Lab 3.7 - Student Notebook

Overview

This lab is a continuation of the guided labs in Module 3.

In this lab, you will create a hyperparameter tuning job to tune the model that you created previously. You will then compare the metrics of the two models.

Introduction to the business scenario

You work for a healthcare provider, and want to improve the detection of abnormalities in orthopedic patients.

You are tasked with solving this problem by using machine learning (ML). You have access to a dataset that contains six biomechanical features and a target of *normal* or *abnormal*. You can use this dataset to train an ML model to predict if a patient will have an abnormality.

About this dataset

This biomedical dataset was built by Dr. Henrique da Mota during a medical residence period in the Group of Applied Research in Orthopaedics (GARO) of the Centre Médico-Chirurgical de Réadaptation des Massues, Lyon, France. The data has been organized in two different, but related, classification tasks.

The first task consists in classifying patients as belonging to one of three categories:

- *Normal* (100 patients)
- *Disk Hernia* (60 patients)
- *Spondylolisthesis* (150 patients)

For the second task, the categories *Disk Hernia* and *Spondylolisthesis* were merged into a single category that is labeled as *abnormal*. Thus, the second task consists in classifying patients as belonging to one of two categories: *Normal* (100 patients) or *Abnormal* (210 patients).

Attribute information

Each patient is represented in the dataset by six biomechanical attributes that are derived from the shape and orientation of the pelvis and lumbar spine (in this order):

- Pelvic incidence
- Pelvic tilt
- Lumbar lordosis angle
- Sacral slope
- Pelvic radius
- Grade of spondylolisthesis

The following convention is used for the class labels:

- DH (Disk Hernia)
- Spondylolisthesis (SL)
- Normal (NO)
- Abnormal (AB)

For more information about this dataset, see the Vertebral Column dataset webpage.

Dataset attributions

This dataset was obtained from: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). Irvine, CA: University of California, School of Information and Computer Science.

Lab setup

Because this solution is split across several labs in the module, you run the following cells so that you can load the data and train the model to be deployed.

Note: The setup can take up to 5 minutes to complete.

Importing the data, and training, testing and validating the model

By running the following cells, the data will be imported, and the model will be trained, tested and validated and ready for use.

Note: The following cells represent the key steps in the previous labs.

In order to tune the model it must be ready, then you can tweak the mdoel with hyperparameters later in step 2.

In [1]:
```
bucket='c134412a3409743l8225410t1w851725395798-labbucket-5nsifr2uehlu'
```

In [2]:
```
import time
start = time.time()
```

```
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

import os
import boto3
import sagemaker
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split

from sklearn.metrics import roc_auc_score, roc_curve, auc, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/s
agemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-
user/.config/sagemaker/config.yaml
Matplotlib is building the font cache; this may take a moment.
```

Note: The following cell takes approximately **10** minutes to complete. Observe the code and how it processes, this will help you to better understand what is going on in the background. Keep in mind that this cell completes all the steps you did in previous labs in this module including:

- Importing the data
- Loading the data into a dataframe
- Splitting the data into training, test and validation datasets
- Uploading the split datasets to S3
- Training, testing and validating the model with the datasets

