



Amazon SageMaker

Amazon SageMaker is a fully managed machine learning (ML) service provided by AWS that helps developers and data scientists to quickly and efficiently build, train, and deploy ML models at scale. It is designed to simplify the complex steps involved in the ML lifecycle—data preparation, model building, training, tuning, and deployment—without the need to manage underlying infrastructure.

Core Features of Amazon SageMaker:

1. **SageMaker Studio:** A web-based integrated development environment (IDE) that allows users to build, train, and deploy models from a single interface. It supports the full ML workflow, including code editing, experimentation, training, tuning, and monitoring.
2. **SageMaker Autopilot:** Automatically builds, trains, and tunes the best machine learning models based on your data, allowing you to get started without deep ML expertise.
3. **SageMaker Data Wrangler:** Helps streamline the data preparation process, allowing you to import, transform, and analyze your data quickly.
4. **SageMaker Experiments:** Tracks and organizes experiments, helping you compare different model versions and settings to find the best-performing model.
5. **SageMaker Model Monitor:** Continuously monitors deployed models for data and performance drift, ensuring the model predictions stay accurate over time.
6. **SageMaker Debugger:** Provides insights into model training, automatically identifying and notifying you of issues such as overfitting, so you can take corrective actions.
7. **SageMaker Pipelines:** Helps automate ML workflows by defining, automating, and scaling end-to-end ML pipelines, from data preparation to model training and deployment.
8. **Built-in Algorithms:** SageMaker provides optimized implementations of popular machine learning algorithms such as XGBoost, k-means clustering, linear regression, and image classification.
9. **Elastic Inference:** Allows you to reduce the cost of deploying machine learning models by scaling inference acceleration across different instance types.

Advantages of Amazon SageMaker:

1. **End-to-End Machine Learning Workflow:** SageMaker covers all stages of the machine learning pipeline, from data preparation and model training to deployment and monitoring, which helps developers manage the full lifecycle in one platform.
2. **Managed Infrastructure:** SageMaker abstracts the complexity of managing and scaling the underlying infrastructure for training and deploying models, so developers can focus on the ML part without worrying about resource allocation or server setup.

3. **Supports Multiple Frameworks:** SageMaker supports popular ML and deep learning frameworks like TensorFlow, PyTorch, MXNet, and Scikit-learn. Users can also bring their own algorithms and custom frameworks.
4. **Pay-As-You-Go:** You only pay for the resources you use, such as computing power for training or inference, without the need to invest in expensive on-premises hardware.
5. **Distributed Training:** SageMaker can distribute large training jobs across multiple machines, allowing faster training for complex models with large datasets.
6. **Automatic Model Tuning (Hyperparameter Optimization):** SageMaker's hyperparameter tuning automatically searches for the best model parameters, helping to optimize model performance.
7. **Flexible Deployment Options:** SageMaker enables you to deploy models in various ways, such as real-time endpoints for interactive applications, batch transformations for large datasets, or multi-model endpoints to host multiple models on the same instance.

Use Cases of Amazon SageMaker:

1. **Financial Services (Fraud Detection):** SageMaker can be used to build models that detect fraudulent transactions in real-time, helping banks and financial institutions reduce fraud risk by analyzing patterns of behavior.
2. **Healthcare (Medical Imaging):** In healthcare, SageMaker can help build models that analyze medical images (e.g., X-rays, MRIs) to detect diseases or abnormalities like tumors, automating and speeding up the diagnostic process.
3. **Retail (Personalized Recommendations):** Retailers can use SageMaker to analyze customer behavior and purchase history to provide personalized product recommendations, increasing sales and customer satisfaction.
4. **Manufacturing (Predictive Maintenance):** SageMaker can help manufacturers predict equipment failure by analyzing machine sensor data, reducing downtime and maintenance costs.
5. **Autonomous Vehicles (Model Training for Self-driving Cars):** Companies working on autonomous driving can use SageMaker to train deep learning models that process sensor data from cameras and LiDAR to help vehicles navigate.

Disadvantages of Amazon SageMaker:

1. **Complexity for Beginners:** Although SageMaker simplifies many aspects of ML, it still requires a certain level of ML and AWS expertise to use effectively, especially when dealing with custom models and advanced features.
2. **Cost Management:** While SageMaker offers cost-efficiency, managing costs can become a challenge, particularly for long-running training jobs, large datasets, or high-demand inference models. Mismanaging resource allocation (e.g., using large, expensive instances unnecessarily) can result in higher-than-expected costs.
3. **Dependency on AWS Ecosystem:** SageMaker works best within the AWS ecosystem. If you are using non-AWS tools or services, integration may be more challenging.

