Problem: Predicting Airplane Delays

The goals of this notebook are:

- Process and create a dataset from downloaded .zip files
- Perform exploratory data analysis (EDA)
- Establish a baseline model
- Move from a simple model to an ensemble model
- Perform hyperparameter optimization
- Check feature importance

Introduction to business scenario

You work for a travel booking website that wants to improve the customer experience for flights that were delayed. The company wants to create a feature to let customers know if the flight will be delayed because of weather when they book a flight to or from the busiest airports for domestic travel in the US.

You are tasked with solving part of this problem by using machine learning (ML) to identify whether the flight will be delayed because of weather. You have been given access to the a dataset about the on-time performance of domestic flights that were operated by large air carriers. You can use this data to train an ML model to predict if the flight is going to be delayed for the busiest airports.

About this dataset

This dataset contains scheduled and actual departure and arrival times reported by certified US air carriers that account for at least 1 percent of domestic scheduled passenger revenues. The data was collected by the U.S. Office of Airline Information, Bureau of Transportation Statistics (BTS). The dataset contains date, time, origin, destination, airline, distance, and delay status of flights for flights between 2013 and 2018.

Features

For more information about features in the dataset, see On-time delay dataset features.

Dataset attributions

Website: https://www.transtats.bts.gov/

Dataset(s) used in this lab were compiled by the U.S. Office of Airline Information, Bureau of Transportation Statistics (BTS), Airline On-Time Performance Data, available at https://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=120&DB_URL=Mode_ID=1&Mode_Desc=Aviation&Subject_ID2=0.

Step 1: Problem formulation and data collection

Start this project by writing a few sentences that summarize the business problem and the business goal that you want to achieve in this scenario. You can write down your ideas in the following sections. Include a business metric that you would like your team to aspire toward. After you define that information, write the ML problem statement. Finally, add a comment or two about the type of ML this activity represents.

Project presentation: Include a summary of these details in your project presentation.

1. Determine if and why ML is an appropriate solution to deploy for this scenario.

In [ ]:
```
# Write your answer here
```

2. Formulate the business problem, success metrics, and desired ML output.

In [ ]:
```
# Write your answer here
```

3. Identify the type of ML problem that you're working with.

In [ ]:
```
# Write your answer here
```

4. Analyze the appropriateness of the data that you're working with.

In [ ]:
```
# Write your answer here
```

Setup

Now that you have decided where you want to focus your attention, you will set up this lab so that you can start solving the problem.

Note: This notebook was created and tested on an `ml.m4.xlarge` notebook instance with 25 GB storage.

In [ ]:
```
import os
from pathlib2 import Path
from zipfile import ZipFile
import time

import pandas as pd
import numpy as np
import subprocess

import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
instance_type='ml.m4.xlarge'

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

Step 2: Data preprocessing and visualization

In this data preprocessing phase, you explore and visualize your data to better understand it. First, import the necessary libraries and read the data into a pandas DataFrame. After you import the data, explore the dataset. Look for the shape of the dataset and explore your columns and the types of columns that you will work with (numerical, categorical). Consider performing basic statistics on the features to get a sense of feature means and ranges. Examine your target column closely, and determine its distribution.

Specific questions to consider

Throughout this section of the lab, consider the following questions:

1. What can you deduce from the basic statistics that you ran on the features?
2. What can you deduce from the distributions of the target classes?
3. Is there anything else you can deduce by exploring the data?

Project presentation: Include a summary of your answers to these questions (and other similar questions) in your project presentation.

Start by bringing in the dataset from a public Amazon Simple Storage Service (Amazon S3) bucket to this notebook environment.

In [ ]:
```
# download the files

zip_path = '/home/ec2-user/SageMaker/project/data/FlightDelays/'
base_path = '/home/ec2-user/SageMaker/project/data/FlightDelays/'
csv_base_path = '/home/ec2-user/SageMaker/project/data/csvFlightDelays/'

!mkdir -p {zip_path}
!mkdir -p {csv_base_path}
!aws s3 cp s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/
```

In [ ]:
```
zip_files = [str(file) for file in list(Path(base_path).iterdir()) if '.zip' in str
len(zip_files)
```

Extract comma-separated values (CSV) files from the .zip files.

In [ ]:
```
def zip2csv(zipFile_name , file_path):
    """
    Extract csv from zip files
    zipFile_name: name of the zip file
    file_path : name of the folder to store csv
    """

    try:
        with ZipFile(zipFile_name, 'r') as z:
            print(f'Extracting {zipFile_name} ')
            z.extractall(path=file_path)
    except:
        print(f'zip2csv failed for {zipFile_name}')

for file in zip_files:
    zip2csv(file, csv_base_path)

print("Files Extracted")
```

In [ ]:
```
csv_files = [str(file) for file in list(Path(csv_base_path).iterdir()) if '.csv' in
len(csv_files)
```

Before you load the CSV file, read the HTML file from the extracted folder. This HTML file includes the background and more information about the features that are included in the dataset.

In [ ]:
```
from IPython.display import IFrame

