

End-to-End Machine Learning with Amazon SageMaker

Building, training, and deploying machine learning models with Amazon SageMaker have become pervasive practices in the realms of artificial intelligence and data science. SageMaker provides an extensive suite of tools and services that not only simplify but also optimize the entire machine-learning workflow. This accessibility and efficiency make it equally valuable for both newcomers and seasoned practitioners.

From the crucial steps of data preprocessing and model training to the intricate process of hyperparameter tuning and the deployment of models to scalable production environments, SageMaker offers a seamless and scalable solution that caters to all your machine learning requirements. Its adaptability is further augmented by seamless integration with various AWS services, solidifying its position as the go-to choice among data scientists and engineers seeking to develop robust and dependable machine-learning solutions.

In this document, we will delve into the creation of a custom machine learning model using scikit-learn (sklearn) on an open-source Kaggle dataset related to advertising. We will also explore the crucial aspects of model training and hyperparameter tuning, ensuring that our model reaches its full potential in terms of accuracy and performance. Last but not least, we will guide you through the deployment process, illustrating how to create an endpoint and leverage this endpoint to make predictions on test data, thereby completing the end-to-end machine-learning pipeline with Amazon SageMaker.

Steps Involved

To implement this solution, complete the following high-level steps:

1. Create an S3 bucket and upload the data
2. Creating Sagemaker Domain and notebook instance
3. Building Model in notebook instance
4. Training the Model and deploying it for prediction
5. Hyperparameter Tuning and updating the prediction

Prerequisite

1. Python
2. Sagemaker
3. Machine learning library - Scikitlearn
4. Jupyter Notebooks

Code

The code files and the current PDF can be found in the Bitbucket repository [here](#)

Step 1: Create an S3 bucket and upload the data

1. The project uses the existing bucket (**bucket--ganesh**) instead of creating a new one in case not sure how to create an S3 Bucket here is an example
 - a. Open the [Amazon S3 console](#) and select the **Buckets** page

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with navigation links like 'Buckets', 'Access Points', 'Object Lambda Access Points', etc. The main area displays a table of buckets. The table has columns for 'Name', 'AWS Region', 'Access', and 'Creation date'. One row is highlighted with a red border. The 'Access' column for the highlighted row shows 'Bucket and objects not public'. The 'Creation date' column shows 'September 28, 2023, 04:37:08 (UTC-02:00)'. A large orange 'Create bucket' button is located at the top right of the main content area.

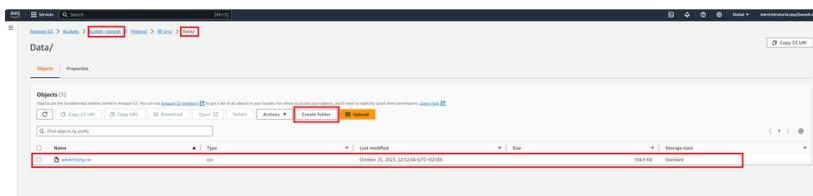
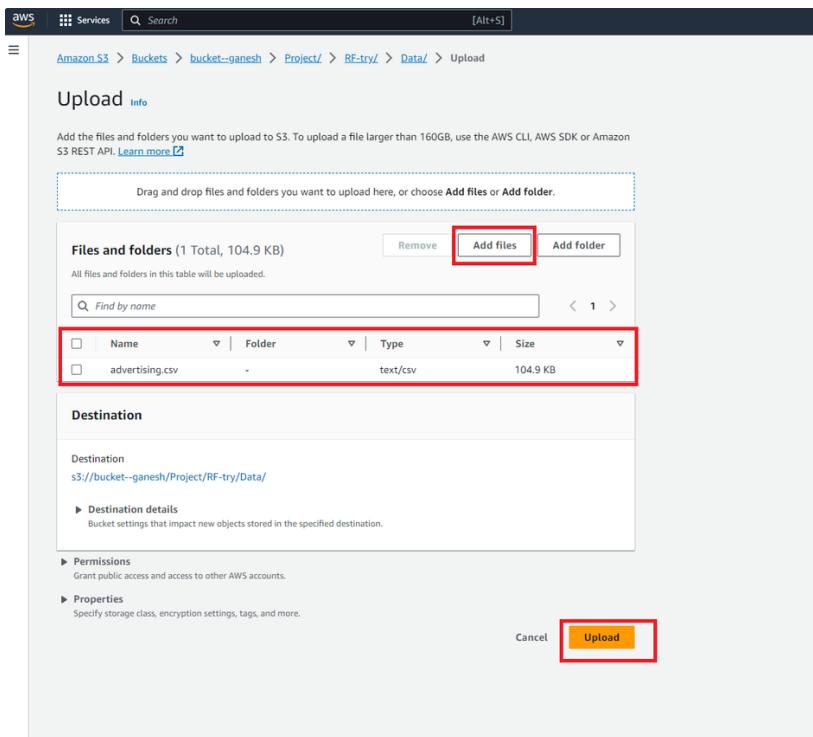
Name	AWS Region	Access	Creation date
advosense-elb-virginia-logs	US East (N. Virginia) us-east-1	Objects can be public	September 7, 2023, 10:39:14 (UTC-02:00)
elasticsearch-logs-us-east-2-977473325382	US East (Ohio) us-east-2	Objects can be public	February 8, 2022, 19:39:41 (UTC-01:00)
prod-advosense-cert	Canada (Central) ca-central-1	Objects can be public	June 15, 2022, 15:26:17 (UTC-02:00)
advosense-prod-elb-logs	Canada (Central) ca-central-1	Bucket and objects not public	June 14, 2022, 14:45:15 (UTC-02:00)
aws-athena-query-results-977473325382-us-east-1	US East (N. Virginia) us-east-1	Bucket and objects not public	October 13, 2023, 14:15:58 (UTC-02:00)
aws-cloudtrail-logs-977473325382-48cfcf32	US East (N. Virginia) us-east-1	Bucket and objects not public	October 13, 2023, 15:47:09 (UTC-02:00)
bucket-ganesh	Europe (Frankfurt) eu-central-1	Bucket and objects not public	August 30, 2023, 16:29:27 (UTC-02:00)
capseed.advosense	US East (N. Virginia) us-east-1	Bucket and objects not public	September 28, 2023, 04:37:08 (UTC-02:00)
bucket-indata		Not found	October 18, 2023, 16:43:48 (UTC-02:00)

- b. Choose **Create bucket** to load the input data

c. Under **General configuration**, do the following:

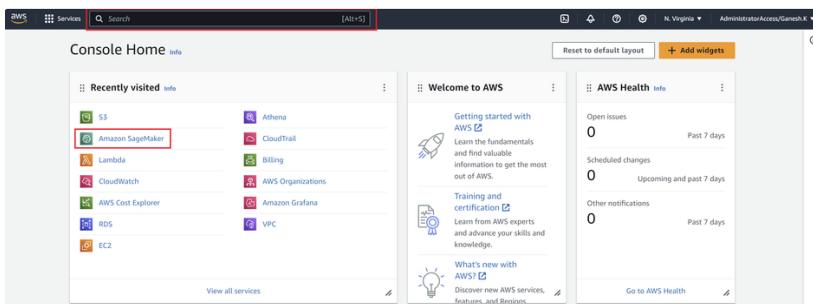
- For the **Bucket name**, enter a globally unique name that meets the Amazon S3 [Bucket naming rules](#). Bucket names can contain only lowercase letters, numbers, dots (.), and hyphens (-).
- For **AWS Region**, choose a Region. Later in the tutorial, you must create your Lambda function in the same Region.
- Leave all other options set to their default values and choose **Create bucket**.

