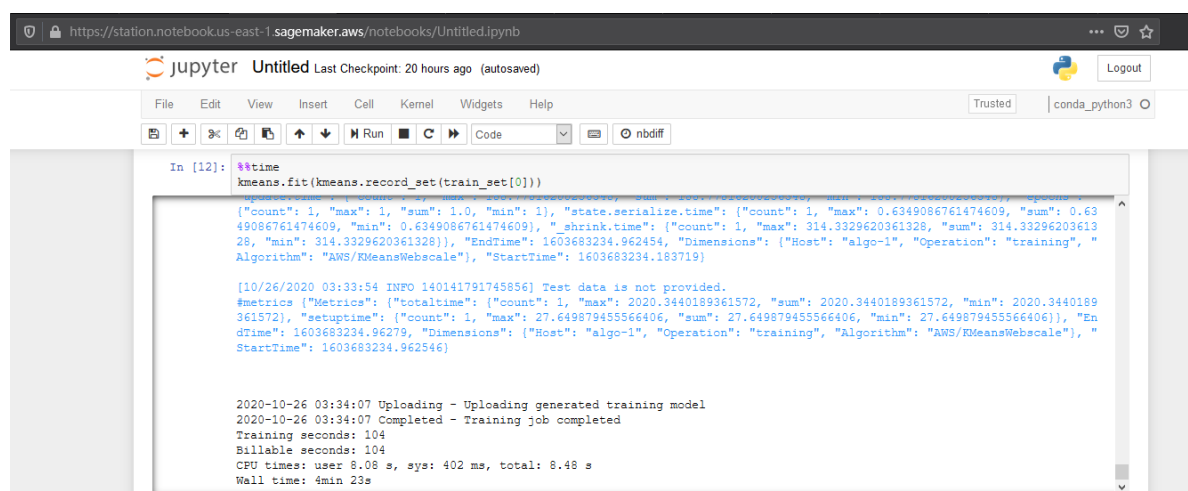
data_location = 's3://{}/kmeans_highlevel_example/data'.format(bucket)
output_location = 's3://{}/kmeans_example/output'.format(bucket)

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)

training data will be uploaded to : s3://sagemaker-us-east-1-759455400448/kmeans_highlevel_example/data
training artifacts will be uploaded to: s3://sagemaker-us-east-1-759455400448/kmeans_example/output
```

Call kmeans's .fit function for training. This .fit method should also be defined by sagemaker itself. The trained model is also uploaded, making the original simple kmens algorithm simpler.



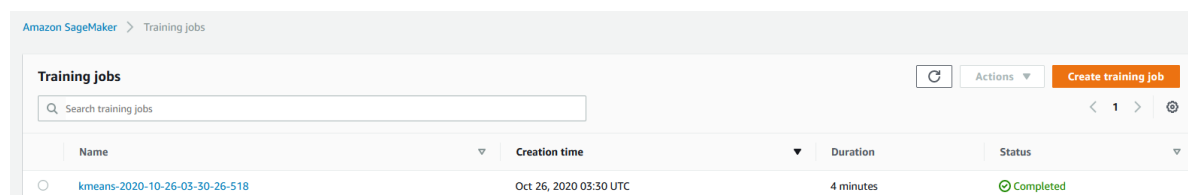
```
In [12]: %time
kmeans.fit(kmeans.record_set(train_set[0]))

[{"count": 1, "max": 1, "sum": 1.0, "min": 1}, {"state.serialize.time": {"count": 1, "max": 0.6349086761474609, "sum": 0.6349086761474609, "min": 0.6349086761474609}, {"_shrink.time": {"count": 1, "max": 314.3329620361328, "sum": 314.3329620361328, "min": 314.3329620361328}}, {"EndTime": 1603683234.962454, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "AWS/KMeansWebscale"}, {"StartTime": 1603683234.183719}]

[10/26/2020 03:33:54 INFO 140141791745856] Test data is not provided.
#metrics {"Metrics": {"totaltime": {"count": 1, "max": 2020.3440189361572, "sum": 2020.3440189361572, "min": 2020.3440189361572}, {"setuptime": {"count": 1, "max": 27.649879455566406, "sum": 27.649879455566406, "min": 27.649879455566406}}, {"EndTime": 1603683234.96279, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "AWS/KMeansWebscale"}, {"StartTime": 1603683234.962546}]