Migrating workloads or integrating with third-party cloud providers can add complexity.

4. **Limited Control for Custom Solutions:** While SageMaker's pre-built tools and algorithms are powerful, some users might find them restrictive if they need more granular control or customization over their ML pipelines, such as specific hardware choices or algorithm optimizations.

Example:

Example 1: Building a Sentiment Analysis Model

Let's say an e-commerce company wants to understand customer feedback by analyzing product reviews. Using Amazon SageMaker, they can:

1. Import customer review data from Amazon S3.
2. Use SageMaker's built-in natural language processing (NLP) algorithms to train a sentiment analysis model.
3. Deploy the model using SageMaker's real-time endpoint feature.
4. Continuously monitor the model using SageMaker Model Monitor to ensure its predictions remain accurate.

Example 2: Image Recognition for Quality Control

A manufacturing company wants to detect defects in products from images taken on the assembly line. Using SageMaker:

1. They upload images of products to Amazon S3.
2. Use SageMaker to train a custom image classification model that can detect whether a product is defective or not.
3. Deploy the model to detect defects in real-time, improving the quality control process and reducing human error.

Conclusion:

Amazon SageMaker is a powerful service for managing machine learning workflows at scale, providing tools for data preparation, model training, tuning, deployment, and monitoring. It offers benefits like scalability, cost efficiency, and ease of integration with other AWS services. However, it requires some ML and cloud computing expertise to use effectively and can become costly if not managed well.

To begin with the Lab:

1. In this lab, we are going to launch a SageMaker Notebook instance and then we are going to upload our notebook. We will push the datafile that we got into our S3 bucket and then we are going to consume the data from the S3 bucket.
2. We will prepare the training data that is required for us in order to perform the fit on our model and we will push the data back to our S3 bucket.

3. And once we perform the training, we will then push the model artifacts back into our S3 bucket.
4. So, we'll be using this S3 bucket as a source and as a place where we can store all those data files and we can also perform the versioning if you want to capture the lineage.
5. You will also have a folder with this lab which contains some files that we need to use for this lab.
6. Now go to your AWS Console and search for Amazon SageMaker and the service accordingly.

7. In SageMaker from the left pane expand Application and IDEs and choose Notebooks. Then click on create notebook instances.

8. Now give your notebook a name and keep instance type and platform identifier to default.

Create notebook instance

Amazon SageMaker provides pre-built fully managed notebook instances that run Jupyter notebooks. The notebook instances include example code for common model training and hosting exercises. [Learn more](#)

Notebook instance settings

Notebook instance name

hepatitis-notebook

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

Notebook instance type

ml.t3.medium

Platform identifier [Learn more](#)

Amazon Linux 2, Jupyter Lab 3

► Additional configuration

9. For Permission and Encryption section you need to click on create new IAM role and then choose the same settings as shown below and create you new role.

Create an IAM role

X

Passing an IAM role gives Amazon SageMaker permission to perform actions in other AWS services on your behalf. Creating a role here will grant permissions described by the [AmazonSageMakerFullAccess](#) IAM policy to the role you create.

The IAM role you create will provide access to:

S3 buckets you specify - *optional*

Any S3 bucket

Allow users that have access to your notebook instance access to any bucket and its contents in your account.

Specific S3 buckets

Example: `bucket-name-1, bucket-name-2, bl`

Comma delimited. ARNs, "*" and "/" are not supported.

None

Any S3 bucket with "sagemaker" in the name

Any S3 object with "sagemaker" in the name

Any S3 object with the tag "sagemaker" and value "true"

[See Object tagging](#)

S3 bucket with a Bucket Policy allowing access to SageMaker

[See S3 bucket policies](#)

Cancel

Create role

10. After that keep rest of the things to default and create your Notebook. It will take at least 5 minutes to be in ready state.

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.

AmazonSageMaker-ExecutionRole-20241018T153327



Success! You created an IAM role.

AmazonSageMaker-ExecutionRole-20241018T153327



Create role using the role creation wizard

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.