In [3]:
```
%%time

def plot_roc(test_labels, target_predicted_binary):
    TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).
    # Sensitivity, hit rate, recall, or true positive rate
    Sensitivity  = float(TP)/(TP+FN)*100
    # Specificity or true negative rate
    Specificity  = float(TN)/(TN+FP)*100
    # Precision or positive predictive value
    Precision = float(TP)/(TP+FP)*100
    # Negative predictive value
    NPV = float(TN)/(TN+FN)*100
    # Fall out or false positive rate
    FPR = float(FP)/(FP+TN)*100
    # False negative rate
    FNR = float(FN)/(TP+FN)*100
    # False discovery rate
    FDR = float(FP)/(TP+FP)*100
    # Overall accuracy
    ACC = float(TP+TN)/(TP+FP+FN+TN)*100
```

```
    print(f"Sensitivity or TPR: {Sensitivity}%")
    print(f"Specificity or TNR: {Specificity}%")
    print(f"Precision: {Precision}%")
    print(f"Negative Predictive Value: {NPV}%")
    print( f"False Positive Rate: {FPR}%")
    print(f"False Negative Rate: {FNR}%")
    print(f"False Discovery Rate: {FDR}%" )
    print(f"Accuracy: {ACC}%")

    test_labels = test.iloc[:,0];
    print("Validation AUC", roc_auc_score(test_labels, target_predicted_bina

    fpr, tpr, thresholds = roc_curve(test_labels, target_predicted_binary)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")

    # create the axis of thresholds (scores)
    ax2 = plt.gca().twinx()
    ax2.plot(fpr, thresholds, markeredgecolor='r',linestyle='dashed', color=
    ax2.set_ylabel('Threshold',color='r')
    ax2.set_ylim([thresholds[-1],thresholds[0]])
    ax2.set_xlim([fpr[0],fpr[-1]])

    print(plt.figure())

def plot_confusion_matrix(test_labels, target_predicted):
    matrix = confusion_matrix(test_labels, target_predicted)
    df_confusion = pd.DataFrame(matrix)
    colormap = sns.color_palette("BrBG", 10)
    sns.heatmap(df_confusion, annot=True, fmt='.2f', cbar=None, cmap=colorma
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.ylabel("True Class")
    plt.xlabel("Predicted Class")
    plt.show()


f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vert
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])

class_mapper = {b'Abnormal':1,b'Normal':0}
df['class']=df['class'].replace(class_mapper)
```

```
cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]

train, test_and_validate = train_test_split(df, test_size=0.2, random_state=
test, validate = train_test_split(test_and_validate, test_size=0.5, random_s

prefix='lab3'

train_file='vertebral_train.csv'
test_file='vertebral_test.csv'
validate_file='vertebral_validate.csv'

s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False )
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)

upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)

container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')

hyperparams={"num_round":"42",
             "eval_metric": "auc",
             "objective": "binary:logistic",
             "silent" : 1}

s3_output_location="s3://{}/{}/output/".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                        sagemaker.get_execution_role(),
                                        instance_count=1,
                                        instance_type='ml.m5.2xlarge',
                                        output_path=s3_output_location,
                                         hyperparameters=hyperparams,
                                         sagemaker_session=sagemaker.Session(

train_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/train/".format(bucket,prefix,train_file),
    content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/validate/".format(bucket,prefix,validate_file),
    content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}

xgb_model.fit(inputs=data_channels, logs=False)

batch_X = test.iloc[:,1:];

batch_X_file='batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)
```

```
batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = xgb_model.transformer(instance_count=1,
                                        instance_type='ml.m5.2xlarge',
                                        strategy='MultiRecord',
                                        assemble_with='Line',
                                        output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                          data_type='S3Prefix',
                          content_type='text/csv',
                          split_type='Line')
xgb_transformer.wait(logs=False)
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2024-11-03-21-45-30-920
2024-11-03 21:45:33 Starting - Starting the training job..
2024-11-03 21:45:47 Starting - Preparing the instances for training....
2024-11-03 21:46:12 Downloading - Downloading input data..
2024-11-03 21:46:27 Downloading - Downloading the training image......
2024-11-03 21:47:02 Training - Training image download completed. Training in progress.....
2024-11-03 21:47:28 Uploading - Uploading generated training model
2024-11-03 21:47:36 Completed - Training job completed
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2024-11-03-21-47-37-542
```

```
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2024-11-03-21-47-38-184
```

```
.........................[2024-11-03:21:52:00:INFO] No GPUs detected (norm
al if no gpus installed)
[2024-11-03:21:52:00:INFO] No GPUs detected (normal if no gpus installed)
[2024-11-03:21:52:00:INFO] nginx config:
worker_processes auto;
daemon off;
pid /tmp/nginx.pid;
error_log  /dev/stderr;
worker_rlimit_nofile 4096;
events {
  worker_connections 2048;
}
http {
  include /etc/nginx/mime.types;
  default_type application/octet-stream;
  access_log /dev/stdout combined;
  upstream gunicorn {
    server unix:/tmp/gunicorn.sock;
  }
  server {
    listen 8080 deferred;
    client_max_body_size 0;
    keepalive_timeout 3;
    location ~ ^/(ping|invocations|execution-parameters) {
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header Host $http_host;
      proxy_redirect off;
      proxy_read_timeout 60s;
      proxy_pass http://gunicorn;
    }
    location / {
      return 404 "{}";
    }
  }
}
[2024-11-03 21:52:00 +0000] [27] [INFO] Starting gunicorn 19.10.0
[2024-11-03 21:52:00 +0000] [27] [INFO] Listening at: unix:/tmp/gunicorn.so
ck (27)
[2024-11-03 21:52:00 +0000] [27] [INFO] Using worker: gevent
[2024-11-03 21:52:00 +0000] [38] [INFO] Booting worker with pid: 38
[2024-11-03 21:52:00 +0000] [39] [INFO] Booting worker with pid: 39
[2024-11-03 21:52:00 +0000] [40] [INFO] Booting worker with pid: 40
[2024-11-03 21:52:00 +0000] [41] [INFO] Booting worker with pid: 41
[2024-11-03 21:52:00 +0000] [56] [INFO] Booting worker with pid: 56
[2024-11-03 21:52:00 +0000] [57] [INFO] Booting worker with pid: 57
[2024-11-03 21:52:00 +0000] [58] [INFO] Booting worker with pid: 58
[2024-11-03 21:52:00 +0000] [73] [INFO] Booting worker with pid: 73
[2024-11-03:21:52:05:INFO] No GPUs detected (normal if no gpus installed)
169.254.255.130 - - [03/Nov/2024:21:52:05 +0000] "GET /ping HTTP/1.1" 200 0
"-" "Go-http-client/1.1"
[2024-11-03:21:52:05:INFO] No GPUs detected (normal if no gpus installed)
169.254.255.130 - - [03/Nov/2024:21:52:05 +0000] "GET /execution-parameters
HTTP/1.1" 200 84 "-" "Go-http-client/1.1"
[2024-11-03:21:52:05:INFO] No GPUs detected (normal if no gpus installed)
[2024-11-03:21:52:05:INFO] Determined delimiter of CSV input is ','
169.254.255.130 - - [03/Nov/2024:21:52:05 +0000] "POST /invocations HTTP/1.
```

```
1" 200 598 "-" "Go-http-client/1.1"

2024-11-03T21:52:05.157:[sagemaker logs]: MaxConcurrentTransforms=8, MaxPay
loadInMB=6, BatchStrategy=MULTI_RECORD
!
CPU times: user 1.59 s, sys: 137 ms, total: 1.72 s
Wall time: 7min 6s
```

