Sagemaker lab notes

#Importing the modules

```
import warnings, requests, zipfile, io

warnings.simplefilter('ignore')

import pandas as pd

from scipy.io import arff

import boto3
```

#Importing data

```
f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'

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)
```

#Step 1: Exploring the data

```
#First, use shape to examine the number of rows and columns.

df.shape

(310, 7)

#Next, get a list of the columns.

df.columns

Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class'],
      dtype='object')
```

#Step 2: Preparing the data

#Moving the target column position. XGBoost requires the training data to be in a single file. The file must have the target value be the first column.

#Get the target column and move it to the first position.

cols = df.columns.tolist()

cols = cols[-1:] + cols[:-1]

df = df[cols]

#You should see that the class is now the first column.

df.columns

```
Index(['class', 'pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis'],
      dtype='object')
```

#Splitting the dataset into two datasets. You will use the train_test_split function from the scikit-learn #library, which is a free machine learning library for Python. It has many #algorithms and useful #functions, such as the one you will use.

from sklearn.model_selection import train_test_split

train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42, stratify=df['class'])

#Next, split the test_and_validate dataset into two equal parts.

test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])

#Examine the three datasets.

print(train.shape)

print(test.shape)

print(validate.shape)

(248, 7)

(31, 7)

(31, 7)

#Now, check the distribution of the classes.

print(train['class'].value_counts())

print(test['class'].value_counts())

print(validate['class'].value_counts())

class

1   168

0   80

Name: count, dtype: int64

class

1   21

0   10

Name: count, dtype: int64

class

1   21

0   10

Name: count, dtype: int64

#Uploading the data to Amazon S3

#XGboost will load the data for training from Amazon Simple Storage Service (Amazon S3). Thus, you must write the data to a comma-separated values (CSV) file, and then upload the file to Amazon S3.

#Start by setting up some variables to the S3 bucket, then create a function to upload the CSV file to Amazon S3. You can reuse this function.

bucket='mys3bucket07282023'

prefix='lab'

train_file='vertebral_train.csv'

test_file='vertebral_test.csv'

validate_file='vertebral_validate.csv'

import os

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)).put(Body=csv_buffer.getvalue())


#Use the function that you created to upload the three datasets.

upload_s3_csv(train_file, 'train', train)

upload_s3_csv(test_file, 'test', test)

upload_s3_csv(validate_file, 'validate', validate)
```

#Step 3: Training the model

```
#Now that the data in Amazon S3, you can train a model.

#The first step is to get the XGBoost container URI.

import boto3

from sagemaker.image_uris import retrieve

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


#Next, you must set some hyperparameters for the model.

hyperparams={"num_round":"42",

        "eval_metric": "auc",

        "objective": "binary:logistic"}


#Use the estimator function to set up the model. Here are a few parameters of interest:

#instance_count - This defines how many instances will be used for training. You will use one instance.

#instance_type - This defines the instance type for training. In this case, it's ml.m4.xlarge.

import sagemaker

s3_output_location="s3://{}/{}/output/".format(bucket,prefix)

xgb_model=sagemaker.estimator.Estimator(container,

                        sagemaker.get_execution_role(),
```

```
                    instance_count=1,
                    instance_type='ml.m4.xlarge',
                    output_path=s3_output_location,
                     hyperparameters=hyperparams,
                     sagemaker_session=sagemaker.Session())
```

#The estimator needs channels to feed data into the model. For training, the train_channel and validate_channel will be used.

```
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}
```

```
#Running fit will train the model.
xgb_model.fit(inputs=data_channels, logs=False)
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2023-07-29-05-00-51-194
2023-07-29 05:00:51 Starting - Starting the training job........
2023-07-29 05:01:38 Starting - Preparing the instances for training.................
2023-07-29 05:03:17 Downloading - Downloading input data.....
2023-07-29 05:03:47 Training - Downloading the training image........
2023-07-29 05:04:33 Training - Training image download completed. Training in progress.....
2023-07-29 05:04:58 Uploading - Uploading generated training model..
2023-07-29 05:05:09 Completed - Training job completed
```

#Step 4: Performing a batch transform

#start by turning your data into a CSV file that the transformer object can take as input. This time, you will use iloc to get all the rows, and all columns except the first column.

```
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.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()


s3 = boto3.client('s3')

obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.csv.out'))

target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),names=['class'])


INFO:sagemaker:Creating model with name: sagemaker-xgboost-2023-07-29-05-40-37-386

INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2023-07-29-05-40-38-001

..................................
```

#Step 5: Exploring the results

#Build a function to convert the positive at probability into binary (0 or 1):

```
def binary_convert(x):
    threshold = 0.3
    if x > threshold:
        return 1
    else:
        return 0
target_predicted_binary = target_predicted['class'].apply(binary_convert)
print(target_predicted_binary.head(5))
0    1
1    1
2    1
3    1
4    1
```

#Step 6: Creating a confusion matrix

#To create a confusion matrix, we need both the target values from test data and the predicted value.

Get the targets from the test DataFrame.

```
test_labels = test.iloc[:,0]
test_labels.head()
136    1
230    0
134    1
130    1
47     1
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(test_labels, target_predicted_binary)
df_confusion = pd.DataFrame(matrix, index=['Nnormal','Abnormal'],columns=['Normal','Abnormal'])
```

df_confusion

	Normal	Abnormal
Nnormal	7	3
Abnormal	2	19

#Step 7: Calculating performance statistics

#To start, extract the values from the confusion matrix cells into variables.

```
from sklearn.metrics import roc_auc_score, roc_curve, auc

TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).ravel()

print(f"True Negative (TN) : {TN}")

print(f"False Positive (FP): {FP}")

print(f"False Negative (FN): {FN}")

print(f"True Positive (TP) : {TP}")
```

True Negative (TN) : 7

False Positive (FP): 3

False Negative (FN): 2

True Positive (TP) : 19

```
Sensitivity  = float(TP)/(TP+FN)*100

print(f"Sensitivity or TPR: {Sensitivity}%")
```

Sensitivity or TPR: 90.47619047619048%

```
Specificity  = float(TN)/(TN+FP)*100

print(f"Specificity or TNR: {Specificity}%")
```

pecificity or TNR: 70.0%

```
Precision = float(TP)/(TP+FP)*100

print(f"Precision: {Precision}%")
```

Precision: 86.36363636363636%