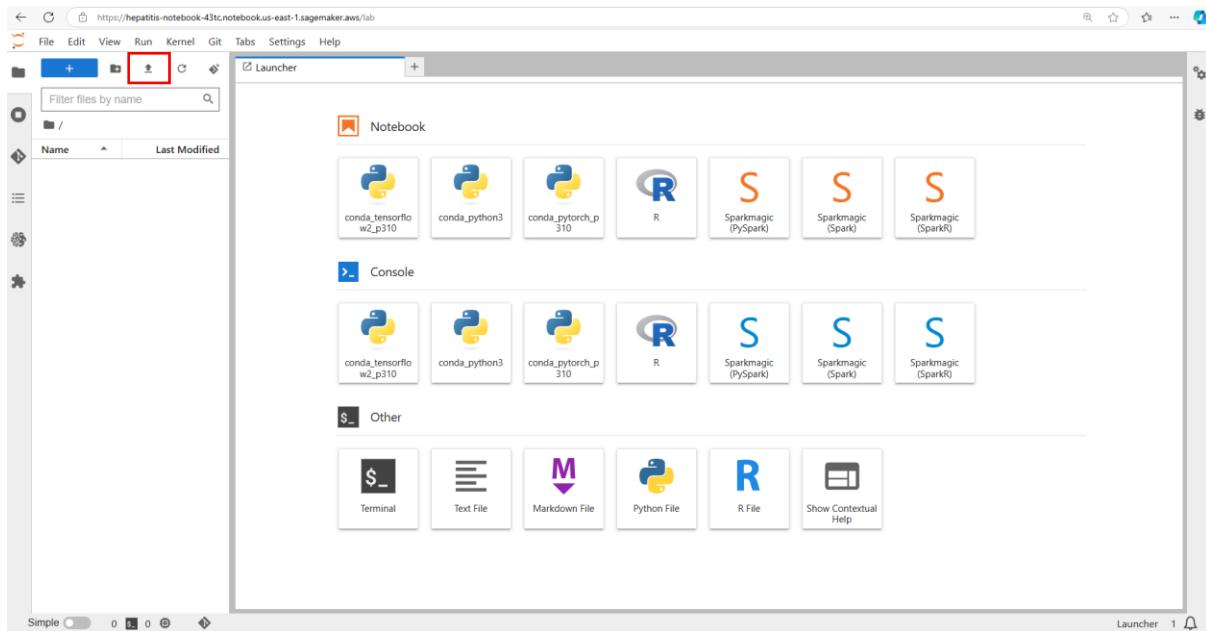
No Custom Encryption

11. Below you can see that your notebook has been created. Now click on JupyterLab to open the Jupyter Notebook.

Amazon SageMaker > Notebook instances

Notebook instances		Info		Actions		Create notebook instance	
		<input type="text"/> Search notebook instances					
Name	Instance	Creation time	Status	Actions			
<input type="radio"/> hepatitis-notebook	ml.t3.medium	10/18/2024, 3:35:13 PM	InService	Open Jupyter	Open JupyterLab		

12. This is how the Notebook would look like. So, it is similar to Jupyter Notebook interface but if you look at the link you can say that this notebook is powered by AWS.
13. Now we are going to click on upload icon which is highlighted.



14. Then choose the Hepatitis Project Code and upload it to your notebook.

Name	Date modified	Type	Size
hepatitis	18-10-2024 12:43	Microsoft Excel W...	21 KB
Hepatitis-Project-Code	18-10-2024 12:43	Jupyter Source File	134 KB
lambda_function	18-10-2024 12:43	Python Source File	1 KB
new_data	18-10-2024 12:43	JSON Source File	1 KB
README	18-10-2024 12:43	Markdown Source...	1 KB

15. Below you can see that our project code have been uploaded and this is how our notebook would look like.

```

[1]: import os
import io

import boto3
import pandas as pd
import numpy as np
from datetime import datetime

import sagemaker
import sagemaker.amazon.common as smac
from sagemaker import get_execution_role

import seaborn as sns

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score,f1_score,recall_score,precision_score,classification_report,confusion_matrix

# Code to ignore warnings
import warnings
warnings.filterwarnings("ignore")

[2]: role = get_execution_role()
buckets='awsml-730qe'
[3]: data_key = 'hepatitis.xlsx'
data_location = f's3://{buckets}/{data_key}'

[4]: data = pd.read_excel(data_location)
data.head()

```

16. Next thing, you need to create an S3 bucket and upload the hepatitis.xlsx file to it.

Amazon S3 > Buckets > awsml-demo12

awsml-demo12 Info

[Objects](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (1) Info

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	hepatitis.xlsx	xlsx	October 18, 2024, 15:51:04 (UTC+05:30)	20.5 KB	Standard

17. Now we just need to execute our Notebook. First things first, import the necessary libraries.

```
[1]: import os
import io

import boto3
import pandas as pd
import numpy as np
from datetime import datetime

import sagemaker
import sagemaker.amazon.common as smac
from sagemaker import get_execution_role

import seaborn as sns

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score,f1_score,recall_score,precision_score,classification_report,confusion_matrix

# Code to ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

18. Then we specified our bucket name and get the execution role attached with our SageMaker. After that we accessed the file from our S3 bucket.