Step 1: Getting model statistics

Before you tune the model, re-familiarize yourself with the current model's metrics.

The setup performed a batch prediction, so you must read in the results from Amazon Simple Storage Service (Amazon S3).

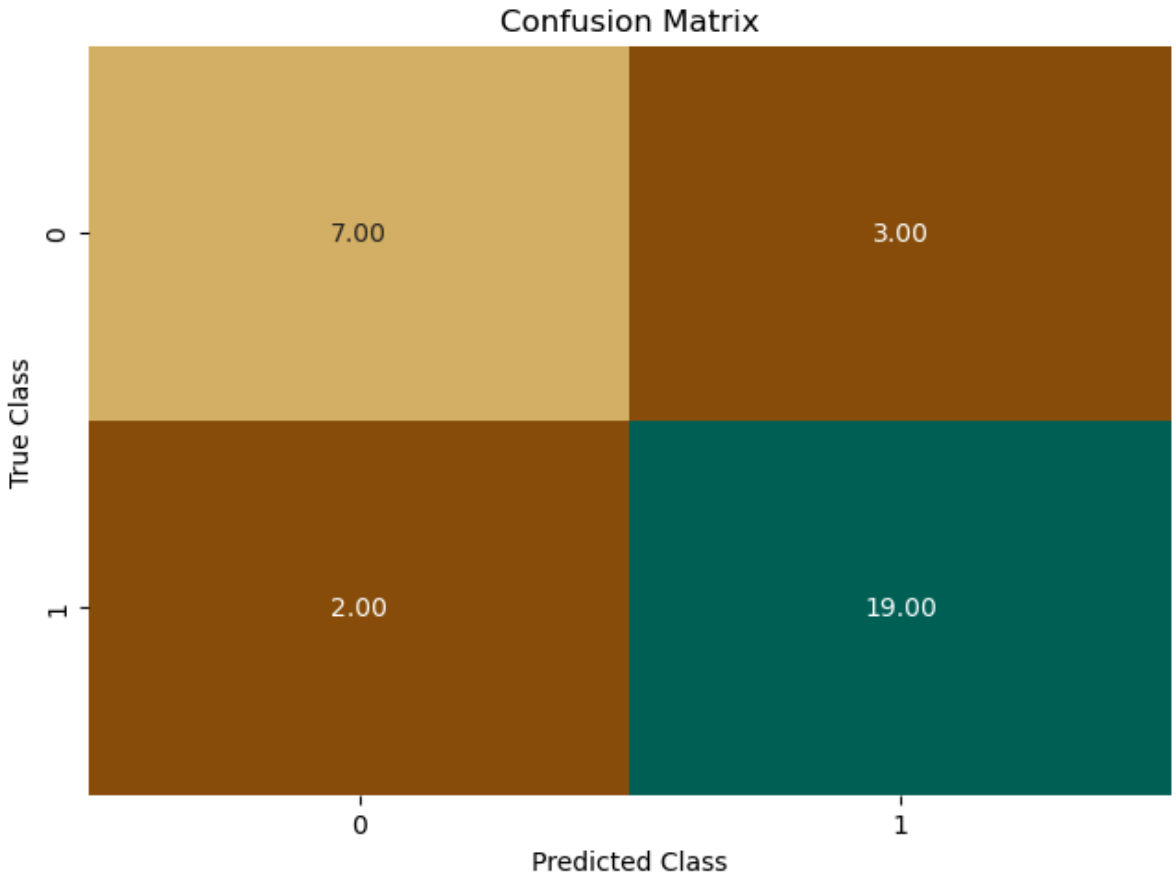
In [4]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batc
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),names=['class'