- Upload the **data**, in the **bucket** after creating a **Data** folder in it. Creating a folder is simple, go to the respective bucket click the **create folder** option give the name **Data**, and create it like creating a folder in the computer. Now navigate inside the folder click **upload** and add the data file to **upload**.

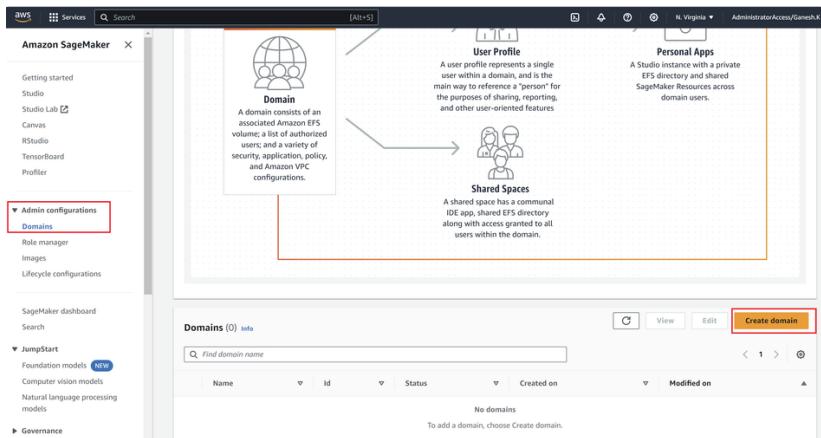


Step 2: Creating Sagemaker Domain and notebook instance

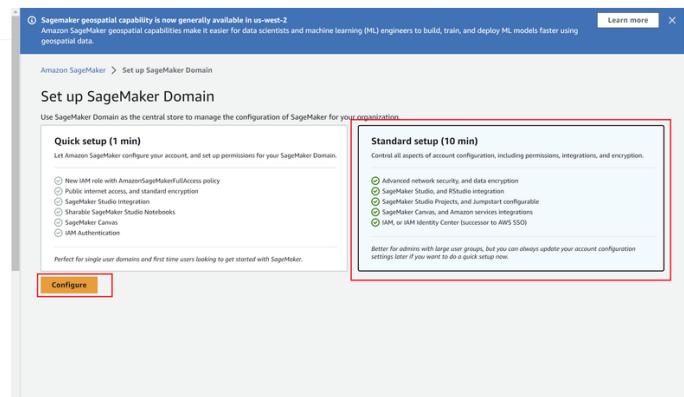
1. Open the [Amazon S3 console](#) and select the **Sagemaker** or search for it in the above search bar



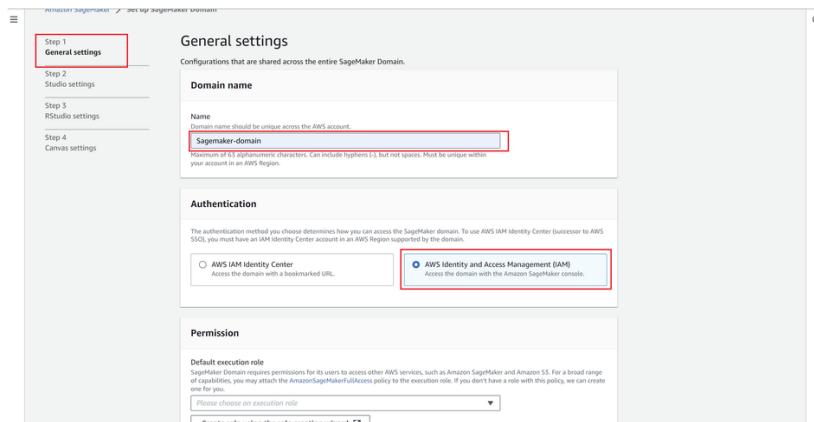
2. In the Sagemaker console, select **Domains** under Admin Configuration



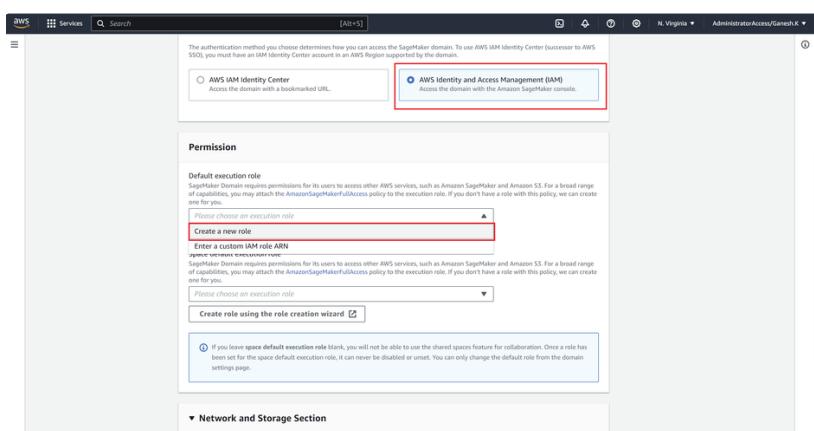
3. In the Set up SageMaker Domain page select Standard setup and click Configure



4. In the Configuration page, give a name for the domain



5. Under permission create a new role, in the new role page select any s3 bucket, so it can access any buckets in the same region



Authentication

The authentication method you choose determines how you access the SageMaker console. To use AWS IAM Identity Center (formerly known as AWS Single Sign-On), you must have an AWS account.

AWS IAM Identity Center
Access the domain with a federated role.

AWS Identity and Access Management (IAM)
Access the domain with the Amazon SageMaker console.

Create an IAM role

Creating an IAM role gives Amazon SageMaker permission to perform actions in other AWS services on your behalf. Creating a role here will grant 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
Amazon SageMaker can assume 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, bucket-name-3
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"
 S3 bucket with a Bucket Policy allowing access to SageMaker

See Object tagging See S3 bucket policies

Cancel **Create role**

▼ Network and Storage Section

VPC

Access the domain with a bookmarked URL. Access the domain with the Amazon SageMaker console.

Permission

Default execution role

SageMaker Domain requires permissions for its users to access other AWS services, such as Amazon SageMaker and Amazon S3. For a broad range of AWS services, you may attach the `AmazonSageMakerFullAccess` policy to the execution role. If you don't have a role with this policy, we can create one for you.

AmazonSageMaker-ExecutionRole-20231027T153174

Success! You created an IAM role.

AmazonSageMaker-ExecutionRole-20231027T153174

Create role using the role creation wizard

Space default execution role

SageMaker Domain requires permissions for its users to access other AWS services, such as Amazon SageMaker and Amazon S3. For a broad range of AWS services, you may attach the `AmazonSageMakerFullAccess` policy to the execution role. If you don't have a role with this policy, we can create one for you.

Please choose an execution role

Create role using the role creation wizard

If you leave space default execution role blank, you will not be able to use the shared spaces feature for collaboration. Once a role has been set for the space default execution role, it can never be disabled or unset. You can only change the default role from the domain settings page.