```
[2]: role = get_execution_role()
bucket='awsml-demo12' #specify the bucket name
```

```
•[3]: data_key = 'hepatitis.xlsx'
data_location = f's3://awsml-demo12/hepatitis.xlsx' #S3 object URI
```

19. Here we read few entries from our dataset and check its shape for rows and columns.

```
[4]: data = pd.read_excel(data_location)
data.head()
```

ID	target	age	gender	steroid	antivirals	fatigue	malaise	anorexia	liverBig	spleen	spiders	ascites	varices	bili	alk	sgot	albu	protine	histology
0	1	1	30	2	1	2	2	2	2	1 ...	2	2	2	2	1.0	85	18	4.0	61	1
1	2	1	50	1	1	2	1	2	2	1 ...	2	2	2	2	0.9	135	42	3.5	61	1
2	3	1	78	1	2	2	1	2	2	2 ...	2	2	2	2	0.7	96	32	4.0	61	1
3	4	1	31	1	1	1	2	2	2	2 ...	2	2	2	2	0.7	46	52	4.0	80	1
4	5	1	34	1	2	2	2	2	2	2 ...	2	2	2	2	1.0	46	200	4.0	61	1

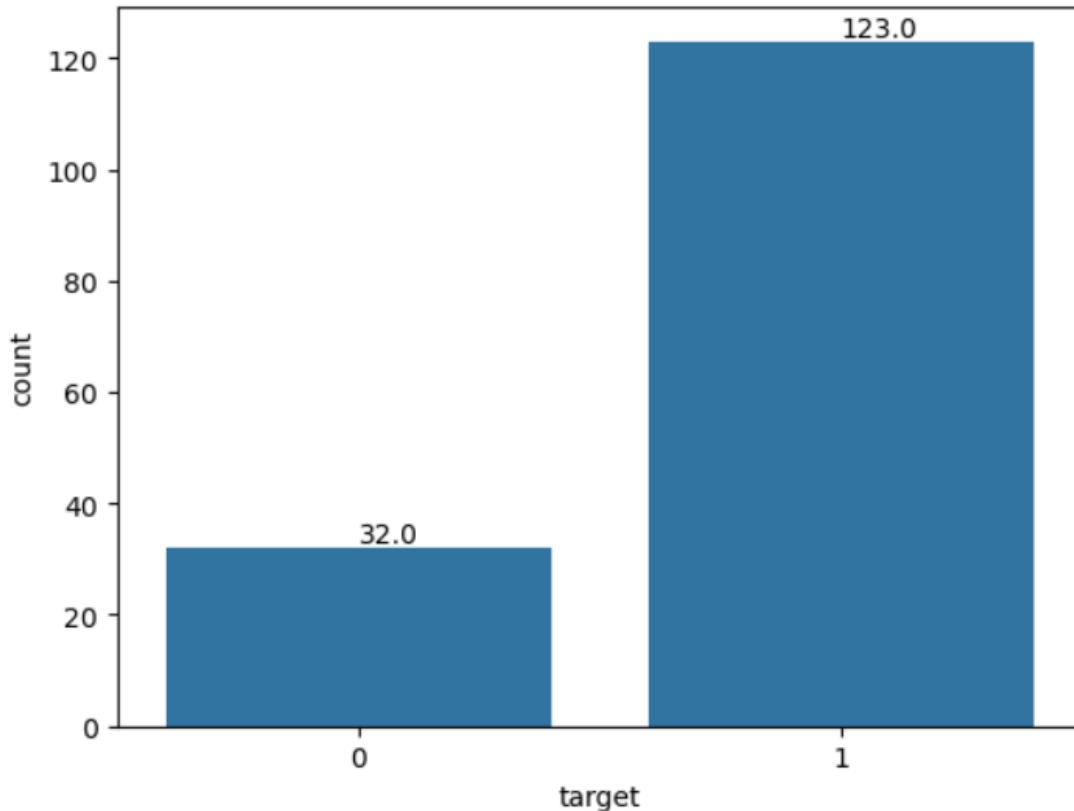
5 rows × 21 columns

```
[5]: print('Dataset has ' + str(data.shape[0]) + ' rows, and ' + str(data.shape[1]) + ' columns')
Dataset has 155 rows, and 21 columns
```

20. We also have the visualization of our data.