def binary_convert(x):
    threshold = 0.5
    if x > threshold:
        return 1
    else:
        return 0

target_predicted_binary = target_predicted['class'].apply(binary_convert)
test_labels = test.iloc[:,0]
```

Plot the confusion matrix and the receiver operating characteristic (ROC) curve for the original model.

In [5]:
```
plot_confusion_matrix(test_labels, target_predicted_binary)
```



In [20]:
```
#plot_roc(test_labels, target_predicted_binary)

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

def plot_roc(test_labels, target_predicted_probs):
    # Calculate the ROC curve
    fpr, tpr, thresholds = roc_curve(test_labels, target_predicted_probs)

    # Calculate the AUC (Area Under the Curve)
    roc_auc = auc(fpr, tpr)

    # Plotting the ROC curve
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'
    plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc='lower right')
    plt.grid()
    plt.show()
```

This plot gives you a starting point. Make a note of the *Validation area under the curve (AUC)*. You will use it later to check your tuned model to see if it's better.

Step 2: Creating a hyperparameter tuning job

A hyperparameter tuning job can take several hours to complete, depending on the value ranges that you provide. To simplify this task, the parameters used in this step are a subset of the recommended ranges. They were tuned to give good results in this lab, without taking multiple hours to complete.

For more information about the parameters to tune for XGBoost, see Tune an XGBoost Model in the AWS Documentation.

Because this next cell can take approximately **45** minutes to complete, go ahead and run the cell. You will examine what's happening, and why these hyperparameter ranges were chosen.

In [21]:
```
%%time
from sagemaker.tuner import IntegerParameter, CategoricalParameter, Continuc

xgb = sagemaker.estimator.Estimator(container,
                                    role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make sure you have
                                    instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(b
                                    sagemaker_session=sagemaker.Session())


xgb.set_hyperparameters(eval_metric='error@.40',
                        objective='binary:logistic',
                        num_round=42)

hyperparameter_ranges = {'alpha': ContinuousParameter(0, 100),
                         'min_child_weight': ContinuousParameter(1, 5),
                         'subsample': ContinuousParameter(0.5, 1),
                         'eta': ContinuousParameter(0.1, 0.3),
                         'num_round': IntegerParameter(1,50)
                         }

objective_metric_name = 'validation:error'
objective_type = 'Minimize'

tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or above depending
                            max_parallel_jobs=1,
                            objective_type=objective_type,
                            early_stopping_type='Auto')
```

```
tuner.fit(inputs=data_channels, include_cls_metadata=False)
tuner.wait()
```

```
WARNING:sagemaker.estimator:No finished training job found associated with
this estimator. Please make sure this estimator is only used for building w
orkflow config
WARNING:sagemaker.estimator:No finished training job found associated with
this estimator. Please make sure this estimator is only used for building w
orkflow config
INFO:sagemaker:Creating hyperparameter tuning job with name: sagemaker-xgbo
ost-241103-2220
.............................................
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
File <timed exec>:33

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/sagemaker/workflow/pipeline_context.py:346, in runnable_by_pipeline.<locals>.wrapper(*args, **kwargs)
    342         return context
    344     return _StepArguments(retrieve_caller_name(self_instance), run_func, *args, **kwargs)
--> 346 return run_func(*args, **kwargs)

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/sagemaker/tuner.py:1044, in HyperparameterTuner.fit(self, inputs, job_name, include_cls_metadata, estimator_kwargs, wait, **kwargs)
   1041     self._fit_with_estimator_dict(inputs, job_name, include_cls_metadata, estimator_kwargs)
   1043 if wait:
-> 1044     self.latest_tuning_job.wait()

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/sagemaker/tuner.py:2353, in _TuningJob.wait(self)
   2351 def wait(self):
   2352     """Placeholder docstring."""
-> 2353     self.sagemaker_session.wait_for_tuning_job(self.name)