IFrame(src=os.path.relpath(f"{csv_base_path}readme.html"), width=1000, height=600)
```

Load sample CSV file

Before you combine all the CSV files, examine the data from a single CSV file. By using pandas, read the `On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_9.csv` file first. You can use the built-in `read_csv` function in Python (pandas.read_csv documentation).

In [ ]:
```
df_temp = pd.read_csv(f"{csv_base_path}On_Time_Reporting_Carrier_On_Time_Performanc
```

Question: Print the row and column length in the dataset, and print the column names.

Hint: To view the rows and columns of a DataFrame, use the `<DataFrame>.shape` function. To view the column names, use the `<DataFrame>.columns` function.

In [ ]:
```
df_shape = # **ENTER YOUR CODE HERE**
print(f'Rows and columns in one CSV file is {df_shape}')
```

Question: Print the first 10 rows of the dataset.

Hint: To print `x` number of rows, use the built-in `head(x)` function in pandas.

In [ ]:
```
# Enter your code here
```

Question: Print all the columns in the dataset. To view the column names, use `<DataFrame>.columns`.

In [ ]:
```
print(f'The column names are :')
print('#########')
for col in <CODE>:# **ENTER YOUR CODE HERE**
    print(col)
```

Question: Print all the columns in the dataset that contain the word *Del*. This will help you see how many columns have *delay data* in them.

Hint: To include values that pass certain `if` statement criteria, you can use a Python list comprehension.

For example: `[x for x in [1,2,3,4,5] if x > 2]`

Hint: To check if the value is in a list, you can use the `in` keyword (Python in Keyword documentation).

For example: `5 in [1,2,3,4,5]`

In [ ]:
```
# Enter your code here
```

Here are some more questions to help you learn more about your dataset.

Questions

1. How many rows and columns does the dataset have?
2. How many years are included in the dataset?
3. What is the date range for the dataset?
4. Which airlines are included in the dataset?
5. Which origin and destination airports are covered?

Hints

- To show the dimensions of the DataFrame, use `df_temp.shape`.
- To refer to a specific column, use `df_temp.columnName` (for example, `df_temp.CarrierDelay`).
- To get unique values for a column, use `df_temp.column.unique()` (for, example `df_temp.Year.unique()`).

In [ ]:
```
print("The #rows and #columns are ", <CODE> , " and ", <CODE>)
print("The years in this dataset are: ", <CODE>)
print("The months covered in this dataset are: ", <CODE>)
print("The date range for data is :" , min(<CODE>), " to ", max(<CODE>))
print("The airlines covered in this dataset are: ", list(<CODE>))
print("The Origin airports covered are: ", list(<CODE>))
print("The Destination airports covered are: ", list(<CODE>))
```

Question: What is the count of all the origin and destination airports?

Hint: To find the values for each airport by using the **Origin** and **Dest** columns, you can use the `values_count` function in pandas (pandas.Series.value_counts documentation).

In [ ]:
```
counts = pd.DataFrame({'Origin':<CODE>, 'Destination':<CODE>})
counts
```

Question: Print the top 15 origin and destination airports based on number of flights in the dataset.

Hint: You can use the `sort_values` function in pandas (pandas.DataFrame.sort_values documentation).

In [ ]:
```
counts.sort_values(by=<CODE>,ascending=False).head(15) # Enter your code here
```

Given all the information about a flight trip, can you predict if it would be delayed?

The **ArrDel15** column is an indicator variable that takes the value *1* when the delay is more than 15 minutes. Otherwise, it takes a value of *0*.

You could use this as a target column for the classification problem.

Now, assume that you are traveling from San Francisco to Los Angeles on a work trip. You want to better manage your reservations in Los Angeles. Thus, want to have an idea of

whether your flight will be delayed, given a set of features. How many features from this dataset would you need to know before your flight?

Columns such as `DepDelay`, `ArrDelay`, `CarrierDelay`, `WeatherDelay`, `NASDelay`, `SecurityDelay`, `LateAircraftDelay`, and `DivArrDelay` contain information about a delay. But this delay could have occured at the origin or the destination. If there were a sudden weather delay 10 minutes before landing, this data wouldn't be helpful to managing your Los Angeles reservations.

So to simplify the problem statement, consider the following columns to predict an arrival delay:

`Year`, `Quarter`, `Month`, `DayofMonth`, `DayOfWeek`, `FlightDate`, `Reporting_Airline`, `Origin`, `OriginState`, `Dest`, `DestState`, `CRSDepTime`, `DepDelayMinutes`, `DepartureDelayGroups`, `Cancelled`, `Diverted`, `Distance`, `DistanceGroup`, `ArrDelay`, `ArrDelayMinutes`, `ArrDel15`, `AirTime`

You will also filter the source and destination airports to be:

- Top airports: ATL, ORD, DFW, DEN, CLT, LAX, IAH, PHX, SFO
- Top five airlines: UA, OO, WN, AA, DL

This information should help reduce the size of data across the CSV files that will be combined.

Combine all CSV files

First, create an empy DataFrame that you will use to copy your individual DataFrames from each file. Then, for each file in the `csv_files` list:

1. Read the CSV file into a dataframe
2. Filter the columns based on the `filter_cols` variable

```
columns = ['col1', 'col2']
df_filter = df[columns]
```

3. Keep only the `subset_vals` in each of the `subset_cols`. To check if the `val` is in the DataFrame column, use the `isin` function in pandas (pandas.DataFram.isin documentation). Then, choose the rows that include it.

```
df_eg[df_eg['col1'].isin('5')]
```

4. Concatenate the DataFrame with the empty DataFrame

In [ ]:
```
def combine_csv(csv_files, filter_cols, subset_cols, subset_vals, file_name):

    """
```

```
    Combine csv files into one Data Frame
    csv_files: list of csv file paths
    filter_cols: list of columns to filter
    subset_cols: list of columns to subset rows
    subset_vals: list of list of values to subset rows
    """

    df = pd.DataFrame()

    for file in csv_files:
        df_temp = pd.read_csv(file)
        df_temp = df_temp[filter_cols]
        for col, val in zip(subset_cols,subset_vals):
            df_temp = df_temp[df_temp[col].isin(val)]

        df = pd.concat([df, df_temp], axis=0)

    df.to_csv(file_name, index=False)
    print(f'Combined csv stored at {file_name}')
```

In [ ]:
```
#cols is the list of columns to predict Arrival Delay
cols = ['Year','Quarter','Month','DayofMonth','DayOfWeek','FlightDate',
        'Reporting_Airline','Origin','OriginState','Dest','DestState',
        'CRSDepTime','Cancelled','Diverted','Distance','DistanceGroup',
        'ArrDelay','ArrDelayMinutes','ArrDel15','AirTime']

subset_cols = ['Origin', 'Dest', 'Reporting_Airline']