▼ Network and Storage Section

6. Select default VPC and click Next

Please choose an execution role

Create role using the role creation wizard

If you leave space default execution role blank, you will not be able to use the shared spaces feature for collaboration. Once a role has been set for the space default execution role, it can never be disabled or unset. You can only change the default role from the domain settings page.

▼ Network and Storage Section

To enable internet access, make sure that your VPC has a NAT gateway and your security group allows outbound connections.

Default vpc-0ee95eb60ca7f73708 (172.31.0.0/16)

Subnet

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

Security groups

These security groups will also be associated with the `RSudioServerPro` App.

Public Internet Only - The SageMaker domain will use default SageMaker internet access. Your VPC is used only for accessing the attached EFS storage

VPC Only - The SageMaker domain will use your VPC. Direct internet access is disabled. To enable internet access, make sure that your VPC has a NAT gateway and your security group allows outbound connections.

Encryption key - optional

SageMaker uses an AWS-managed CMK to encrypt your EFS and EBS file systems by default. To use a customer-managed CMK, enter its key ID or ARN. Learn more

No Custom Encryption

Cancel **Next**

7. Other settings in the subsequent steps can be left as it is, so keep clicking **Next two times followed by Submit on the third time**

8. Creating a Domain might take 5 to 10 minutes

Screenshot of the Amazon SageMaker console showing the Domains page. A domain named "Sagemaker-domain" is listed with status "InService". The "Create domain" button is visible.

9. Once the Domain is created we can create a Notebook Instance. In the same Sagemaker console, select Notebook Instances from Notebook

Screenshot of the Amazon SageMaker console showing the Notebook Instances page. The "Create notebook instance" button is highlighted.

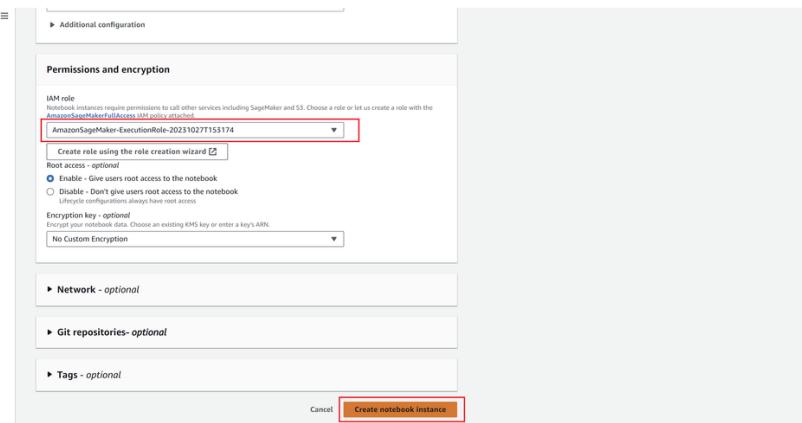
10. In the Notebook Instances page select Create Notebook Instance

Screenshot of the Amazon SageMaker console showing the Create notebook instance page. The "Notebook instance name" field is highlighted.

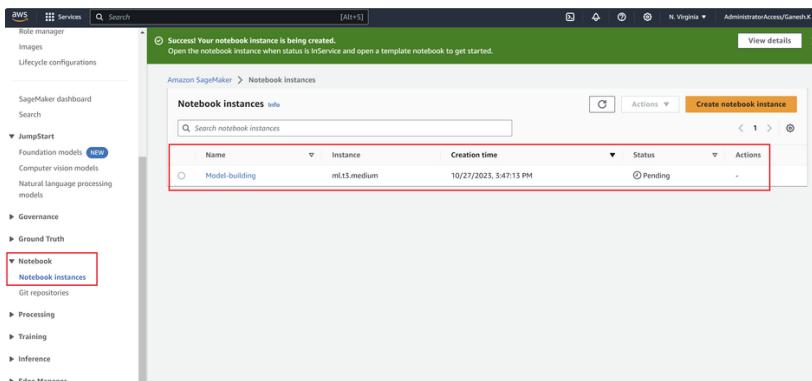
11. On the creation page, give a name for the notebook. Adhere to the AWS naming conventions

Screenshot of the Amazon SageMaker console showing the Create notebook instance settings. The "Notebook instance name" field contains "Model-building".

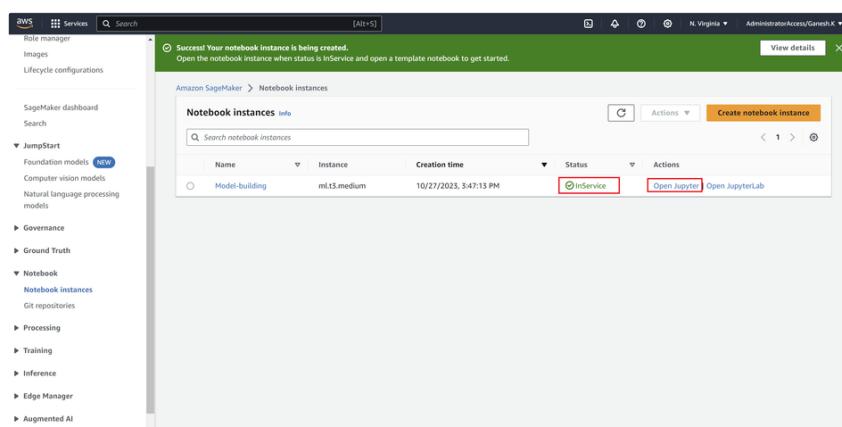
12. Under permission, select the role that has been created during **domain creation** in Step 5 and click **Create Notebook Instance**. Other settings can be left as it was



13. The instance takes a few minutes to create



14. After creation, select **Open Jupyter**



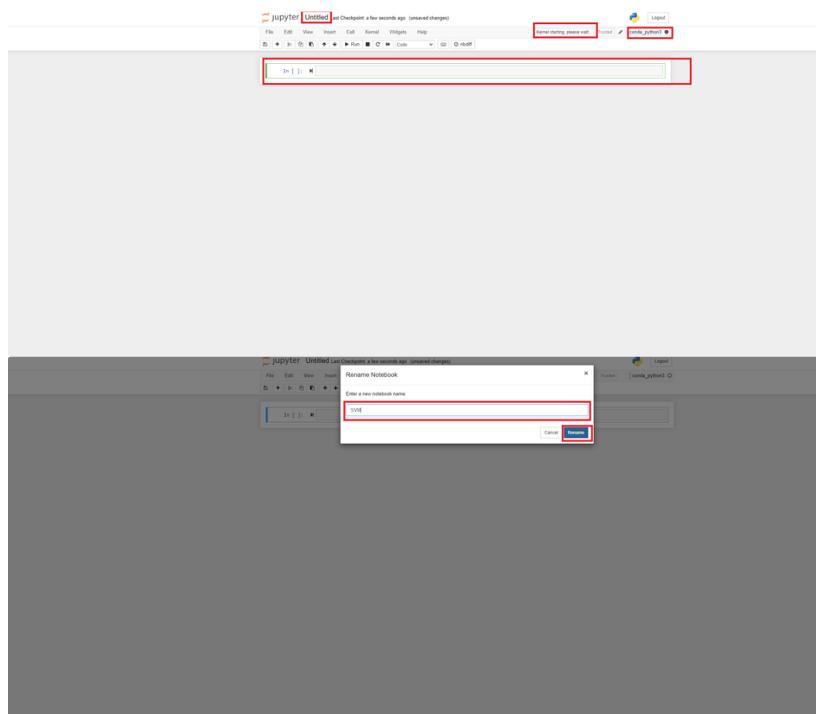
15. In the next window, the Jupyter Notebook opens, as it is in the local machine



16. Create a notebook to write the code. Select **New** and **Conda_python3** as shown below



17. A **new notebook file** opens in the new tab. Double-click the **untitled** and give a **new name** for the file



Step 3: Building Model in notebook instance

1. Build a model file for the data. In this project, the model is SVM, therefore building a model accordingly is given below. This script is designed to train an SVM classifier using SageMaker. It takes training data as input, specifies SVM parameters such as the kernel and 'C' hyperparameter, and saves the trained model for later use. This trained model can then be used for making predictions on new data.

