

STEPS TO BUILD-TRAIN-DEPLOY MODEL IN AWS SAGEMAKER:

STEP 1: CREATE IAM ROLE AND NOTEBOOK INSTANCE

Create IAM role, choose Create a new role, and then choose Create role.

1. This IAM role automatically gets permissions to access any S3 bucket that has sagemaker in the name. It gets these permissions through the AmazonSageMakerFullAccess policy, which SageMaker attaches to the role.

Add permissions [Info](#)

Permissions policies (1) [Info](#)
The type of role that you selected requires the following policy.

Policy name ✎	Type
AmazonSageMakerFullAccess	AWS managed

► **Set permissions boundary - optional**

Cancel Previous **Next**

2. Go to Amazon SageMaker console.
3. Select “Notebook instances” and click “Create notebook instance.”

Amazon SageMaker > Notebook instances

Notebook instances [Info](#) **Actions** [Create notebook instance](#)

Name	Instance	Creation time	Status	Actions
There are currently no resources.				

4. Fill in details:

Notebook instance name: Choose a name.

— Notebook Instance type: Use “ml.t2.medium” (or “ml.t3.medium” if unavailable).

— Platform Identifier: Choose a platform type for OS and JupyterLab version.

5. Leave other fields with default values.

6. Complete the creation process based on preferences.

Amazon SageMaker > Notebook instances > Create notebook instance

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

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

Notebook instance type

Platform identifier [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.

Custom IAM role ARN

[Create role using the role creation wizard](#)

Root access - optional

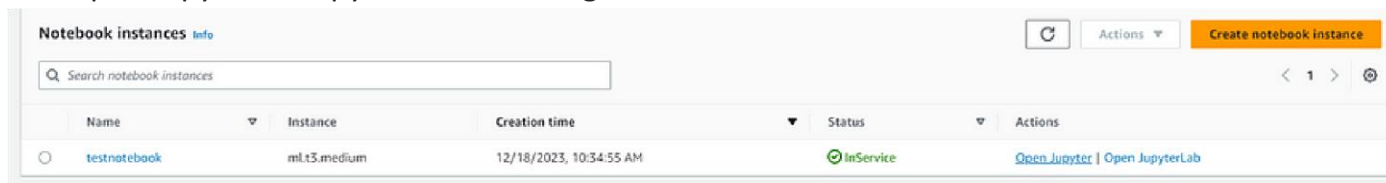
☒ Enable - Give users root access to the notebook

☐ Disable - Don't give users root access to the notebook

Lifecycle configurations always have root access

STEP 2: CREATE A JUPYTER NOTEBOOK

1. Open Jupyter or JupyterLab according to the interface needed.



2. Go to File menu->Choose New-> Notebook.

3. Select Kernel as 'conda_python3'

STEP 3: DOWNLOAD, EXPLORE AND TRANSFORM DATA

1. Load Adult Census dataset using SHAP

(The dataset: <https://archive.ics.uci.edu/dataset/2/adult>)

```
import shap
X, y = shap.datasets.adult()
X_display, y_display = shap.datasets.adult(display=True)
feature_names = list(X.columns)
feature_names
```

2. Split the dataset into train ,test and validation datasets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
X_train_display = X_display.loc[X_train.index]

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
X_train_display = X_display.loc[X_train.index]
X_val_display = X_display.loc[X_val.index]
```

3. Aligning dataset by concatenating the numeric features with the true labels.

```
import pandas as pd
train = pd.concat([pd.Series(y_train, index=X_train.index,
                             name='Income>50K', dtype=int), X_train], axis=1)
validation = pd.concat([pd.Series(y_val, index=X_val.index,
                                   name='Income>50K', dtype=int), X_val], axis=1)
test = pd.concat([pd.Series(y_test, index=X_test.index,
                             name='Income>50K', dtype=int), X_test], axis=1)
```

4. Check if the dataset is split properly.

```
train
test
validate
```

5. Convert train, test and validation datasets to CSV

We need to convert these datasets to match the input file format for XGBoost algorithm which we will be using for model training and deployment and upload dataset to S3

```
train.to_csv('train.csv', index=False, header=False)
validation.to_csv('validation.csv', index=False, header=False)
```

```
import sagemaker, boto3, os
bucket = sagemaker.Session().default_bucket()
prefix = "demo-sagemaker-xgboost-adult-income-prediction"

boto3.Session().resource('s3').Bucket(bucket).Object(
    os.path.join(prefix, 'data/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(
    os.path.join(prefix, 'data/validation.csv')).upload_file('validation.csv')
```

STEP 4: TRAIN THE MODEL

1. Retrieve region and role.

```
import sagemaker

region = sagemaker.Session().boto_region_name
print("AWS Region: {}".format(region))

role = sagemaker.get_execution_role()
print("RoleArn: {}".format(role))
```

2. Create an XGBoost estimator.

XGBoost, short for Extreme Gradient Boosting, is a machine learning algorithm that falls under the category of gradient-boosted decision trees.

```
from sagemaker.debugger import Rule, ProfilerRule, rule_configs
from sagemaker.session import TrainingInput

s3_output_location='s3://{}/{}/{}'.format(bucket, prefix, 'xgboost_model')

container=sagemaker.image_uris.retrieve("xgboost", region, "1.2-1")
print(container)

xgb_model=sagemaker.estimator.Estimator(
    image_uri=container,
    role=role,
    instance_count=1,
    instance_type='ml.m4.xlarge',
    volume_size=5,
    output_path=s3_output_location,
    sagemaker_session=sagemaker.Session(),
    rules=[
        Rule.sagemaker(rule_configs.create_xgboost_report()),
        ProfilerRule.sagemaker(rule_configs.ProfilerReport())
    ]
)
```

3. Set hyperparameters for XGBoost algorithm.