# subset_vals is a list collection of the top origin and destination airports and t
subset_vals = [['ATL', 'ORD', 'DFW', 'DEN', 'CLT', 'LAX', 'IAH', 'PHX', 'SFO'],
               ['ATL', 'ORD', 'DFW', 'DEN', 'CLT', 'LAX', 'IAH', 'PHX', 'SFO'],
               ['UA', 'OO', 'WN', 'AA', 'DL']]
```

Use the previous function to merge all the different files into a single file that you can read easily.

Note: This process will take 5-7 minutes to complete.

In [ ]:
```
start = time.time()
combined_csv_filename = f"{base_path}combined_files.csv"
combine_csv(csv_files, cols, subset_cols, subset_vals, combined_csv_filename)
print(f'CSVs merged in {round((time.time() - start)/60,2)} minutes')
```

Load the dataset

Load the combined dataset.

In [ ]:
```
data = pd.read_csv(combined_csv_filename)
```

Print the first five records.

In [ ]:
```
# Enter your code here
```

Here are some more questions to help you learn more about your dataset.

Questions

1. How many rows and columns does the dataset have?
2. How many years are included in the dataset?
3. What is the date range for the dataset?
4. Which airlines are included in the dataset?
5. Which origin and destination airports are covered?

In [ ]:
```
print("The #rows and #columns are ", <CODE> , " and ", <CODE>)
print("The years in this dataset are: ", list(<CODE>))
print("The months covered in this dataset are: ", sorted(list(<CODE>)))
print("The date range for data is :" , min(<CODE>), " to ", max(<CODE>))
print("The airlines covered in this dataset are: ", list(<CODE>))
print("The Origin airports covered are: ", list(<CODE>))
print("The Destination airports covered are: ", list(<CODE>))
```

Define your target column: **is_delay** (*1* means that the arrival time delayed more than 15 minutes, and *0* means all other cases). To rename the column from **ArrDel15** to *is_delay*, use the `rename` method .

Hint: You can use the `rename` function in pandas (pandas.DataFrame.rename documentation).

For example:

```
data.rename(columns={'col1':'column1'}, inplace=True)
```

In [ ]:
```
data.rename(columns=<CODE>, inplace=True) # Enter your code here
```

Look for nulls across columns. You can use the `isnull()` function (pandas.isnull documentation).

Hint: `isnull()` detects whether the particular value is null or not. It returns a boolean (*True* or *False*) in its place. To sum the number of columns, use the `sum(axis=0)` function (for example, `df.isnull().sum(axis=0)`).

In [ ]:
```
# Enter your code here
```

The arrival delay details and airtime are missing for 22,540 out of 1,658,130 rows, which is 1.3 percent. You can either remove or impute these rows. The documentation doesn't mention any information about missing rows.

In [ ]:
```
### Remove null columns
data = data[~data.is_delay.isnull()]
data.isnull().sum(axis = 0)
```

Get the hour of the day in 24-hour-time format from CRSDepTime.

In [ ]:
```
data['DepHourofDay'] = (data['CRSDepTime']//100)
```

The ML problem statement

- Given a set of features, can you predict if a flight is going to be delayed more than 15 minutes?
- Because the target variable takes only a value of *0* or *1*, you could use a classification algorithm.

Before you start modeling, it's a good practice to look at feature distribution, correlations, and others.

- This will give you an idea of any non-linearity or patterns in the data
  - Linear models: Add power, exponential, or interaction features
  - Try a non-linear model
- Data imbalance
  - Choose metrics that won't give biased model performance (accuracy versus the area under the curve, or AUC)
  - Use weighted or custom loss functions
- Missing data
  - Do imputation based on simple statistics -- mean, median, mode (numerical variables), frequent class (categorical variables)
  - Clustering-based imputation (k-nearest neighbors, or KNNs, to predict column value)
  - Drop column

Data exploration

Check the classes *delay* versus *no delay*.

In [ ]:
```
(data.groupby('is_delay').size()/len(data) ).plot(kind='bar')# Enter your code here
plt.ylabel('Frequency')
plt.title('Distribution of classes')
plt.show()
```

Question: What can you deduce from the bar plot about the ratio of *delay* versus *no delay*?

In [ ]:
```
# Enter your answer here
```

Run the following two cells and answer the questions.

In [ ]:
```
viz_columns = ['Month', 'DepHourofDay', 'DayOfWeek', 'Reporting_Airline', 'Origin',
fig, axes = plt.subplots(3, 2, figsize=(20,20), squeeze=False)
```

```
# fig.autofmt_xdate(rotation=90)

for idx, column in enumerate(viz_columns):
    ax = axes[idx//2, idx%2]
    temp = data.groupby(column)['is_delay'].value_counts(normalize=True).rename('pe
    mul(100).reset_index().sort_values(column)
    sns.barplot(x=column, y="percentage", hue="is_delay", data=temp, ax=ax)
    plt.ylabel('% delay/no-delay')