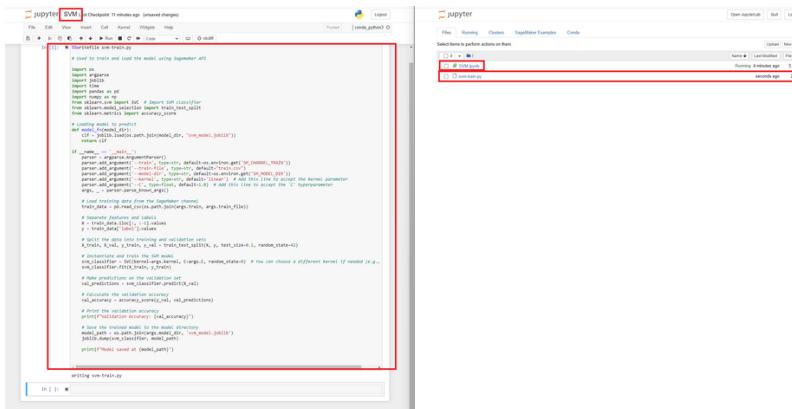
```
1 %%writefile svm-train.py
2
3 # Used to train and load the model using Sagemaker API
4
5 import os
6 import argparse
7 import joblib
8 import time
9 import pandas as pd
10 import numpy as np
11 from sklearn.svm import SVC # Import SVM classifier
12 from sklearn.model_selection import train_test_split
```

```

13 from sklearn.metrics import accuracy_score
14
15 # Loading model to predict
16 def model_fn(model_dir):
17     clf = joblib.load(os.path.join(model_dir, "svm_model.joblib"))
18     return clf
19
20 if __name__ == '__main__':
21     parser = argparse.ArgumentParser()
22     parser.add_argument('--train', type=str, default=os.environ.get('SM_CHANNEL_TRAIN'))
23     parser.add_argument('--train-file', type=str, default="train.csv")
24     parser.add_argument('--model-dir', type=str, default=os.environ.get('SM_MODEL_DIR'))
25     parser.add_argument('--kernel', type=str, default='linear') # Add this line to accept the kernel parameter
26     parser.add_argument('--C', type=float, default=1.0) # Add this line to accept the 'C' hyperparameter
27     args, _ = parser.parse_known_args()
28
29 # Load training data from the SageMaker channel
30 train_data = pd.read_csv(os.path.join(args.train, args.train_file))
31
32 # Separate features and labels
33 X = train_data.iloc[:, :-1].values
34 y = train_data['label'].values
35
36 # Split the data into training and validation sets
37 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, random_state=42)
38
39 # Instantiate and train the SVM model
40 svm_classifier = SVC(kernel=args.kernel, C=args.C, random_state=0) # You can choose a different kernel if needed (e.g., 'rbf')
41 svm_classifier.fit(X_train, y_train)
42
43 # Make predictions on the validation set
44 val_predictions = svm_classifier.predict(X_val)
45
46 # Calculate the validation accuracy
47 val_accuracy = accuracy_score(y_val, val_predictions)
48
49 # Print the validation accuracy
50 print(f"Validation Accuracy: {val_accuracy}")
51
52 # Save the trained model to the model directory
53 model_path = os.path.join(args.model_dir, 'svm_model.joblib')
54 joblib.dump(svm_classifier, model_path)
55
56 print(f"Model saved at {model_path}")

```

2. The above script writes a file **svm-train.py** in the notebook directory which will be used later. The **code file is on the left** and the **directory containing the code file and script file is shown on the right**



3. Import the necessary headers to be used and run the cell

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5 plt.rcParams["figure.figsize"] = [10, 5]
6 import seaborn as sns
7 from scipy import stats
8 from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
9 from sklearn.model_selection import train_test_split
10 import time
11
12 #Define IAM Role
13 import boto3
14 import sagemaker
15 from sagemaker.sklearn.estimator import SKLearn
16 from sagemaker.predictor import Predictor
17 from sagemaker.sklearn.model import SKLearnModel
18 from sagemaker.tuner import HyperparameterTuner, IntegerParameter, ContinuousParameter, CategoricalParameter

```

4. Define **role, region, path, and session** for the s3 bucket and sagemaker as well as define the file name

```

1 # Set SageMaker and S3 client variables
2 sess = sagemaker.Session()
3
4 s3_region = 'us-east-1'
5 s3_client = boto3.client("s3", region_name=s3_region)
6 sm_boto3 = boto3.client("sagemaker")
7
8 sagemaker_role = sagemaker.get_execution_role()
9
10 bucket_name = 'bucket--ganesh'
11 read_prefix = 'Project/RF-try'
12
13 input_file_name = "advertising.csv"
14 train_file_name = "train.csv"
15
16 input_file_path = f"s3://{{bucket_name}}/{{read_prefix}}/Data/"
17 train_file_path = f"s3://{{bucket_name}}/{{read_prefix}}/Data/"

```

5. Now **load the data** into the sagemaker and run the cell. It should display the data frame table that **confirms the steps are correct till now**.

```

1 # Load the data.
2 data = pd.read_csv(input_file_path + input_file_name)
3 data

```

jupyter SVM Last Checkpoint: 16 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda_python3 nbdf

```
# Make predictions on the validation set
val_predictions = svm_classifier.predict(X_val)

# Calculate the validation accuracy
val_accuracy = accuracy_score(y_val, val_predictions)

# Print the validation accuracy
print(f"Validation Accuracy: {val_accuracy}")

# Save the trained model to the model directory
model_path = os.path.join(args.model_dir, 'svm_model.joblib')
joblib.dump(svm_classifier, model_path)

print(f"Model saved at {model_path}")