```
[7]: y_count=sns.countplot(x='target',data=data)
# Shows the count of observations in each categorical bin using bars

for p in y_count.patches:
    height = p.get_height()
    # Add text to the axes
    y_count.text(p.get_x()+p.get_width()/2, height + 1, height)
# The y_count.text method takes an x position, a y position and a string
```



21. Afte that we dropped the ID column and we set the categorical column and displayed the data types.

```

[8]: data.drop(["ID"], axis=1, inplace=True)

[9]: num_cols = ["age", "bili", "alk", "sgot", "protimes"]
cat_cols = ['gender', 'steroid', 'antivirals', 'fatigue', 'malaise', 'anorexia', 'liverBig', 'liverFirm', 'spleen', 'spiders', 'ascites', 'varices', 'histology']

[10]: data[cat_cols] = data[cat_cols].astype('category')

[11]: data.dtypes

[11]: target          int64
      age            int64
      gender         category
      steroid        category
      antivirals     category
      fatigue        category
      malaise        category
      anorexia       category
      liverBig       category
      liverFirm      category
      spleen         category
      spiders        category
      ascites        category
      varices        category
      bili           float64
      alk            int64
      sgot           int64
      albu           float64
      protimes       int64
      histology      category
      dtype: object

```

22. We performed the one hot encoding and displayed the few entries.

```

[13]: # OneHotEncoding
data = pd.get_dummies(data, columns=['gender', 'steroid', 'antivirals', 'fatigue', 'malaise', 'anorexia', 'liverBig', 'liverFirm', 'spleen', 'spiders', 'ascites', 'varices', 'histology'], drop_first=True)

[14]: display(data.head())
display(data.shape)

```

	target	age	bili	alk	sgot	albu	protimes	gender_2	steroid_2	antivirals_2	fatigue_2	malaise_2	anorexia_2	liverBig_2	liverFirm_2	spleen_2	spiders_2	ascites
0	1	30	1.0	85	18	4.0	61	True	False	True	True	True	True	False	True	True	True	True
1	1	50	0.9	135	42	3.5	61	False	False	True	False	True	True	False	True	True	True	True
2	1	78	0.7	96	32	4.0	61	False	True	True	False	True	True	True	True	True	True	True
3	1	31	0.7	46	52	4.0	80	False	False	False	True	True	True	True	True	True	True	True
4	1	34	1.0	46	200	4.0	61	False	True	True	True	True	True	True	True	True	True	True

(155, 20)

23. After that we split the dataset into two parts. So, before we use this data we need to perform a kind of formatting, so for that we are going to generate this training data and we are going to place it in our S3 bucket.

24. In this case, I'll be converting the data into a protobuf IO format. So this is one of the efficient way to process the dataset for training the sagemaker models.

```
[18]: X_train.shape
[18]: (127, 19)

[19]: X_test.shape
[19]: (28, 19)

[20]: train_file = 'training_data'
f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, X_train.astype('float32'), y_train.astype('float32'))
f.seek(0)

boto3.Session().resource('s3').Bucket(bucket).Object(f'{train_file}').upload_fileobj(f)
training_recordIO_protobuf_location = f's3://{bucket}/{train_file}'
print('The Pipe mode recordIO protobuf training data: {}'.format(training_recordIO_protobuf_location))

The Pipe mode recordIO protobuf training data: s3://awsml-demo12/training_data

[21]: validation_file = 'validation_data'
f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, X_test.astype('float32'), y_test.astype('float32'))
f.seek(0)

boto3.Session().resource('s3').Bucket(bucket).Object(f'{validation_file}').upload_fileobj(f)
validate_recordIO_protobuf_location = f's3://{bucket}/{validation_file}'
print('The Pipe mode recordIO protobuf validation data: {}'.format(validate_recordIO_protobuf_location))

The Pipe mode recordIO protobuf validation data: s3://awsml-demo12/validation_data
```

25. After running the command if you go to your S3 bucket you will see that a training data and validation data files has been created.

Name	Type	Last modified	Size	Storage class
hepatitis.xlsx	xlsx	October 18, 2024, 15:51:04 (UTC+05:30)	20.5 KB	Standard
training_data	-	October 18, 2024, 16:01:51 (UTC+05:30)	14.9 KB	Standard
validation_data	-	October 18, 2024, 16:11:56 (UTC+05:30)	3.3 KB	Standard

26. Then just run all the commands until you see the endpoint name.
27. The code is setting up a machine learning training job using the K-Nearest Neighbors (KNN) algorithm on AWS SageMaker. It starts by identifying the necessary container for the KNN algorithm. Then, it creates a unique name for the training job based on the current date and time.
28. Next, it specifies where the trained model will be stored in an S3 bucket. The code also defines the number of features in the dataset and sets the hyperparameters for the KNN model, like the number of neighbors to consider.
29. A SageMaker session is initiated, and an estimator object is created to manage the training process. The training job is then launched with the specified data for training

and validation. Finally, once the model is trained, it is deployed to an endpoint for making predictions.