```
xgb_model.set_hyperparameters(  
    max_depth = 5,  
    eta = 0.2,  
    gamma = 4,  
    min_child_weight = 6,  
    subsample = 0.7,  
    objective = "binary:logistic",  
    num_round = 1000  
)
```

4. Using TrainingInput class to configure data input flow.

```
from sagemaker.session import TrainingInput  
  
train_input = TrainingInput(  
    "s3://{}/{}/{}/{}".format(bucket, prefix, "data/train.csv"), content_type="csv"  
)  
validation_input = TrainingInput(  
    "s3://{}/{}/{}/{}".format(bucket, prefix, "data/validation.csv"), content_type="csv"
```

5. Train the model

```
xgb_model.fit({"train": train_input, "validation": validation_input}, wait=True)
```

6. Check the location of debugging reports generated.

```
rule_output_path = xgb_model.output_path + "/" + xgb_model.latest_training_job.job_name  
! aws s3 ls {rule_output_path} --recursive
```

7. Download the Debugger XGBoost reports and generate link.


```
! aws s3 cp {rule_output_path} ./ --recursive

from IPython.display import FileLink, FileLinks
display("Click link below to view the XGBoost Training report", FileLink("Create
```

XGBoost Training Report by SageMaker Debugger

The SageMaker Debugger `CreateXgboostReport` built-in rule auto-generates this report. This report provides a summary of the XGBoost model training evaluation results, insights of the model performance, and interactive graphs.

Legal disclaimer: In this report, plots and recommendations are provided for informational purposes only and are not definitive. You are responsible for making your own independent assessment of the information.] For more information, see the following documentation:

- [Amazon SageMaker Developer Guide](#)

If you want to use the notebook that generated this report, you need to install the following libraries:

- [SageMaker Debugger Client Library Github](#)
- [The Bokeh Python Visualization Tool](#)


```
In [3]: # Parameters
path = "/opt/ml/processing/input/tensors"
plot_step = 995
s3_path = "s3://sagemaker-ap-south-1-567757221582/demo-sagemaker-xgboost-adult-income-prediction/xgboost_model/sagemaker-xgboost-2023-12-18-05-37-21-694/debug-output"
```

The following parameters are the default values auto-generated by the `CreateXgboostReport` built-in rule.

- `path (str)` - The local path where Debugger has saved output tensors in the training container.
- `plot_step (int)` - The step for which the rule has created the training report.
- `s3_path (str)` - The S3 bucket URI where Debugger has saved the output tensors.

Table of Contents

- [Distribution of True Labels of the Dataset](#)
- [Loss vs Step Graph](#)
- [Feature Importance](#)
- [Confusion Matrix](#)
- [Evaluation of the Confusion Matrix](#)
- [Accuracy Rate of Each Diagonal Element over Iteration](#)
- [Receiver Operating Characteristic Curve](#)
- [Distribution of Residuals at Last Saved Step](#)

 BokehJS 2.2.3 successfully loaded.

8. Get profiling reports for instance resource utilization, system bottleneck etc.

```
profiler_report_name = [rule["RuleConfigurationName"]
                        for rule in xgb_model.latest_training_job.rule_job_summaries
                        if "Profiler" in rule["RuleConfigurationName"]][0]

profiler_report_name
display("Click link below to view the profiler report", FileLink(profiler_report_name))
```

SageMaker Debugger Profiling Report

SageMaker Debugger auto generated this report. You can generate similar reports on all supported training jobs. The report provides summary of training job, system resource usage statistics, framework metrics, rules summary, and detailed analysis from each rule. The graphs and tables are interactive.

Legal disclaimer: This report and any recommendations are provided for informational purposes only and are not definitive. You are responsible for making your own independent assessment of the information.

Training job summary

The following table gives a summary about the training job. The table includes information about when the training job started and ended, how much time initialization, training loop and finalization took. Your training job started on 12/18/2023 at 05:39:28 and ran for 91 seconds. Your training job started on 12/18/2023 at 05:39:28 and ran for 91 seconds. No step information was profiled from your training job. The time spent on initialization and finalization cannot be computed.

#		Job Statistics
0	Start time	05:39:28 12/18/2023
1	End time	05:40:59 12/18/2023
2	Job duration	91 seconds

STEP 5: DEPLOY THE MODEL

```
import sagemaker
from sagemaker.serializers import CSVSerializer
xgb_predictor=xgb_model.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium',
    serializer=CSVSerializer()
)
```

STEP 6: EVALUATE THE MODEL

```
import numpy as np
def predict(data, rows=1000):
    split_array = np.array_split(data, int(data.shape[0] / float(rows) + 1))
    predictions = ''
    for array in split_array:
        predictions = ','.join([predictions, xgb_predictor.predict(array).decode()])
    return np.fromstring(predictions[1:], sep=',')
```



```
import sklearn

cutoff=0.5
print(sklearn.metrics.confusion_matrix(test.iloc[:, 0], np.where(predictions > c
print(sklearn.metrics.classification_report(test.iloc[:, 0], np.where(prediction
```

```
[[4670  356]
 [ 480 1007]]
```

	precision	recall	f1-score	support
0	0.91	0.93	0.92	5026
1	0.74	0.68	0.71	1487
accuracy			0.87	6513
macro avg	0.82	0.80	0.81	6513
weighted avg	0.87	0.87	0.87	6513