Writing svm-train.py
```

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [10, 5]
import seaborn as sns
from scipy import stats
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import train_test_split
import time

#Define IAM Role
import boto3
import sagemaker
from sagemaker.sklearn.estimator import SKLearn
from sagemaker.predictor import Predictor
from sagemaker.sklearn.model import SKLearnModel
from sagemaker.tuner import HyperparameterTuner, IntegerParameter, ContinuousParameter, CategoricalParameter
```

sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml

In [3]:

```
# Set SageMaker and S3 client variables
sess = sagemaker.Session()

s3_region = 'us-east-1'
s3_client = boto3.client("s3", region_name=s3_region)
sm_boto3 = boto3.client("sagemaker")

sagemaker_role = sagemaker.get_execution_role()

bucket_name = 'bucket-ganesh'
read_prefix = 'Project/RF-try'

input_file_name = "advertising.csv"
train_file_name = "train.csv"

input_file_path = f"s3:///{bucket_name}/{read_prefix}/Data/"
train_file_path = f"s3:///{bucket_name}/{read_prefix}/Data/"
```

sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml

In [3]:

```
# Set SageMaker and S3 client variables
sess = sagemaker.Session()

s3_region = 'us-east-1'
s3_client = boto3.client("s3", region_name=s3_region)
sm_boto3 = boto3.client("sagemaker")

sagemaker_role = sagemaker.get_execution_role()

bucket_name = 'bucket-ganesh'
read_prefix = 'Project/RF-try'

input_file_name = "advertising.csv"
train_file_name = "train.csv"

input_file_path = f"s3:///{bucket_name}/{read_prefix}/Data/"
train_file_path = f"s3:///{bucket_name}/{read_prefix}/Data/"
```

sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml

In [4]:

```
# Load the data.
data = pd.read_csv(input_file_path + input_file_name)
data
```

Out[4]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	238.50	Organic bottom-line service-desk	Davitdon	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:38:18	0
...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00	1
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Dariene	1	Bosnia and Herzegovina	2016-04-22 02:07:01	1
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57	1
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54	0
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronnemouth	0	Brazil	2016-06-03 21:43:21	1

1000 rows x 10 columns

6. Now preprocess the data and split the data into test and train. Save the train file to load for training in the S3 bucket data folder where the data file is present.

```
1 # Timestamp conversion to seconds
2 data['Timestamp'] = (pd.to_datetime(data['Timestamp']) - pd.to_datetime(data['Timestamp'][0])).dt.total_seconds()
```

```

3
4 # Drop non-integer columns
5 data = data.drop(columns=['Ad Topic Line', 'Country', 'City'])
6
7 # Extract features and target variable
8 X = data.iloc[:, 1:-1].values.astype('float64')
9 y = data['Clicked on Ad'].values
10
11 # Pre-processing - normalization of data
12 X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
13
14 # Split the data into training and testing sets
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
16
17 # Write the training data to a file
18 trainX = pd.DataFrame(X_train)
19 trainX['label'] = y_train
20
21 trainX.to_csv(train_file_path + train_file_name, index=False, header=True)

```

7. Now both **data** and **model** are ready to train

Step 4: Training the Model and deploying it for prediction

1. Before training **verify the code file/model** working with the below code

```

1 ! python svm-train.py --model-dir ./ \
2                         --C 2.0 \
3                         --kernel 'sigmoid' ./ \
4                         --train {train_file_path} \
5

```

2. Need to see an **output model file** in the notebook directory and **validation accuracy** printed below the cell



jupyter SVM Last Checkpoint 27 minutes ago (unsaved changes) Log out

File Edit View Insert Cell Kernel Widgets Help Trusted conda_python3

Out[4]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned Stigeneration orcland	Wrightburgh	0	Tunisia	2016-03-27 08:53:11	0
1	88.23	31	68441.85	193.77	Monetary recall standardization	West Jodi	1	Nauru	2016-04-24 01:39:02	0
2	69.47	26	59785.94	238.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-20 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Temirtut	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00	1
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	Neu Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01	1
997	51.83	51	42416.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 02:57:57	1
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54	0
999	45.01	26	29875.80	178.35	Virtual Stigeneration emulation	Ronniesmouth	0	Brazil	2016-06-03 21:43:21	1

1000 rows x 10 columns

In [6]:

```
# Timestamp conversion to seconds
data['Timestamp'] = (pd.to_datetime(data['Timestamp']) - pd.to_datetime(data['Timestamp'][0])).dt.total_seconds()

# Drop non-integer columns
data = data.drop(columns=['Ad Topic Line', 'Country', 'City'])

# Extract features and target variable
X = data.iloc[:, 1:-1].values.astype('float64')
y = data['Clicked on Ad'].values

# Pre-processing - normalization of data
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Write the training data to a file
trainX = pd.DataFrame(X_train)
trainX['label'] = y_train

trainX.to_csv(train_file_path + train_file_name, index=False, header=True)
```

In [7]:

```
python svm-train.py --model-dir ./ \
--C 2.0 \
--kernel 'sigmoid' ./ \
--train {train_file_path} \
```

Validation Accuracy: 0.84
Model saved at ./svm_model.joblib

3. Now make the **Sagemaker train the model**. This takes some time, but the job can be seen in the sagemaker console.

```
1 # Define the SageMaker Estimator
2 estimator = SKLearn(entry_point="svm-train.py", # Your training script file
3                      role=sagemaker_role,      # SageMaker IAM Role
4                      instance_type="ml.m4.xlarge", # Choose an appropriate instance type
5                      sagemaker_session=sess,
6                      framework_version="1.2-1", # Your desired scikit-learn version
7                      base_job_name="svm-training"
8                  )
9
10 estimator.fit({'train': train_file_path})
```

4. In the Sagemaker console, select **training Jobs from the Training**. The job created in the notebook is visible here with the status

5. Once it's **successful**, the **status** in the sagemaker is changed to **completed** which means the model is trained and ready to deploy.

6. Now download the **artifacts, and hyperparameter** from the model and save it for later use.

```
1 estimator.latest_training_job.wait(logs="None")  
2 artifact = sm_boto3.describe_training_job(
```

```

3     TrainingJobName=estimator.latest_training_job.name
4 )["ModelArtifacts"]["S3ModelArtifacts"]
5
6 print("Model artifact persisted at " + artifact)

```

```

In [7]: python svm-train.py -model_dir ./ \
    -C 2.0 \
    -kernel 'sigmoid' \
    -train (train_file_path) \
Validation Accuracy: 0.84
Model saved at /svm_model.joblib

In [8]: # Define the SageMaker Estimator
estimator = SKLearn(entry_point="svm-train.py", # Your training script file
                    role=sagemakerRole, # SageMaker IAM Role
                    instance_type='ml.m4.xlarge', # Choose an appropriate instance type
                    framework_version='1.2-1', # Your desired scikit-learn version
                    base_job_name="svm-training"
                    )
estimator.fit((train_file_path))

In [9]: estimator.latest_training_job.wait(logs="None")
artifact = sm.Boto().describe_training_job(
    TrainingJobName=estimator.latest_training_job.name
)[("ModelArtifacts")["S3ModelArtifacts"]]
print("Model artifact persisted at " + artifact)

```

7. Now create the endpoint for inference

```

1 model = SKLearnModel(
2     model_data=artifact,
3     role=sagemaker_role,
4     entry_point="svm-train.py",
5     framework_version="1.2-1",
6 )
7
8 # Deploy the trained model to an endpoint
9 predictor = model.deploy(initial_instance_count=1, instance_type='ml.m5.large', endpoint_name='svm-endpoint')
10