```
[22]: from sagemaker import image_uris, model_uris, script_uris
[23]: container = image_uris.retrieve('knn', boto3.Session().region_name, 'latest')
Defaulting to the only supported framework/algorithm version: 1. Ignoring framework/algorithm version: latest.
[24]: # Create a training job name
job_name = 'hepatitis-job3-{}'.format(datetime.now().strftime("%Y%m%d%H%M%S"))
print('Here is the job name {}'.format(job_name))

# Here is where the model-artifact will be stored
output_location = f's3:///{bucket}/model-artifact/'

Here is the job name hepatitis-job3-20241018120509

[25]: print('The feature_dim hyperparameter needs to be set to {}'.format(X_train.shape[1]))
The feature_dim hyperparameter needs to be set to 19.

[26]: X_train.shape
[26]: (127, 19)

[27]: hyperparams = {"feature_dim": 19, "k": 10, "sample_size": 20, "predictor_type": "classifier"}
[28]: sess = sagemaker.Session()

# Setup the KNNLeaner algorithm from the ECR container
knn = sagemaker.estimator.Estimator(container,
                                      role,
                                      instance_count=1,
                                      instance_type='ml.m4.xlarge',
                                      output_path=output_location,
                                      sagemaker_session=sess,
                                      input_mode='Pipe')

[29]: # Setup the hyperparameters
knn.set_hyperparameters(feature_dim=19,
                       k=3,
                       sample_size=20,
                       predictor_type='classifier')

[30]: # Launch a training job. This method calls the CreateTrainingJob API call
data_channels = {'train': training_recordIO_protobuf_location,
                 'validation': validate_recordIO_protobuf_location}

[31]: knn.fit(data_channels, job_name=job_name)

INFO:sagemaker:Creating training-job with name: hepatitis-job3-20241018120509
2024-10-18 12:05:12 Starting - Starting the training job...
2024-10-18 12:05:26 Starting - Preparing the instances for training...
2024-10-18 12:06:00 Downloading - Downloading input data...
2024-10-18 12:06:25 Downloading - Downloading the training image.....
2024-10-18 12:11:03 Training - Training image download completed. Training in progress....
2024-10-18 12:11:46 Uploading - Uploading generated training model
2024-10-18 12:11:46 Completed - Training job completed
```

```
[32]: print('Here is the location of the trained KNN model: {} / {} / output / model.tar.gz'.format(output_location, job_name))

Here is the location of the trained KNN model: s3://awsml-demo12/model-artifact/hepatitis-job3-20241018120509/output/model.tar.gz

[33]: hepatitis_predictor = knn.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')

INFO:sagemaker:Creating model with name: knn-2024-10-18-12-12-22-614
INFO:sagemaker:Creating endpoint-config with name knn-2024-10-18-12-12-22-614
INFO:sagemaker:Creating endpoint with name knn-2024-10-18-12-12-22-614
-----!

[34]: hepatitis_predictor.endpoint_name

[34]: 'knn-2024-10-18-12-12-22-614'
```

30. So, when we run the command on 31st cell we did the KNN fit and by doing that we create a Training job that took 7 minutes to completed. You can see that job in your SageMaker by expanding the training section and going to training jobs.

Name	Creation time	Duration	Job status	Warm pool status	Time left
hepatitis-job3-20241018120509	10/18/2024, 5:35:10 PM	7 minutes	Completed	-	-
hepatitis-job3-20241018104313	10/18/2024, 4:13:33 PM	7 minutes	Completed	-	-

31. When we run the command on 33rd cell we created an endpoint which took almost 15 minutes and make a note of this endpoint.

Name	ARN	Creation time	Status	Last updated
knn-2024-10-18-12-12-22-614	arn:aws:sagemaker:us-east-1:878893308172:endpoint/knn-2024-10-18-12-12-22-614	10/18/2024, 5:42:23 PM	InService	10/18/2024, 5:50:02 PM
knn-2024-10-18-11-14-42-988	arn:aws:sagemaker:us-east-1:878893308172:endpoint/knn-2024-10-18-11-14-42-988	10/18/2024, 4:44:44 PM	InService	10/18/2024, 4:52:00 PM

32. Then in you AWS Console go to Lambda and create a new function using Python 3.11 as your environment.

Function name	Description	Package type	Runtime	Last modified
hepatitis-function	-	Zip	Python 3.11	12 minutes ago

33. Once your function is created then you need to change the code of it. For that you can open the folder you get with this lab and open the lambda function file. Copy and paste the code in your function and click on deploy.