2020-10-26 03:34:07 Uploading - Uploading generated training model
2020-10-26 03:34:07 Completed - Training job completed
Training seconds: 104
Billable seconds: 104
CPU times: user 8.08 s, sys: 402 ms, total: 8.48 s
Wall time: 4min 23s
```

In console of Sagemaker, a training job takes 4 minutes.



Training jobs			
<input type="text" value="Search training jobs"/>			
Name	Creation time	Duration	Status
kmeans-2020-10-26-03-30-26-518	Oct 26, 2020 03:30 UTC	4 minutes	Completed

Click it and we can see more information about it,

Amazon SageMaker > Training jobs > kmeans-2020-10-26-03-30-26-518

kmeans-2020-10-26-03-30-26-518

Clone Create model package Stop Create model

Job settings

Job name kmeans-2020-10-26-03-30-26-518	Status Completed View history	SageMaker metrics time series Disabled	IAM role ARN arn:aws:iam::[redacted]:role/service-role/AmazonSageMaker-ExecutionRole-[redacted]
ARN arn:aws:sagemaker:us-east-1:[redacted]:job/kmeans-2020-10-26-03-30-26-518	Creation time Oct 26, 2020 03:30 UTC	Training time (seconds) 52	
	Last modified time Oct 26, 2020 03:34 UTC	Billable time (seconds) 52	
		Managed spot training savings 0%	
		Tuning job source/parent -	

Algorithm

Algorithm ARN -	Instance type ml.m4.xlarge	Additional volume size (GB) 30	Volume encryption key -
Training image 382416733822.dkr.ecr.us-east-1.amazonaws.com/kmeans:1	Instance count 2	Maximum runtime (s) 86400	
Input mode File		Maximum wait time for managed spot training(s) -	
		Managed spot training Disabled	

In the Sagemaker console, you can see the model you just trained.

Amazon SageMaker > Models

Models

Create endpoint Create endpoint configuration Actions Create model

Search models

Name	ARN	Creation time
kmeans-2020-10-26-03-30-26-518	arn:aws:sagemaker:us-east-1:759455400448:model/kmeans-2020-10-26-03-30-26-518	Oct 26, 2020 03:34 UTC

Deploy

Next step is to deploy the trained model on the m4.xlarge instance, which takes 8 minutes.

https://station.notebook.us-east-1.sagemaker.aws/notebooks/Untitled.ipynb

jupyter Untitled Last Checkpoint: 20 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted conda_python3

```
In [13]: %%time
kmeans_predictor = kmeans.deploy(initial_instance_count=1,
                                  instance_type='ml.m4.xlarge')
Parameter image will be renamed to image_uri in SageMaker Python SDK v2.
-----!CPU times: user 311 ms, sys: 9.65 ms, total: 321 ms
Wall time: 8min 32s
```

Bring a piece of data into the model, look at which cluster the data is closest to, and see how far it is from the closest cluster.

```
https://station.notebook.us-east-1.sagemaker.aws/notebooks/Untitled.ipynb
jupyter Untitled Last Checkpoint: 20 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted conda_python3

In [17]: result = kmeans_predictor.predict(train_set[0][30:32])
print(result)

[{'label': {'key': 'closest_cluster', 'value': {'float32_tensor': {'values': 0.0}}}},
 {'label': {'key': 'distance_to_cluster', 'value': {'float32_tensor': {'values': 6.899248600061035}}}},
 {'label': {'key': 'closest_cluster', 'value': {'float32_tensor': {'values': 5.0}}}},
 {'label': {'key': 'distance_to_cluster', 'value': {'float32_tensor': {'values': 7.284411430358887}}}}]]
```

Result

Find out 100 verification data, divide them into 0-9, 10 categories, 10 clusters. If the numbers in a cluster are the same, it proves that the recognition is not bad.

```
https://station.notebook.us-east-1.sagemaker.aws/notebooks/Untitled.ipynb
jupyter Untitled Last Checkpoint: 20 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted conda_python3

In [15]: %%time
result = kmeans_predictor.predict(train_set[0][0:100])
clusters = [r.label['closest_cluster'].float32_tensor.values[0] for r in result]

CPU times: user 26.9 ms, sys: 0 ns, total: 26.9 ms
Wall time: 86.5 ms

In [16]: 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()
```

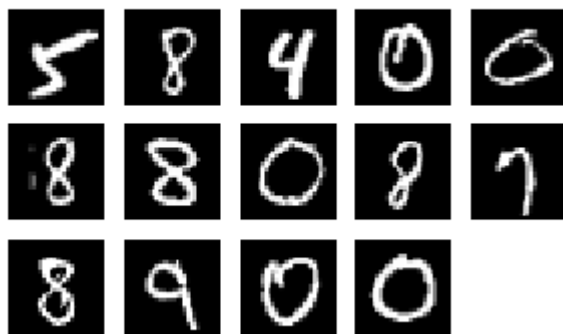
Cluster 3



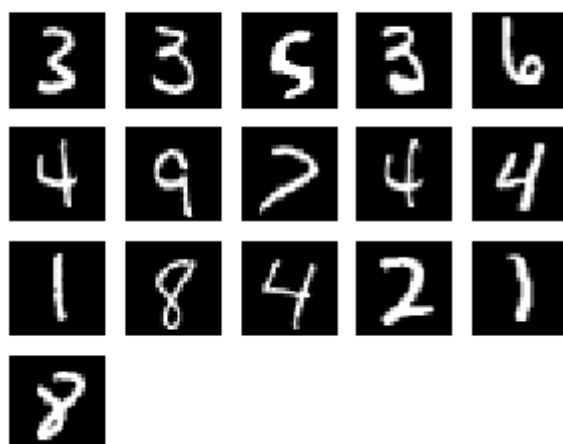
Cluster 0



Cluster 2



Cluster 5



Cluster 4



End

Finally, don't forget to delete the training instance you created.

```
In [18]: print(kmeans_predictor.endpoint)
kmeans-2020-10-26-03-30-26-518

In [ ]: import sagemaker
sagemaker.Session().delete_endpoint(kmeans_predictor.endpoint)
```