```

8. Similar to the training job, even the endpoint can be seen in the sagemaker console in **Endpoints under Inference**

Name	ARN	Create time	Status	Last updated
svm-endpoint	arn:aws:sagemaker:us-east-1:977473325382:endpoint/svm-endpoint	2023-10-27 15:52:43	Creating	2023-10-27 15:52:43

```

In [9]: estimator.latest_training_job.wait(logs="None")
TrainingJobName=estimator.latest_training_job.name
print("Model artifact persisted at " + artifact)

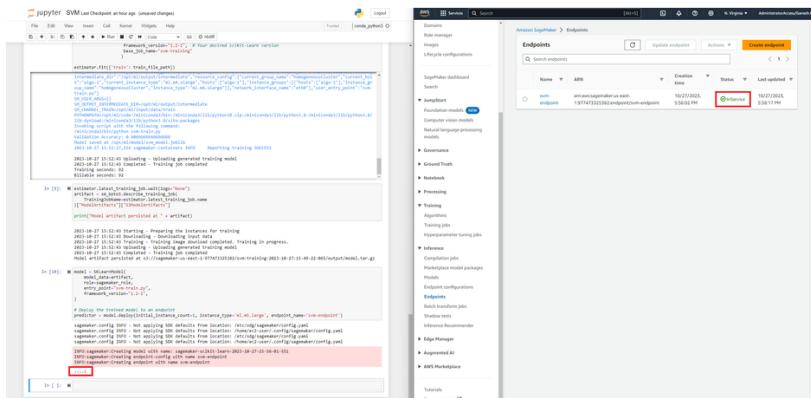
2023-10-27 15:52:43 Starting - Preparing the instances for training
2023-10-27 15:52:43 Downloading - Downloading input data
2023-10-27 15:52:43 Preparing - Preparing training instances completed. Training in progress.
2023-10-27 15:52:43 Uploading - Uploading generated training model
2023-10-27 15:52:43 Completed - Training job completed
Model artifact persisted at s3://sagemaker-us-east-1-977473325382/svm-training-2023-10-27-15-40-22-065/output/model.tar.gz

In [10]: model = SKLearnModel(
    model_data=artifact,
    role=sagemakerRole,
    entry_point="svm-train.py",
    framework_version="1.2-1",
    instance_type='ml.m5.large',
    predictor_type='SageMaker'
)
predictor = model.deploy(initial_instance_count=1, instance_type='ml.m5.large', endpoint_name='svm-endpoint')

2023-10-27 15:52:43 Starting - Preparing the instances for training
2023-10-27 15:52:43 Downloading - Downloading input data
2023-10-27 15:52:43 Preparing - Preparing training instances completed. Training in progress.
2023-10-27 15:52:43 Uploading - Uploading generated training model
2023-10-27 15:52:43 Completed - Training job completed
Model artifact persisted at s3://sagemaker-us-east-1-977473325382/svm-training-2023-10-27-15-40-22-065/output/model.tar.gz

```

9. Once the creation is done, the **status** changes to **Inservice**

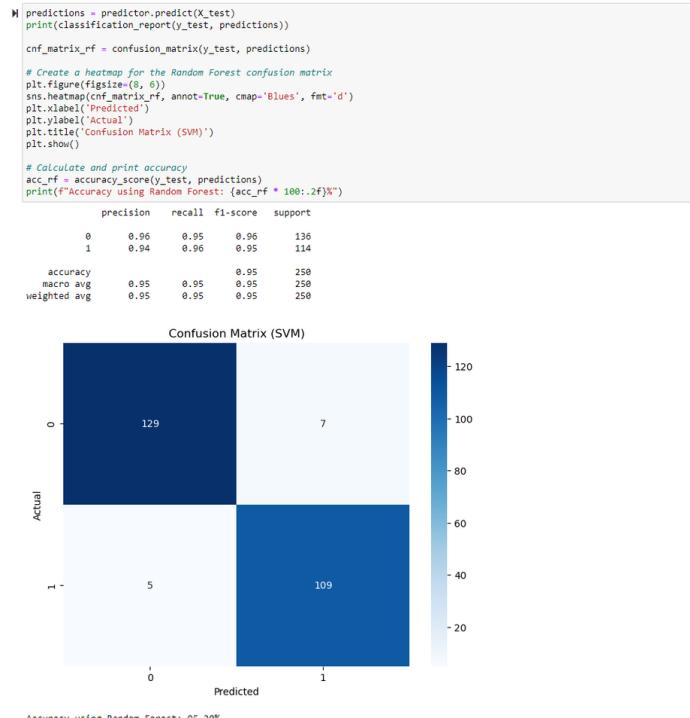


10. Now the **endpoint** can be used to predict the label for **test data**

```

1 predictions = predictor.predict(X_test)
2 print(classification_report(y_test, predictions))
3
4 cnf_matrix_rf = confusion_matrix(y_test, predictions)
5
6 # Create a heatmap for the Random Forest confusion matrix
7 plt.figure(figsize=(8, 6))
8 sns.heatmap(cnf_matrix_rf, annot=True, cmap='Blues', fmt='d')
9 plt.xlabel('Predicted')
10 plt.ylabel('Actual')
11 plt.title('Confusion Matrix (SVM)')
12 plt.show()
13
14 # Calculate and print accuracy
15 acc_rf = accuracy_score(y_test, predictions)
16 print(f"Accuracy using Random Forest: {acc_rf * 100:.2f}%")
17

```



Step 5: Hyperparameter Tuning and updating the prediction

1. Now **hyperparameter tuning** to get the best parameter for the current model. In this project for demo, only **two parameters are used that is 'C' and 'kernel'** for SVM.

```
1 # Define hyperparameter ranges and strategy
2 hyperparameter_ranges = {
3     'C': IntegerParameter(1, 3), # The regularization parameter
4     'kernel': CategoricalParameter(['linear', 'rbf', 'poly', 'sigmoid']), # The kernel type
5     #'gamma': ContinuousParameter(0.1, 10.0), # Gamma parameter (used in 'rbf', 'poly', and 'sigmoid' kernels)
6     #'degree': IntegerParameter(2, 5) # Degree parameter (used in 'poly' kernel)
7 }
8
9 num_C_values = 3 - 1 + 1 # 1 to 3 inclusive
10 num_kernel_values = len(['linear', 'rbf', 'poly', 'sigmoid'])
11 total_combinations = num_C_values * num_kernel_values # Total number of combinations
12
13
14 objective_metric_name = 'validation:accuracy'
15
16 # Define the metric definition
17 metric_definitions = [{Name: 'validation:accuracy', Regex: 'Validation Accuracy: ([0-9.]+)'}]
18
19
20 # Create a HyperparameterTuner
21 tuner = HyperparameterTuner(
22     estimator=estimator,
23     objective_metric_name=objective_metric_name,
24     hyperparameter_ranges=hyperparameter_ranges,
25     metric_definitions=metric_definitions,
26     strategy='Bayesian', # You can use other strategies like 'Bayesian' or 'Grid'
27     max_jobs=total_combinations, # Number of hyperparameter combinations to try
28     max_parallel_jobs=2, # Number of jobs to run in parallel
29     base_tuning_job_name='svm-hyperparameter-tuning',
30 )
31
32 # Start hyperparameter tuning job
33 tuner.fit({'train': train_file_path})
34
35 while True:
36     # Check the status of the tuning job
37     status = sm_boto3.describe_hyper_parameter_tuning_job(
38         HyperParameterTuningJobName=tuner.latest_tuning_job.job_name
39     )['HyperParameterTuningJobStatus']
40
41     if status == 'Completed':
42         print("Hyperparameter tuning job has completed.")
43         break
44     elif status == 'Failed' or status == 'Stopped':
45         print("Hyperparameter tuning job has failed or stopped.")
46         break
47
48     # Wait for a while before checking again
49     time.sleep(60) # You can adjust the polling interval
```

2. Even the **hyperparameter** job can be visible the same as the **training job**, actually, it's down to the training job option itself. Since a **parallel** job is two, it runs as a pair of jobs from the total jobs.