```

1 import os
2 import io
3 import boto3
4 import json
5 import csv
6
7 # grab environment variables
8 ENDPOINT_NAME = os.environ['ENDPOINT_NAME']
9 runtime= boto3.client('runtime.sagemaker')
10
11 def lambda_handler(event, context):
12     print("Received event: " + json.dumps(event, indent=2))
13
14     data = json.loads(json.dumps(event))
15     payload = data['data']
16     print(payload)
17
18     response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
19                                         ContentType="text/csv",
20                                         Body=payload)
21     print(response)
22     result = json.loads(response['Body'].read().decode())
23     print(result)
24     pred = int(result['predictions'][0]['predicted_label'])
25
26     if(pred == 0):
27         return 'Alive'
28     if(pred == 1):
29         return 'die'
30

```

34. Now go to configurations then to Permission open the IAM role attached to your Lambda function then add sage maker full access policy to your role and come back to your lambda function.

Policy name	Type	Attached entities
AmazonSageMakerFullAccess	AWS managed	2
AWSLambdaBasicExecutionRole-07fec...	Customer managed	1

35. After that open the Environment variable tab and add a variable for the endpoint we got in SageMaker as you can see below. We have defined the key in our code and our code will get the value from the environment variable.

Environment variables (1)	
The environment variables below are encrypted at rest with the default Lambda service key.	
<input type="text"/> Find environment variables	
Key	Value
ENDPOINT_NAME	knn-2024-10-18-12-12-22-614

36. Once this is all done come to Test tab and we are going to test our function. Now we need click on create new event for testing and then in the JSON area we need to paste this script. You can find this script in the file as well.

Event JSON

```

1 [{}]
2 "data": "30.0,2.0,1.0,2.0,2.0,2.0,2.0,1.0,2.0,2.0,2.0,2.0,2.0,1.0,85.0,18.0,4.0,1.0,1.0"
3 []

```

37. So, after running the test you can see that our test got executed and we got the output.

	Executing function: succeeded (logs)
▼ Details	
The area below shows the last 4 KB of the execution log.	
<pre>"die"</pre>	
Summary	
Code SHA-256	Execution time
DPpX8wLS1iHLtB3oSD/Yg4v2ft+BR6qN61OHbEL6mV4=	21 minutes ago
Request ID	Function version
9d7f04db-d528-4450-816b-302c5dba0e30	\$LATEST
Init duration	Duration
289.41 ms	258.09 ms
Billed duration	Resources configured
259 ms	128 MB
Max memory used	
79 MB	

38. After that search and navigate to API gateway in your Console and click on build REST API. Just give it a name and then click on create.

Create REST API

API details

New API

Create a new REST API.

Clone existing API

Create a copy of an API in this AWS account.

Import API

Import an API from an OpenAPI definition.

Example API

Learn about API Gateway with an example API.

API name

hepatitis

Description - optional

API endpoint type

Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.

Regional

39. Here you need to create a resource, click on create resource.

Resources

Create resource

/

Resource details

Update documentation

Enable CORS

Path

/

Resource ID

43ze723fg9

Methods (0)

Delete

Create method

Method type

Integration type

Authorization

API key

No methods

No methods defined.

40. Then you need to give a resource name and enable CORS, click on create.

Create resource

Resource details

Proxy resource [Info](#)
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path Resource name

CORS (Cross Origin Resource Sharing) [Info](#)
Create an OPTIONS method that allows all origins, all methods, and several common headers.

[Cancel](#) [Create resource](#)

41. Now choose you hepatitis resource and click on create method.

API Gateway > APIs > Resources - hepatitis (66njkas7jb)

Resources

[Create resource](#) [API actions ▾](#) [Deploy API](#)

Resource details		Delete	Update documentation	Enable CORS
Path <code>/hepatitis</code>	Resource ID 4b19fb			
Methods (1)				
Method type	Integration type	Authorization	API key	
<input type="radio"/> OPTIONS	Mock	None	Not required	Create method

42. You need to choose POST as your method type and scroll down then choose your Lambda function.

Method type

POST

Integration type

Lambda function

Integrate your API with a Lambda function.



HTTP

Integrate with an existing HTTP endpoint.



Mock

Generate a response based on API Gateway mappings and transformations.



AWS service

Integrate with an AWS Service.



VPC link

Integrate with a resource that isn't accessible over the public internet.



Lambda proxy integration

Send the request to your Lambda function as a structured event.

Lambda function

Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1

arn:aws:lambda:us-east-1:878893308172:function:hepa

43. Then choose your POST method and click on Test tab. In the request body paste the same JSON script you used in your Lambda Function.

Method request | Integration request | Integration response | Method response | **Test**

Test method
Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method.

Query strings
param1=value1¶m2=value2

Headers
Enter a header name and value separated by a colon (:). Use a new line for each header.
header1:value1
header2:value2

Client certificate
No client certificates have been generated.

Request body