plt.show()
```

In [ ]:
```
sns.lmplot( x="is_delay", y="Distance", data=data, fit_reg=False, hue='is_delay', l
plt.legend(loc='center')
plt.xlabel('is_delay')
plt.ylabel('Distance')
plt.show()
```

Questions

Using the data from the previous charts, answer these questions:

- Which months have the most delays?
- What time of the day has the most delays?
- What day of the week has the most delays?
- Which airline has the most delays?
- Which origin and destination airports have the most delays?
- Is flight distance a factor in the delays?

In [ ]:
```
# Enter your answers here
```

Features

Look at all the columns and what their specific types are.

In [ ]:
```
data.columns
```

In [ ]:
```
data.dtypes
```

Filtering the required columns:

- *Date* is redundant, because you have *Year*, *Quarter*, *Month*, *DayofMonth*, and *DayOfWeek* to describe the date.
- Use *Origin* and *Dest* codes instead of *OriginState* and *DestState*.
- Because you are only classifying whether the flight is delayed or not, you don't need *TotalDelayMinutes*, *DepDelayMinutes*, and *ArrDelayMinutes*.

Treat *DepHourofDay* as a categorical variable because it doesn't have any quantitative relation with the target.

- If you needed to do a one-hot encoding of this variable, it would result in 23 more columns.
- Other alternatives to handling categorical variables include hash encoding, regularized mean encoding, and bucketizing the values, among others.
- In this case, you only need to split into buckets.

To change a column type to category, use the `astype` function (pandas.DataFrame.astype documentation).

In [ ]:
```
data_orig = data.copy()
data = data[[ 'is_delay', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek',
       'Reporting_Airline', 'Origin', 'Dest','Distance','DepHourofDay']]
categorical_columns  = ['Quarter', 'Month', 'DayofMonth', 'DayOfWeek',
       'Reporting_Airline', 'Origin', 'Dest', 'DepHourofDay']
for c in categorical_columns:
    data[c] = data[c].astype('category')
```

To use one-hot encoding, use the `get_dummies` function in pandas for the categorical columns that you selected. Then, you can concatenate those generated features to your original dataset by using the `concat` function in pandas. For encoding categorical variables, you can also use *dummy encoding* by using a keyword `drop_first=True`. For more information about dummy encoding, see Dummy variable (statistics).

For example:

```
pd.get_dummies(df[['column1','columns2']], drop_first=True)
```

In [ ]:
```
data_dummies = pd.get_dummies(<CODE>, drop_first=True) # Enter your code here
data_dummies = data_dummies.replace({True: 1, False: 0})
data = pd.concat([<CODE>, <CODE>], axis = 1)
data.drop(categorical_columns,axis=1, inplace=True)
```

Check the length of the dataset and the new columns.

Hint: Use the `shape` and `columns` properties.

In [ ]:
```
# Enter your code here
```

In [ ]:
```
# Enter your code here
```

You are now ready to train the model. Before you split the data, rename the **is_delay** column to *target*.

Hint: You can use the `rename` function in pandas (pandas.DataFrame.rename documentation).

In [ ]:
```
data.rename(columns = {<CODE>:<CODE>}, inplace=True )# Enter your code here
```

End of Step 2

Save the project file to your local computer. Follow these steps:

1. In the file explorer on the left, right-click the notebook that you're working on.
2. Choose **Download**, and save the file locally.

This action downloads the current notebook to the default download folder on your computer.

Step 3: Model training and evaluation

You must include some preliminary steps when you convert the dataset from a DataFrame to a format that a machine learning algorithm can use. For Amazon SageMaker, you must perform these steps:

1. Split the data into `train_data`, `validation_data`, and `test_data` by using `sklearn.model_selection.train_test_split`.
2. Convert the dataset to an appropriate file format that the Amazon SageMaker training job can use. This can be either a CSV file or record protobuf. For more information, see Common Data Formats for Training.
3. Upload the data to your S3 bucket. If you haven't created one before, see Create a Bucket.

Use the following cells to complete these steps. Insert and delete cells where needed.

Project presentation: In your project presentation, write down the key decisions that you made in this phase.

Train-test split

In [ ]:
```
from sklearn.model_selection import train_test_split
def split_data(data):
    train, test_and_validate = train_test_split(data, test_size=0.2, random_state=4
    test, validate = train_test_split(test_and_validate, test_size=0.5, random_stat
    return train, validate, test
```

In [ ]:
```
train, validate, test = split_data(data)
print(train['target'].value_counts())
print(test['target'].value_counts())
print(validate['target'].value_counts())
```

Sample answer

```
0.0    1033570
1.0     274902
Name: target, dtype: int64
0.0    129076
1.0     34483
Name: target, dtype: int64
0.0    129612
1.0     33947
Name: target, dtype: int64
```

Baseline classification model

In [ ]:
```
import sagemaker
from sagemaker.serializers import CSVSerializer
from sagemaker.amazon.amazon_estimator import RecordSet
import boto3

# Instantiate the LinearLearner estimator object with 1 ml.m4.xlarge
classifier_estimator = sagemaker.LinearLearner(role=sagemaker.get_execution_role(),
                                               instance_count=<CODE>,
                                               instance_type=<CODE>,
                                               predictor_type=<CODE>,
                                               binary_classifier_model_selection_cr
```

Sample code

```
num_classes = len(pd.unique(train_labels))
classifier_estimator =
sagemaker.LinearLearner(role=sagemaker.get_execution_role(),
                                                    instance_count=1,

instance_type='ml.m4.xlarge',

predictor_type='binary_classifier',

binary_classifier_model_selection_criteria = 'cross_entropy_loss')
```

Linear learner accepts training data in protobuf or CSV content types. It also accepts inference requests in protobuf, CSV, or JavaScript Object Notation (JSON) content types. Training data has features and ground-truth labels, but the data in an inference request has only features.

In a production pipeline, AWS recommends converting the data to the Amazon SageMaker protobuf format and storing it in Amazon S3. To get up and running quickly, AWS provides the `record_set` operation for converting and uploading the dataset when it's small enough to fit in local memory. It accepts NumPy arrays like the ones you already have, so you will use it for this step. The `RecordSet` object will track the temporary Amazon S3 location of your data. Create train, validation, and test records by using the

`estimator.record_set` function. Then, start your training job by using the `estimator.fit` function.

In [ ]:
```
### Create train, validate, and test records
train_records = classifier_estimator.record_set(train.values[:, 1:].astype(np.float
val_records = classifier_estimator.record_set(validate.values[:, 1:].astype(np.floa
test_records = classifier_estimator.record_set(test.values[:, 1:].astype(np.float32
```

Now, train your model on the dataset that you just uploaded.

Sample code