3. After completion

The screenshot shows two browser windows side-by-side. The left window is a Jupyter Notebook cell containing Python code for creating and monitoring a hyperparameter tuning job. The right window is the AWS SageMaker console, specifically the 'Hyperparameter tuning jobs' page, listing several completed tuning jobs.

Jupyter Notebook Code:

```

# Import libraries
import sagemaker
from sagemaker import get_execution_role
from sagemaker.hyperparameter_tuning import HyperparameterTuningJob

# Set up environment
role = get_execution_role()
s3_bucket = "s3:///"
prefix = "sagemaker-hptuning"

# Create hyperparameter tuning job
hp_tuning_job = HyperparameterTuningJob(
    role=role,
    base_job_name="hp-tuning-job",
    strategy="Bayesian",
    max_parallel_jobs=10,
    max_runtime_per_training_job_in_seconds=3600,
    max_total_run_time_in_hours=100,
    metric_definitions=[{"Name": "Validation Accuracy", "Regex": "Validation Accuracy: ([0-9.]+)"}],
    parameter_ranges={
        "n_estimators": {
            "minValue": 3,
            "maxValue": 5,
            "stepSize": 1
        },
        "max_depth": {
            "minValue": 1,
            "maxValue": 10,
            "stepSize": 1
        }
    },
    training_instances=10,
    validation_data={
        "InputMode": "File",
        "Location": s3_bucket + "train.csv"
    },
    validation_metric="Validation Accuracy",
    validation_regex="Validation Accuracy: ([0-9.]+)"
)
hp_tuning_job.create()

# Monitor hyperparameter tuning job
while True:
    status = hp_tuning_job.describe_hyper_parameter_tuning()
    print(f"Hyperparameter tuning job {status['HyperparameterTuningJobName']} is {status['Status']}")
```

AWS SageMaker Console - Hyperparameter tuning jobs:

Name	Status	Training duration (min)	Creation time	Duration
hp-tuning-job-23102023-014507	Completed	12 / 10	2023/10/20, 11:45:07	1 minutes
hp-tuning-job-23102023-014508	Completed	10 / 10	2023/10/20, 11:45:08	1 minutes
23102023-014509	Completed	10 / 10	2023/10/20, 11:45:09	1 minutes
hp-tuning-job-23102023-014510	Completed	10 / 10	2023/10/20, 11:45:10	1 minutes
23102023-014511	Completed	10 / 10	2023/10/20, 11:45:11	1 minutes
23102023-014512	Completed	10 / 10	2023/10/20, 11:45:12	1 minutes
23102023-014513	Completed	10 / 10	2023/10/20, 11:45:13	1 minutes
23102023-014514	Completed	10 / 10	2023/10/20, 11:45:14	1 minutes
23102023-014515	Completed	10 / 10	2023/10/20, 11:45:15	1 minutes
23102023-014516	Completed	10 / 10	2023/10/20, 11:45:16	1 minutes
23102023-014517	Completed	10 / 10	2023/10/20, 11:45:17	1 minutes
23102023-014518	Completed	10 / 10	2023/10/20, 11:45:18	1 minutes
23102023-014519	Completed	10 / 10	2023/10/20, 11:45:19	1 minutes
23102023-014520	Completed	10 / 10	2023/10/20, 11:45:20	1 minutes
23102023-014521	Completed	10 / 10	2023/10/20, 11:45:21	1 minutes

4. Also possible to print what parameters have been used and what accuracy each parameter setting has generated. Also the **Sagemaker's best pick**

```
1 # Retrieve the details of the completed hyperparameter tuning job
2 tuning_job_name = tuner.latest_tuning_job.job_name
3 training_job_summaries = sm_boto3.list_training_jobs_for_hyper_parameter_tuning_job(
4     HyperParameterTuningJobName=tuning_job_name
5 )['TrainingJobSummaries']
6
7 # Get the best training job from SageMaker
8 tuning_job_description = sm_boto3.describe_hyper_parameter_tuning_job(
9     HyperParameterTuningJobName=tuning_job_name
10 )
11
12 # Print hyperparameters and validation accuracy for each training job
13 results = []
14 for job_summary in training_job_summaries:
15     job_name = job_summary['TrainingJobName']
16     hyperparameters = job_summary['TunedHyperParameters']
17     validation_metric = job_summary['FinalHyperParameterTuningJobObjectiveMetric']
18
19     print(f"Job Name: {job_name}")
20     print(f"Hyperparameters: {hyperparameters}")
21     print(f"Validation Metric ({objective_metric_name}): {validation_metric}")
22     print("")
23
24     results.append({
25         'TrainingJobName': job_name,
26         'Hyperparameters': hyperparameters,
27         'ValidationMetric': validation_metric
28     })
29
30 # Retrieve the best training job based on SageMaker-selected objective metric
31 best_training_job_name = tuner.best_training_job()
32 best_training_job_description = sm_boto3.describe_training_job(TrainingJobName=best_training_job_name)
33 best_hyperparameters = best_training_job_description['HyperParameters']
34 best_validation_metric = best_training_job_description['FinalMetricDataList'][0] # Assuming you want the first metric
35
36 print(f"Training Job Name: {best_training_job_name}")
37 print("Hyperparameters:")
```

```
38 for key, value in best_hyperparameters.items():
39     print(f" {key}: {value}")
40 print(f"Validation Metric ({objective_metric_name}): {best_validation_metric['Value']:.4f}")
41
42 # Convert the results to a DataFrame for further analysis if needed
43 results_df = pd.DataFrame(results)
```