```
1 [ {  
2   "data": "30.0,2.0,1.0,2.0,2.0,2.0,2.0,1.0,2.0,2.0,2.0,2.0,2.0,1.0,  
     ,85.0,18.0,4.0,1.0,1.0"  
3 } ]
```

44. Below you can see that you got the response.

/hepatitis - POST method test results

Request	Latency ms	Status
/hepatitis	908	200

Response body

```
"die"
```

Response headers

```
{
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-67125dd7-5dc09b320ecb26d675c5ddf6;Parent=177f0a0fe65e1633;Sampled=0;Lineage=1:5f245cdc:0"
}
```

45. Now click on your hepatitis resource and click on Enable CORS. Here you need to enable the same things as you can see in the snapshot.

API Gateway > APIs > Resources - hepatitis (66njkas7jb)

Resources

Resource details	
<input type="button" value="Delete"/> <input type="button" value="Update documentation"/> <input style="border: 2px solid red; padding: 2px; margin-left: 10px;" type="button" value="Enable CORS"/>	
Path	Resource ID /hepatitis 4b19fb

Methods (2)

Method type	Integration type	Authorization	API key
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required

Enable CORS

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.

Gateway responses

API Gateway will configure CORS for the selected gateway responses.

Default 4XX

Default 5XX

Access-Control-Allow-Methods

OPTIONS

POST

46. Then click on Deploy API, choose new stage give it a name and click on Deploy.

Deploy API



Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

Stage

New stage



Stage name

Prod

A new stage will be created with the default settings. Edit your stage settings on the [Stage](#) page.

Deployment description

Cancel

Deploy

47. Here you can see that we got the stage ready. Now copy the Invoke URL.

The screenshot shows the AWS API Gateway Stages page. On the left, there's a sidebar with 'Prod' selected. The main area is titled 'Stage details' for 'Prod'. It shows the following configuration:

- Stage name:** Prod
- Cache cluster:** Inactive
- Default method-level caching:** Inactive
- Rate Info:** -
- Burst Info:** -
- Web ACL:** -
- Client certificate:** -

Below this, the 'Invoke URL' is listed as `https://66njkas7jb.execute-api.us-east-1.amazonaws.com/Prod`. At the bottom, it says 'Active deployment' with the timestamp 's97yc on October 18, 2024, 18:43 (UTC+05:30)'. There are 'Stage actions' and 'Create stage' buttons at the top right.

48. Now open the PostMan in your Laptop. Create a new workspace in it.

49. Then choose POST as your method and paste the URL here then append it with the hepatitis name which is your resource name.

50. Then copy the JSON script and paste it in the body section. Click on Send.

51. Below you can see that you have the response.

The screenshot shows the Postman interface. The left sidebar shows 'My Workspace' with a collection named 'My first collection' containing several requests. The main area shows a POST request to `https://66njkas7jb.execute-api.us-east-1.amazonaws.com/Prod/hepatitis`. The 'Body' tab is selected, showing the following JSON payload:

```
1 {  
2   "data": "30.0,2.0,1.0,2.0,2.0,2.0,1.0,2.0,2.0,2.0,2.0,1.0,85.0,18.0,4.0,1.0,1.0"  
3 }
```

The response tab shows a 200 OK status with a response time of 333 ms and a size of 372 B. The response body is:

```
1 "die"
```

52. Once you are done delete your Endpoints from SageMaker then stop your Notebook and delete it.

53. After that delete your lambda function and your API Gateway.