```
linear.fit([train_records,val_records,test_records])
```

In [ ]:
```
### Fit the classifier
# Enter your code here
```

Model evaluation

In this section, you will evaluate your trained model.

First, examine the metrics for the training job:

In [ ]:
```
sagemaker.analytics.TrainingJobAnalytics(classifier_estimator._current_job_name,
                                         metric_names = ['test:objective_loss',
                                                         'test:binary_f_beta',
                                                         'test:precision',
                                                         'test:recall']
                                        ).dataframe()
```

Next, set up some functions that will help load the test data into Amazon S3 and perform a prediction by using the batch prediction function. Using batch prediction will help reduce costs because the instances will only run when predictions are performed on the supplied test data.

Note: Replace `<LabBucketName>` with the name of the lab bucket that was created during the lab setup.

In [ ]:
```
import io
#bucket='<LabBucketName>'
prefix='flight-linear'
train_file='flight_train.csv'
test_file='flight_test.csv'
validate_file='flight_validate.csv'
whole_file='flight.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)).put(B
```

In [ ]:
```
def batch_linear_predict(test_data, estimator):
    batch_X = test_data.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)

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

    classifier_transformer.transform(data=batch_input,
                             data_type='S3Prefix',
                             content_type='text/csv',
                             split_type='Line')

    classifier_transformer.wait()

    s3 = boto3.client('s3')
    obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-i
    target_predicted_df = pd.read_json(io.BytesIO(obj['Body'].read()),orient="recor
    return test_data.iloc[:,0], target_predicted_df.iloc[:,0]
```

To run the predictions on the test dataset, run the `batch_linear_predict` function (which was defined previously) on your test dataset.

In [ ]:
```
test_labels, target_predicted = batch_linear_predict(test, classifier_estimator)
```

To view a plot of the confusion matrix, and various scoring metrics, create a couple of functions:

In [ ]:
```
from sklearn.metrics import confusion_matrix

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=colormap)
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.ylabel("True Class")
    plt.xlabel("Predicted Class")
    plt.show()
```

In [ ]:
```
from sklearn import metrics

def plot_roc(test_labels, target_predicted):
```

```
    TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted).ravel()
    # 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("Sensitivity or TPR: ", Sensitivity, "%")
    print( "Specificity or TNR: ",Specificity, "%")
    print("Precision: ",Precision, "%")
    print("Negative Predictive Value: ",NPV, "%")
    print( "False Positive Rate: ",FPR,"%")
    print("False Negative Rate: ",FNR, "%")
    print("False Discovery Rate: ",FDR, "%" )
    print("Accuracy: ",ACC, "%")

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

    fpr, tpr, thresholds = metrics.roc_curve(test_labels, target_predicted)
    roc_auc = metrics.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='r')
    ax2.set_ylabel('Threshold',color='r')
    ax2.set_ylim([thresholds[-1],thresholds[0]])
    ax2.set_xlim([fpr[0],fpr[-1]])

    print(plt.figure())
```

To plot the confusion matrix, call the `plot_confusion_matrix` function on the `test_labels` and the `target_predicted` data from your batch job:

In [ ]:
```
# Enter your code here
```

Key questions to consider:

1. How does your model's performance on the test set compare to its performance on the training set? What can you deduce from this comparison?
2. Are there obvious differences between the outcomes of metrics like accuracy, precision, and recall? If so, why might you be seeing those differences?
3. Given your business situation and goals, which metric (or metrics) is the most important for you to consider? Why?
4. From a business standpoint, is the outcome for the metric (or metrics) that you consider to be the most important sufficient for what you need? If not, what are some things you might change in your next iteration? (This will happen in the feature engineering section, which is next.)

Use the following cells to answer these (and other) questions. Insert and delete cells where needed.

Project presentation: In your project presentation, write down your answers to these questions -- and other similar questions that you might answer -- in this section. Record the key details and decisions that you made.

Question: What can you summarize from the confusion matrix?

In [ ]:
```
# Enter your answer here
```

End of Step 3

Save the project file to your local computer. Follow these steps:

1. In the file explorer on the left, right-click the notebook that you're working on.
2. Select **Download**, and save the file locally.

This action downloads the current notebook to the default download folder on your computer.

Iteration II

Step 4: Feature engineering

You have now gone through one iteration of training and evaluating your model. Given that the first outcome that you reached for your model probably wasn't sufficient for solving your business problem, what could you change about your data to possibly improve model performance?

Key questions to consider:

1. How might the balance of your two main classes (*delay* and *no delay*) impact model performance?
2. Do you have any features that are correlated?
3. At this stage, could you perform any feature-reduction techniques that might have a positive impact on model performance?
4. Can you think of adding some more data or datasets?
5. After performing some feature engineering, how does the performance of your model compare to the first iteration?

Use the following cells to perform specific feature-engineering techniques that you think could improve your model performance (use the previous questions as a guide). Insert and delete cells where needed.

Project presentation: In your project presentation, record your key decisions and the methods that you use in this section. Also include any new performance metrics that you obtain after you evaluate your model again.

Before you start, think about why the precision and recall are around 80 percent, and the accuracy is at 99 percent.

Add more features:

1. Holidays
2. Weather

Because the list of holidays from 2014 to 2018 is known, you can create an indicator variable **is_holiday** to mark them.

The hypothesis is that airplane delays could be higher during holidays compared to the rest of the days. Add a boolean variable `is_holiday` that includes the holidays for the years 2014-2018.

In [ ]:
```
# Source: http://www.calendarpedia.com/holidays/federal-holidays-2014.html