jupyter SVM Last Checkpoint an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted conda_python3

```
})
# Retrieve the best training job based on SageMaker-selected objective metric
best_training_job_name = tuner.best_training_job()
best_training_job_description = sm_boto3.describe_training_job(TrainingJobName=best_training_job_name)
best_hyperparameters = best_training_job_description['HyperParameters']
best_validation_metric = best_training_job_description['FinalMetricDataList'][0] # Assuming you want the first metric

print("Training Job Name: (%s)" % (best_training_job_name))
print("Hyperparameters:")
for key, value in best_hyperparameters.items():
    print("%s (%s): (%s)" % (key, value))
print("Validation Metric (%s): (%s)" % (objective_metric_name, (best_validation_metric['Value']):.4f))

Job Name: svm-hyperparameter-t-231027-1607-012-050f8a0
Hyperparameters: {'C': '2', 'kernel': 'linear'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9066666960716248}

Job Name: svm-hyperparameter-t-231027-1607-011-33ffff9eb
Hyperparameters: {'C': '1', 'kernel': 'rbf'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9066666960716248}

Job Name: svm-hyperparameter-t-231027-1607-010-910-cdeadfc20
Hyperparameters: {'C': '3', 'kernel': 'rbf'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9066666960716248}

Job Name: svm-hyperparameter-t-231027-1607-009-99983c19
Hyperparameters: {'C': '1', 'kernel': 'linear'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9066666960716248}

Job Name: svm-hyperparameter-t-231027-1607-008-5d2fa941
Hyperparameters: {'C': '3', 'kernel': 'sigmoid'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9200000156893005}

Job Name: svm-hyperparameter-t-231027-1607-007-4cb058d0
Hyperparameters: {'C': '3', 'kernel': 'linear'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9066666960716248}

Job Name: svm-hyperparameter-t-231027-1607-006-fdf9340f
Hyperparameters: {'C': '2', 'kernel': 'sigmoid'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9200000156893005}

Job Name: svm-hyperparameter-t-231027-1607-005-1e3f7fd6
Hyperparameters: {'C': '2', 'kernel': 'rbf'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9066666960716248}

Job Name: svm-hyperparameter-t-231027-1607-004-aed7073
Hyperparameters: {'C': '2', 'kernel': 'poly'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9066666960716248}

Job Name: svm-hyperparameter-t-231027-1607-003-35605d41
Hyperparameters: {'C': '1', 'kernel': 'sigmoid'}
Validation Metric (validation:accuracy): {'MetricName': 'validation:accuracy', 'Value': 0.9200000156893005}

Training Job Name: svm-hyperparameter-t-231027-1607-008-5d2fa941
Hyperparameters:
  C: 3
  _tuning_objective_metric: validation:accuracy
  kernel: "sigmoid"
  sgemaker_container_log_level: 2
  sgemaker_estimator_class_name: "sklearn"
  sgemaker_estimator_name: "SgemakerEstimator"
  sgemaker_job_name: "svm-training-2023-10-27-16-07-57-550"
  sgemaker_program: "svm-train.py"
  sgemaker_region: "us-east-1"
  sgemaker_submit_directory: "s3://sagemaker-us-east-1-977473325382/svm-training-2023-10-27-16-07-57-550/source/sourcedir.tgz"
Validation Metric (validation:accuracy): 0.92000
```

5. Now fit the best model create a new endpoint and delete the old endpoint

The screenshot displays the AWS SageMaker console interface. On the left, the 'Endpoints' section shows a table with one row for the endpoint 'sagemaker-endpoint'. The status is listed as 'Creating' with a timestamp of '14/07/2023 13:14:00'. On the right, the 'Logs' tab shows the command-line output for creating the endpoint:

```
aws sagemaker create-endpoint --endpoint-name sagemaker-endpoint --model-name sagemaker-model --instance-type ml.t2.medium --output text --query 'EndpointStatus'| jq .
```

The log output indicates the endpoint is being created and provides a tracking link: <https://us-east-1.console.aws.amazon.com/sagemaker/endpoint?region=us-east-1&tab=logs&logGroupFilter=sagemaker-endpoint&logStreamFilter=&start=2023-07-14T13:14:00Z&end=2023-07-14T13:15:00Z>.

The screenshot shows two parallel windows. On the left is a Jupyter notebook cell containing Python code for training an SVM model using scikit-learn. The code includes imports, parameter definitions, and a call to `predictor.predict(X_test)`. A red box highlights the command `predictor = Model.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge', endpoint_name='tuned-svm-endpoint')`. Below the code, the terminal output shows the deployment process, including the creation of a Lambda function and its execution.

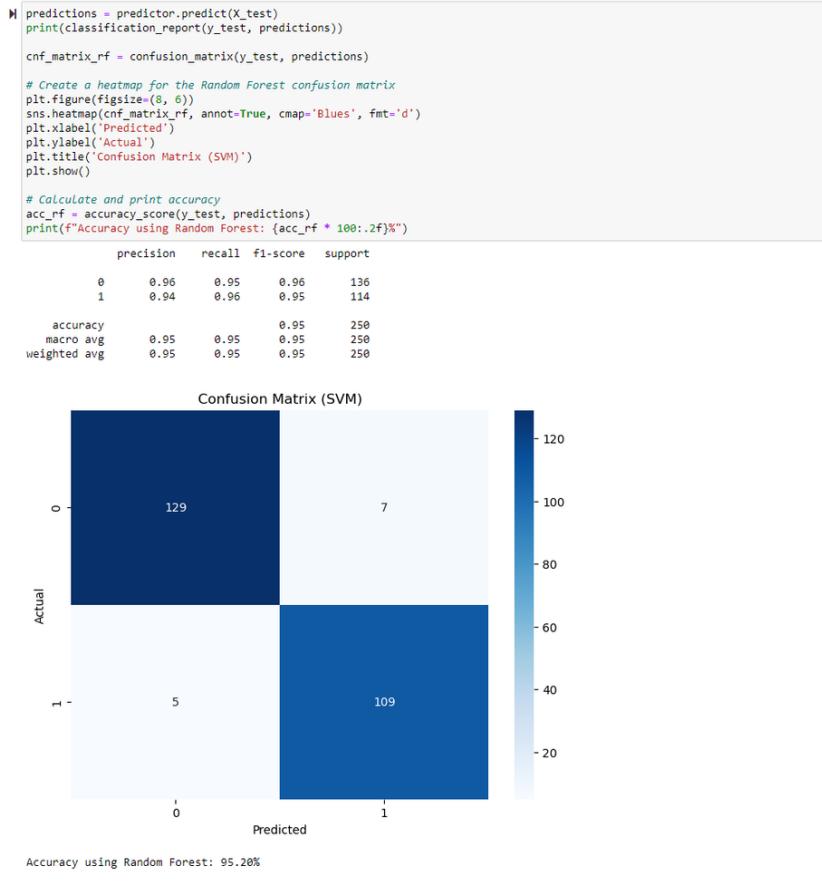
On the right is the 'Amazon SageMaker > Endpoints' section of the AWS Management Console. It lists an endpoint named 'tuned-svm-endpoint' with ARN 'arn:aws:sagemaker:us-east-1:917071212382:endpoint/tuned-svm-endpoint'. The status is 'Inference' with a green icon. A red box highlights this endpoint entry.

6. Now run the endpoint for inference on test data

```

1 predictions = predictor.predict(X_test)
2 print(classification_report(y_test, predictions))
3
4 cnf_matrix_rf = confusion_matrix(y_test, predictions)
5
6 # Create a heatmap for the Random Forest confusion matrix
7 plt.figure(figsize=(8, 6))
8 sns.heatmap(cnf_matrix_rf, annot=True, cmap='Blues', fmt='d')
9 plt.xlabel('Predicted')
10 plt.ylabel('Actual')
11 plt.title('Confusion Matrix (Random Forest)')
12 plt.show()
13
14 # Calculate and print accuracy
15 acc_rf = accuracy_score(y_test, predictions)
16 print(f"Accuracy using Random Forest: {acc_rf * 100:.2f}%")

```



Although immediate improvements may not be readily apparent, as we scale up our data and leverage more intricate models, this particular step takes on greater significance. Consequently, we've undertaken the task of building, training, and optimizing a Support Vector Machine (SVM) model utilizing the advertising dataset. Moreover, we've established an endpoint that serves as the final inference point for our test data. This process stands as a pivotal phase in our data analysis and modeling endeavors, executed within the AWS infrastructure through Amazon SageMaker, a platform that streamlines machine learning development and deployment.