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/sagemaker/session.py:5364, in Session.wait_for_tuning_job(self, job, poll)
   5350 def wait_for_tuning_job(self, job, poll=5):
   5351     """Wait for an Amazon SageMaker hyperparameter tuning job to complete.
   5352
   5353     Args:
   (...)
   5362         exceptions.UnexpectedStatusException: If the hyperparameter tuning job fails.
   5363     """
-> 5364     desc = _wait_until(lambda: _tuning_job_status(self.sagemaker_client, job), poll)
   5365     _check_job_status(job, desc, "HyperParameterTuningJobStatus")
   5366     return desc

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/sagemaker/session.py:8358, in _wait_until(callable_fn, poll)
   8356 try:
   8357     elapsed_time += poll
-> 8358     time.sleep(poll)
   8359     result = callable_fn()
   8360 except botocore.exceptions.ClientError as err:
   8361     # For initial 5 mins we accept/pass AccessDeniedException.
   8362     # The reason is to await tag propagation to avoid false AccessDenied claims for an
   8363     # access policy based on resource tags, The caveat here is for true AccessDenied
   8364     # cases the routine will fail after 5 mins
```

```
KeyboardInterrupt:
```

First, you will create the model that you want to tune.

```
xgb = sagemaker.estimator.Estimator(container,

role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make
sure you have limit set for these instances

instance_type='ml.m4.xlarge',

output_path='s3://{}/{}/output'.format(bucket, prefix),

sagemaker_session=sagemaker.Session())

xgb.set_hyperparameters(eval_metric='[error@.40]',
                        objective='binary:logistic',
                        num_round=42)
```

Notice that the *eval_metric* of the model was changed to *error@.40*, with a goal of minimizing that value.

error is the binary classification error rate. It's calculated as *#(wrong cases)/#(all cases)*. For predictions, the evaluation will consider the instances that have a prediction value larger than 0.4 to be positive instances, and the others as negative instances.

Next, you must specify the hyperparameters that you want to tune, in addition to the ranges that you must select for each parameter.

The hyperparameters that have the largest effect on XGBoost objective metrics are:

- alpha
- min_child_weight
- subsample
- eta
- num_round

The recommended tuning ranges can be found in the AWS Documentation at Tune an XGBoost Model.

For this lab, you will use a *subset* of values. These values were obtained by running the tuning job with the full range, then minimizing the range so that you can use fewer iterations to get better performance. Though this practice isn't strictly realistic, it prevents you from waiting several hours in this lab for the tuning job to complete.

```
    hyperparameter_ranges = {'alpha': ContinuousParameter(0,
    100),
                             'min_child_weight':
    ContinuousParameter(1, 5),
                             'subsample':
    ContinuousParameter(0.5, 1),
                             'eta': ContinuousParameter(0.1,
    0.3),
                             'num_round': IntegerParameter(1,50)
                             }
```

You must specify how you are rating the model. You could use several different objective metrics, a subset of which applies to a binary classifcation problem. Because the evaluation metric is **error**, you set the objective to *error*.

```
    objective_metric_name = 'validation:error'
    objective_type = 'Minimize'
```

Finally, you run the tuning job.

```
    tuner = HyperparameterTuner(xgb,
                                objective_metric_name,
                                hyperparameter_ranges,
                                max_jobs=10, # Set this to 10 or
    above depending upon budget & available time.
                                max_parallel_jobs=1,
                                objective_type=objective_type,
                                early_stopping_type='Auto')

    tuner.fit(inputs=data_channels, include_cls_metadata=False)
    tuner.wait()
```

Wait until the training job is finished. It might take up to **45** minutes. While you are waiting, observe the job status in the console, as described in the following instructions.

To monitor hyperparameter optimization jobs:

1. In the AWS Management Console, on the **Services** menu, choose **Amazon SageMaker**.
2. Choose **Training > Hyperparameter tuning jobs**.
3. You can check the status of each hyperparameter tuning job, its objective metric value, and its logs.

After the training job is finished, check the job and make sure that it completed successfully.

In [10]:
```
boto3.client('sagemaker').describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuner.latest_tuning_job.job_name)['HyperPara
```

Out[10]: 'Completed'

Step 3: Investigating the tuning job results

Now that the job is complete, there should be 10 completed jobs. One of the jobs should be marked as the best.

You can examine the metrics by getting *HyperparameterTuningJobAnalytics* and loading that data into a pandas DataFrame.

In [11]:
```
from pprint import pprint
from sagemaker.analytics import HyperparameterTuningJobAnalytics

tuner_analytics = HyperparameterTuningJobAnalytics(tuner.latest_tuning_job.r

df_tuning_job_analytics = tuner_analytics.dataframe()