holidays_14 = ['2014-01-01',  '2014-01-20', '2014-02-17', '2014-05-26', '2014-07-04
holidays_15 = ['2015-01-01',  '2015-01-19', '2015-02-16', '2015-05-25', '2015-06-03
holidays_16 = ['2016-01-01',  '2016-01-18', '2016-02-15', '2016-05-30', '2016-07-04
holidays_17 = ['2017-01-02', '2017-01-16', '2017-02-20', '2017-05-29' , '2017-07-04
holidays_18 = ['2018-01-01', '2018-01-15', '2018-02-19', '2018-05-28' , '2018-07-04
holidays = holidays_14+ holidays_15+ holidays_16 + holidays_17+ holidays_18
```

```
### Add indicator variable for holidays
data_orig['is_holiday'] = # Enter your code here
```

Weather data was fetched from https://www.ncei.noaa.gov/access/services/data/v1?dataset=daily-summaries&stations=USW00023174,USW00012960,USW00003017,USW00094846,USW0001387 01-01&endDate=2018-12-31.

This dataset has information on wind speed, precipitation, snow, and temperature for cities by their airport codes.

Question: Could bad weather because of rain, heavy winds, or snow lead to airplane delays? You will now check.

In [ ]:
```
!aws s3 cp s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data2/
#!wget 'https://www.ncei.noaa.gov/access/services/data/v1?dataset=daily-summaries&s
```

Import the weather data that was prepared for the airport codes in the dataset. Use the following stations and airports for the analysis. Create a new column called *airport* that maps the weather station to the airport name.

In [ ]:
```
weather = pd.read_csv('/home/ec2-user/SageMaker/project/data/daily-summaries.csv')
station = ['USW00023174','USW00012960','USW00003017','USW00094846','USW00013874','U
airports = ['LAX', 'IAH', 'DEN', 'ORD', 'ATL', 'SFO', 'DFW', 'PHX', 'CLT']

### Map weather stations to airport code
station_map = {s:a for s,a in zip(station, airports)}
weather['airport'] = weather['STATION'].map(station_map)
```

From the **DATE** column, create another column called *MONTH*.

In [ ]:
```
weather['MONTH'] = weather['DATE'].apply(lambda x: x.split('-')[1])
weather.head()
```

Sample output

```
  STATION     DATE       AWND PRCP SNOW SNWD TAVG TMAX  TMIN airport
MONTH
0 USW00023174 2014-01-01 16   0    NaN  NaN 131.0 178.0 78.0  LAX
01
1 USW00023174 2014-01-02 22   0    NaN  NaN 159.0 256.0 100.0 LAX
01
2 USW00023174 2014-01-03 17   0    NaN  NaN 140.0 178.0 83.0  LAX
01
3 USW00023174 2014-01-04 18   0    NaN  NaN 136.0 183.0 100.0 LAX
01
```

```
4 USW00023174 2014-01-05 18   0   NaN  NaN 151.0 244.0 83.0  LAX
01
```

Analyze and handle the **SNOW** and **SNWD** columns for missing values by using `fillna()`. To check the missing values for all the columns, use the `isna()` function.

In [ ]:
```
weather.SNOW.fillna(0, inplace=True)
weather.SNWD.fillna(0, inplace=True)
weather.isna().sum()
```

Question: Print the index of the rows that have missing values for *TAVG*, *TMAX*, *TMIN*.

Hint: To find the rows that are missing, use the `isna()` function. Then, to get the index, use the list on the *idx* variable.

In [ ]:
```
idx = np.array([i for i in range(len(weather))])
TAVG_idx = idx[weather.TAVG.isna()]
TMAX_idx = # Enter your code here
TMIN_idx = # Enter your code here
TAVG_idx
```

Sample output

```
array([ 3956,  3957,  3958,  3959,  3960,  3961,  3962,  3963,
3964,
        3965,  3966,  3967,  3968,  3969,  3970,  3971,  3972,
3973,
        3974,  3975,  3976,  3977,  3978,  3979,  3980,  3981,
3982,
        3983,  3984,  3985,  4017,  4018,  4019,  4020,  4021,
4022,
        4023,  4024,  4025,  4026,  4027,  4028,  4029,  4030,
4031,
        4032,  4033,  4034,  4035,  4036,  4037,  4038,  4039,
4040,
        4041,  4042,  4043,  4044,  4045,  4046,  4047, 13420])
```

You can replace the missing *TAVG*, *TMAX*, and *TMIN* values with the average value for a particular station or airport. Because consecutive rows of *TAVG_idx* are missing, replacing them with a previous value would not be possible. Instead, replace them with the mean. Use the `groupby` function to aggregate the variables with a mean value.

Hint: Group by `MONTH` and `STATION`.

In [ ]:
```
weather_impute = weather.groupby([<CODE>]).agg({'TAVG':'mean','TMAX':'mean', 'TMIN'
weather_impute.head(2)
```

Merge the mean data with the weather data.

In [ ]:
```
weather = pd.merge(weather, weather_impute,  how='left', left_on=['MONTH','STATION'
.rename(columns = {'TAVG_y':'TAVG_AVG',
                   'TMAX_y':'TMAX_AVG',
                   'TMIN_y':'TMIN_AVG',
                   'TAVG_x':'TAVG',
                   'TMAX_x':'TMAX',
                   'TMIN_x':'TMIN'})
```

Check for missing values again.

In [ ]:
```
weather.TAVG[TAVG_idx] = weather.TAVG_AVG[TAVG_idx]
weather.TMAX[TMAX_idx] = weather.TMAX_AVG[TMAX_idx]
weather.TMIN[TMIN_idx] = weather.TMIN_AVG[TMIN_idx]
weather.isna().sum()
```

Drop `STATION,MONTH,TAVG_AVG,TMAX_AVG,TMIN_AVG,TMAX,TMIN,SNWD` from the dataset.

In [ ]:
```
weather.drop(columns=['STATION','MONTH','TAVG_AVG', 'TMAX_AVG', 'TMIN_AVG', 'TMAX'
```

Add the origin and destination weather conditions to the dataset.

In [ ]:
```
### Add origin weather conditions
data_orig = pd.merge(data_orig, weather,  how='left', left_on=['FlightDate','Origin
.rename(columns = {'AWND':'AWND_O','PRCP':'PRCP_O', 'TAVG':'TAVG_O', 'SNOW': 'SNOW_
.drop(columns=['DATE','airport'])

### Add destination weather conditions
data_orig = pd.merge(data_orig, weather,  how='left', left_on=['FlightDate','Dest']
.rename(columns = {'AWND':'AWND_D','PRCP':'PRCP_D', 'TAVG':'TAVG_D', 'SNOW': 'SNOW_
.drop(columns=['DATE','airport'])
```

Note: It's always a good practice to check for nulls or NAs after joins.

In [ ]:
```
sum(data.isna().any())
```

In [ ]:
```
data_orig.columns
```

Convert the categorical data into numerical data by using one-hot encoding.

In [ ]:
```
data = data_orig.copy()
data = data[['is_delay', 'Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek',
       'Reporting_Airline', 'Origin', 'Dest','Distance','DepHourofDay','is_holiday'
       'TAVG_O', 'AWND_D', 'PRCP_D', 'TAVG_D', 'SNOW_O', 'SNOW_D']]


categorical_columns  = ['Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek',
       'Reporting_Airline', 'Origin', 'Dest', 'is_holiday']
for c in categorical_columns:
    data[c] = data[c].astype('category')
```

In [ ]:
```
data_dummies = pd.get_dummies(data[['Year', 'Quarter', 'Month', 'DayofMonth', 'DayO
data_dummies = data_dummies.replace({True: 1, False: 0})
data = pd.concat([data, data_dummies], axis = 1)
data.drop(categorical_columns,axis=1, inplace=True)
```

Check the new columns.

In [ ]:
```
data.shape
```

In [ ]:
```
data.columns
```

Sample output

```
Index(['Distance', 'DepHourofDay', 'is_delay', 'AWND_O', 'PRCP_O',
'TAVG_O',
       'AWND_D', 'PRCP_D', 'TAVG_D', 'SNOW_O', 'SNOW_D',
'Year_2015',
       'Year_2016', 'Year_2017', 'Year_2018', 'Quarter_2',
'Quarter_3',
       'Quarter_4', 'Month_2', 'Month_3', 'Month_4', 'Month_5',
'Month_6',
       'Month_7', 'Month_8', 'Month_9', 'Month_10', 'Month_11',
'Month_12',
       'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4',
'DayofMonth_5',
       'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8',
'DayofMonth_9',
       'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12',
'DayofMonth_13',
       'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16',
'DayofMonth_17',
       'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20',
'DayofMonth_21',
       'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24',
'DayofMonth_25',
       'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28',
'DayofMonth_29',
       'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2',
'DayOfWeek_3',
       'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWeek_7',
       'Reporting_Airline_DL', 'Reporting_Airline_OO',
'Reporting_Airline_UA',
       'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN',
'Origin_DFW',
       'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PHX',
'Origin_SFO',
       'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_LAX',
'Dest_ORD',
       'Dest_PHX', 'Dest_SFO', 'is_holiday_1'],
      dtype='object')
```

Rename the **is_delay** column to *target* again. Use the same code that you used previously.

In [ ]:
```
data.rename(columns = {<CODE>:<CODE>}, inplace=True )# Enter your code here
```

Create the training sets again.

Hint: Use the `split_data` function that you defined (and used) earlier.

In [ ]:
```
# Enter your code here
```

New baseline classifier

Now, see if these new features add any predictive power to the model.

In [ ]:
```
# Instantiate the LinearLearner estimator object
classifier_estimator2 = # Enter your code here
```

Sample code

```
num_classes = len(pd.unique(train_labels))
classifier_estimator2 =
sagemaker.LinearLearner(role=sagemaker.get_execution_role(),
                                                    instance_count=1,

instance_type='ml.m4.xlarge',

predictor_type='binary_classifier',

binary_classifier_model_selection_criteria = 'cross_entropy_loss')
```

In [ ]:
```
train_records = classifier_estimator2.record_set(train.values[:, 1:].astype(np.floa
val_records = classifier_estimator2.record_set(validate.values[:, 1:].astype(np.flo
test_records = classifier_estimator2.record_set(test.values[:, 1:].astype(np.float3
```

Train your model by using the three datasets that you just created.

In [ ]:
```
# Enter your code here
```

Perform a batch prediction by using the newly trained model.

In [ ]:
```
# Enter your code here
```

Plot a confusion matrix.

In [ ]:
```
# Enter your code here
```

The linear model shows only a little improvement in performance. Try a tree-based ensemble model, which is called *XGBoost*, with Amazon SageMaker.

Try the XGBoost model

Perform these steps:

1. Use the training set variables and save them as CSV files: train.csv, validation.csv and test.csv.
2. Store the bucket name in the variable. The Amazon S3 bucket name is provided to the left of the lab instructions.

a. `bucket = <LabBucketName>`
b. `prefix = 'flight-xgb'`
3. Use the AWS SDK for Python (Boto3) to upload the model to the bucket.

In [ ]:
```
bucket='c141641a3636227l8403132t1w85172556060-flightbucket-ch0wohlcstup'
prefix='flight-xgb'
train_file='flight_train.csv'
test_file='flight_test.csv'
validate_file='flight_validate.csv'
whole_file='flight.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)).put(B

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

Use the `sagemaker.inputs.TrainingInput` function to create a `record_set` for the training and validation datasets.