# Sort the tuning job analytics by the final metrics value
df_tuning_job_analytics.sort_values(
    by=['FinalObjectiveValue'],
    inplace=True,
    ascending=False if tuner.objective_type == "Maximize" else True)

# Show detailed analytics for the top 20 models
df_tuning_job_analytics.head(20)
```

Out[11]:

	alpha	eta	min_child_weight	num_round	subsample	TrainingJobName	Traini
0	0.248525	0.145309	1.027660	15.0	0.829974	sagemaker-xgboost-241103-2157-010-285537e2	
1	0.000000	0.138341	1.000000	31.0	0.500000	sagemaker-xgboost-241103-2157-009-e5d0e547	
3	6.488520	0.292151	1.000000	18.0	0.691709	sagemaker-xgboost-241103-2157-007-3a1cf6b7	
2	9.570012	0.270619	2.019028	27.0	0.675733	sagemaker-xgboost-241103-2157-008-27e13c09	
4	2.098603	0.300000	5.000000	33.0	1.000000	sagemaker-xgboost-241103-2157-006-96854056	
6	24.284118	0.182020	1.227030	44.0	0.761065	sagemaker-xgboost-241103-2157-004-586dc701	
7	19.127396	0.114827	1.106674	21.0	0.605035	sagemaker-xgboost-241103-2157-003-574b6821	
9	29.936026	0.266307	2.514112	36.0	0.991369	sagemaker-xgboost-241103-2157-001-91f921db	
5	55.139907	0.270625	3.706694	35.0	0.677826	sagemaker-xgboost-241103-2157-005-d0aaf1fb	
8	85.855490	0.100717	2.668204	24.0	0.941681	sagemaker-xgboost-241103-2157-002-7387158b	

You should be able to see the hyperparameters that were used for each job, along with the score. You could use those parameters and create a model, or you can get the best model from the hyperparameter tuning job.

In [12]:
```
attached_tuner = HyperparameterTuner.attach(tuner.latest_tuning_job.name, sa
best_training_job = attached_tuner.best_training_job()
```

Now, you must attach to the best training job and create the model.

In [13]:
```
from sagemaker.estimator import Estimator
algo_estimator = Estimator.attach(best_training_job)

best_algo_model = algo_estimator.create_model(env={'SAGEMAKER_DEFAULT_INVOCA
```

```
2024-11-03 22:07:35 Starting - Found matching resource for reuse
2024-11-03 22:07:35 Downloading - Downloading the training image
2024-11-03 22:07:35 Training - Training image download completed. Training
in progress.
2024-11-03 22:07:35 Uploading - Uploading generated training model
2024-11-03 22:07:35 Completed - Resource retained for reuse
```

Then, you can use the transform method to perform a batch prediction by using your testing data. Remember that the testing data is data that the model has never seen before.

In [14]:
```
%%time
batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = best_algo_model.transformer(instance_count=1,
                                              instance_type='ml.m4.xlarge',
                                              strategy='MultiRecord',
                                              assemble_with='Line',
                                              output_path=batch_output)


xgb_transformer.transform(data=batch_input,
                          data_type='S3Prefix',
                          content_type='text/csv',
                          split_type='Line')
xgb_transformer.wait(logs=False)
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2024-11-03-22-07
-57-762
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2024-11-
03-22-07-58-295
```

```
.....................................[2024-11-03:22:14:19:INFO] No GPUs de
tected (normal if no gpus installed)
[2024-11-03:22:14:19:INFO] No GPUs detected (normal if no gpus installed)
[2024-11-03:22:14:19:INFO] nginx config:
worker_processes auto;
daemon off;
pid /tmp/nginx.pid;
error_log  /dev/stderr;
worker_rlimit_nofile 4096;
events {
  worker_connections 2048;
}
http {
  include /etc/nginx/mime.types;
  default_type application/octet-stream;
  access_log /dev/stdout combined;
  upstream gunicorn {
    server unix:/tmp/gunicorn.sock;
  }
  server {
    listen 8080 deferred;
    client_max_body_size 0;
    keepalive_timeout 3;
    location ~ ^/(ping|invocations|execution-parameters) {
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header Host $http_host;
      proxy_redirect off;
      proxy_read_timeout 60s;
      proxy_pass http://gunicorn;
    }
    location / {
      return 404 "{}";
    }
  }
}
[2024-11-03 22:14:19 +0000] [19] [INFO] Starting gunicorn 19.10.0
[2024-11-03 22:14:19 +0000] [19] [INFO] Listening at: unix:/tmp/gunicorn.so
ck (19)
[2024-11-03 22:14:19 +0000] [19] [INFO] Using worker: gevent
[2024-11-03 22:14:19 +0000] [26] [INFO] Booting worker with pid: 26
[2024-11-03 22:14:19 +0000] [27] [INFO] Booting worker with pid: 27
[2024-11-03 22:14:19 +0000] [28] [INFO] Booting worker with pid: 28
[2024-11-03 22:14:19 +0000] [29] [INFO] Booting worker with pid: 29
[2024-11-03:22:14:25:INFO] No GPUs detected (normal if no gpus installed)
169.254.255.130 - - [03/Nov/2024:22:14:25 +0000] "GET /ping HTTP/1.1" 200 0
"-" "Go-http-client/1.1"
169.254.255.130 - - [03/Nov/2024:22:14:25 +0000] "GET /execution-parameters
HTTP/1.1" 200 84 "-" "Go-http-client/1.1"
[2024-11-03:22:14:25:INFO] No GPUs detected (normal if no gpus installed)
[2024-11-03:22:14:25:INFO] Determined delimiter of CSV input is ','
169.254.255.130 - - [03/Nov/2024:22:14:25 +0000] "POST /invocations HTTP/1.
1" 200 588 "-" "Go-http-client/1.1"

2024-11-03T22:14:25.239:[sagemaker logs]: MaxConcurrentTransforms=4, MaxPay
loadInMB=6, BatchStrategy=MULTI_RECORD
!
```

```
CPU times: user 828 ms, sys: 43 ms, total: 871 ms
Wall time: 6min 59s
```

Get the predicted target and the test labels of the model.

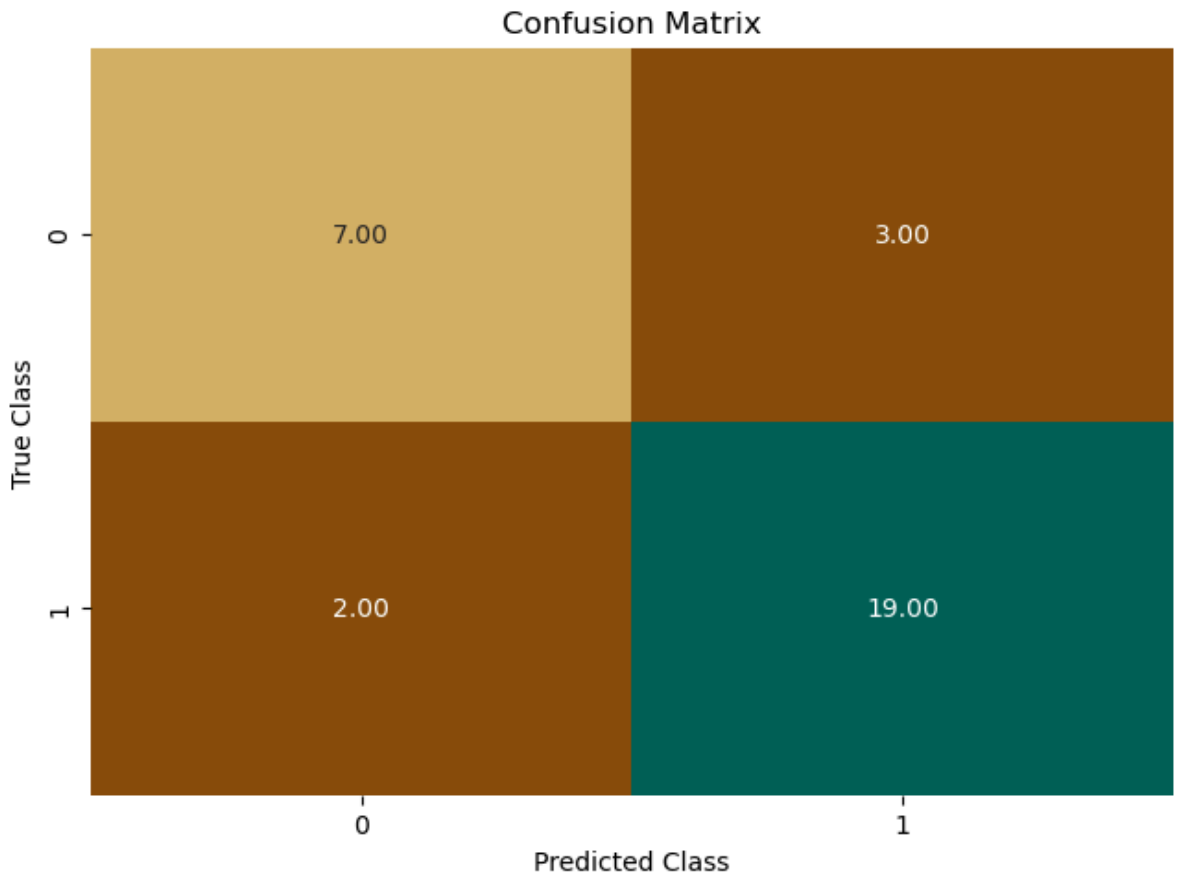
In [15]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batc
best_target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),names=['c

def binary_convert(x):
    threshold = 0.5
    if x > threshold:
        return 1
    else:
        return 0

best_target_predicted_binary = best_target_predicted['class'].apply(binary_c
test_labels = test.iloc[:,0]
```