In [ ]:
```
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}
```

In [ ]:
```
from sagemaker.image_uris import retrieve
container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')
```

In [ ]:
```
sess = sagemaker.Session()
s3_output_location="s3://{}/{}/output/".format(bucket,prefix)

xgb = sagemaker.estimator.Estimator(container,
                                    role = sagemaker.get_execution_role(),
                                    instance_count=1,
                                    instance_type=instance_type,
                                    output_path=s3_output_location,
                                    sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        eval_metric = "auc",
                        num_round=100)

xgb.fit(inputs=data_channels)
```

Use the batch transformer for your new model, and evaluate the model on the test dataset.

In [ ]:
```
batch_X = test.iloc[:,1:];
batch_X_file='batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)
```

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

xgb_transformer = xgb.transformer(instance_count=1,
                                  instance_type=instance_type,
                                  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()
```

Get the predicted target and test labels.

In [ ]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.cs
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),sep=',',names=['targe
test_labels = test.iloc[:,0]
```

Calculate the predicted values based on the defined threshold.

Note: The predicted target will be a score, which must be converted to a binary class.

In [ ]:
```
print(target_predicted.head())

def binary_convert(x):
    threshold = 0.55
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['target'] = target_predicted['target'].apply(binary_convert)

test_labels = test.iloc[:,0]

print(target_predicted.head())
```

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

In [ ]:
```
# Enter your code here
```

Try different thresholds

Question: Based on how well the model handled the test set, what can you conclude?

In [ ]:
```
#Enter your answer here
```

Hyperparameter optimization (HPO)

In [ ]:
```
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousParam

### You can spin up multiple instances to do hyperparameter optimization in paralle

xgb = sagemaker.estimator.Estimator(container,
                                    role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make sure you have a limit
                                    instance_type=instance_type,
                                    output_path='s3://{}/{}/output'.format(bucket,
                                    sagemaker_session=sess)

xgb.set_hyperparameters(eval_metric='auc',
                        objective='binary:logistic',
                        num_round=100,
                        rate_drop=0.3,
                        tweedie_variance_power=1.4)

hyperparameter_ranges = {'alpha': ContinuousParameter(0, 1000, scaling_type='Linear
                         'eta': ContinuousParameter(0.1, 0.5, scaling_type='Linear'
                         'min_child_weight': ContinuousParameter(3, 10, scaling_typ
                         'subsample': ContinuousParameter(0.5, 1),
                         'num_round': IntegerParameter(10,150)}

objective_metric_name = 'validation:auc'
```

```
tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or above depending upon b
                            max_parallel_jobs=1)
```

In [ ]:
```
tuner.fit(inputs=data_channels)
tuner.wait()
```

Wait until the training job is finished. It might take 25-30 minutes.

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.

Check that the job completed successfully.

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

The hyperparameter tuning job will have a model that worked the best. You can get the information about that model from the tuning job.

In [ ]:
```
sage_client = boto3.Session().client('sagemaker')
tuning_job_name = tuner.latest_tuning_job.job_name
print(f'tuning job name:{tuning_job_name}')
tuning_job_result = sage_client.describe_hyper_parameter_tuning_job(HyperParameterT
best_training_job = tuning_job_result['BestTrainingJob']
best_training_job_name = best_training_job['TrainingJobName']
print(f"best training job: {best_training_job_name}")

best_estimator = tuner.best_estimator()

tuner_df = sagemaker.HyperparameterTuningJobAnalytics(tuning_job_name).dataframe()
tuner_df.head()
```

Use the estimator `best_estimator` and train it by using the data.

Tip: See the previous XGBoost estimator fit function.

In [ ]:
```
# Enter your code here'
```

Use the batch transformer for your new model, and evaluate the model on the test dataset.

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

xgb_transformer = best_estimator.transformer(instance_count=1,
```

```
                                        instance_type=instance_type,
                                        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()
```

In [ ]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.cs
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),sep=',',names=['targe
test_labels = test.iloc[:,0]
```

Get the predicted target and test labels.

In [ ]:
```
print(target_predicted.head())

def binary_convert(x):
    threshold = 0.55
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['target'] = target_predicted['target'].apply(binary_convert)

test_labels = test.iloc[:,0]

print(target_predicted.head())
```

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

In [ ]:
```
# Enter your code here
```

Question: Try different hyperparameters and hyperparameter ranges. Do these changes improve the model?

Conclusion

You have now iterated through training and evaluating your model at least a couple of times. It's time to wrap up this project and reflect on:

- What you learned
- What types of steps you might take moving forward (assuming that you had more time)

Use the following cell to answer some of these questions and other relevant questions:

1. Does your model performance meet your business goal? If not, what are some things you'd like to do differently if you had more time for tuning?
2. How much did your model improve as you made changes to your dataset, features, and hyperparameters? What types of techniques did you employ throughout this project, and which yielded the greatest improvements in your model?
3. What were some of the biggest challenges that you encountered throughout this project?
4. Do you have any unanswered questions about aspects of the pipeline that didn't make sense to you?
5. What were the three most important things that you learned about machine learning while working on this project?

Project presentation: Make sure that you also summarize your answers to these questions in your project presentation. Combine all your notes for your project presentation and prepare to present your findings to the class.

In [ ]:
```
# Write your answers here
```