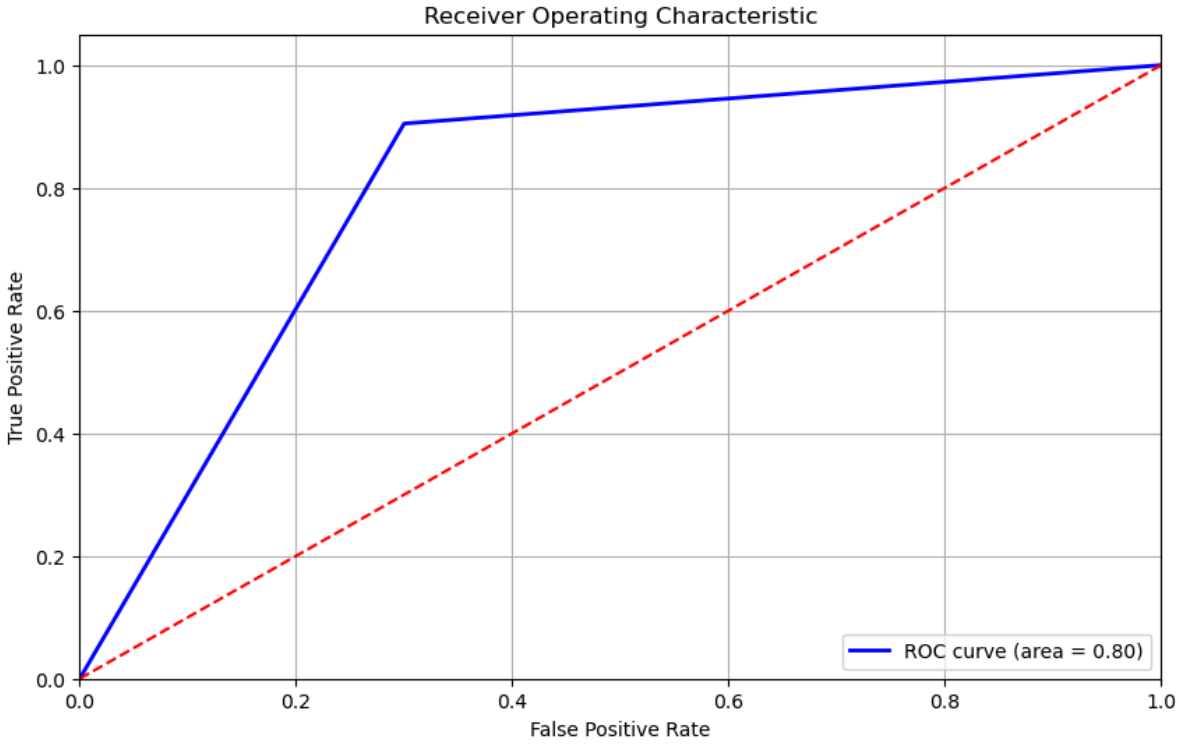
Plot a confusion matrix for your `best_target_predicted` and `test_labels`.

In [16]:
```
plot_confusion_matrix(test_labels, best_target_predicted_binary)
```



Plot the ROC chart.

In [25]:
```
plot_roc(test_labels, best_target_predicted_binary)
```



Question: How do these results differ from the original? Are these results better or worse?

You might not always see an improvement. There are a few reasons for this result:

- The model might already be good from the initial pass (what counts as *good* is subjective).
- You don't have a large amount of data to train with.
- You are using a *subset* of the hyperparameter tuning ranges to save time in this lab.

Increasing the hyperparameter ranges (as recommended by the documentation) and running more than 30 jobs will typically improve the model. However, this process will take 2-3 hours to complete.

Congratulations!

You have completed this lab, and you can now end the lab by following the lab guide instructions.