CoxAssignment08

Task: Predicting Delayed Flights

This dataset contains information on all commercial flights departing the Washington, DC area and arriving at New York during January 2004. For each flight, there is information on the departure and arrival airports, the distance of the route, the scheduled time and date of the flight, and so on. The variable that we are trying to predict is whether or not a flight is delayed. A delay is defined as an arrival that is at least 15 minutes later than scheduled.

Dataset: FlightDelays

In [1]:
```
pip install pydotplus
```

```
Requirement already satisfied: pydotplus in c:\users\jcjcb\anaconda3\lib\site-package
s (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\jcjcb\anaconda3\lib\site-
packages (from pydotplus) (3.0.4)Note: you may need to restart the kernel to use upda
ted packages.
```

In [126…
```
# Import required packages for this chapter
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
import matplotlib.pylab as plt
%matplotlib inline
import graphviz
from sklearn import tree
```

Data Preprocessing

- Transform variable day of week (DAY_WEEK) info a categorical variable.
- Bin the scheduled departure time into eight bins.
- Use these and all other columns as predictors (excluding DAY_OF_MONTH).

In [114…
```
# Load the data
delays_df = pd.read_csv('FlightDelays.csv')
delays_df
```

Out[114]:

	CRS_DEP_TIME	CARRIER	DEP_TIME	DEST	DISTANCE	FL_DATE	FL_NUM	ORIGIN	Weather
0	1455	OH	1455	JFK	184	01/01/2004	5935	BWI	0
1	1640	DH	1640	JFK	213	01/01/2004	6155	DCA	0
2	1245	DH	1245	LGA	229	01/01/2004	7208	IAD	0
3	1715	DH	1709	LGA	229	01/01/2004	7215	IAD	0
4	1039	DH	1035	LGA	229	01/01/2004	7792	IAD	0
...	...	...	...	...	...	...	...	...	...
2196	645	RU	644	EWR	199	1/31/2004	2761	DCA	0
2197	1700	RU	1653	EWR	213	1/31/2004	2497	IAD	0
2198	1600	RU	1558	EWR	199	1/31/2004	2361	DCA	0
2199	1359	RU	1403	EWR	199	1/31/2004	2216	DCA	0
2200	1730	RU	1736	EWR	199	1/31/2004	2097	DCA	0

2201 rows × 13 columns

In [115…

```
delays_df.dtypes
```

Out[115]:

```
CRS_DEP_TIME      int64
CARRIER          object
DEP_TIME          int64
DEST             object
DISTANCE          int64
FL_DATE          object
FL_NUM            int64
ORIGIN           object
Weather           int64
DAY_WEEK          int64
DAY_OF_MONTH      int64
TAIL_NUM         object
Flight Status    object
dtype: object
```

In [116…

```
# convert variable DAY_WEEK to categorical data type
delays_df['DAY_WEEK'] = delays_df['DAY_WEEK'].astype('category')
```

In [117…

```
delays_df.dtypes
```

Out[117]:
```
CRS_DEP_TIME         int64
CARRIER             object
DEP_TIME             int64
DEST                object
DISTANCE             int64
FL_DATE             object
FL_NUM               int64
ORIGIN              object
Weather              int64
DAY_WEEK          category
DAY_OF_MONTH         int64
TAIL_NUM            object
Flight Status       object
dtype: object
```

In [118…
```
bins = [300,600,900,1200,1500,1800,2100,2400]
labels= [1,2,3,4,5,6,7]
delays_df['binned_CRS_DEP_TIME'] = pd.cut(delays_df['CRS_DEP_TIME'], bins=bins, labels
```

In [119…
```
delays_df.dtypes
```

Out[119]:
```
CRS_DEP_TIME                int64
CARRIER                    object
DEP_TIME                    int64
DEST                       object
DISTANCE                    int64
FL_DATE                    object
FL_NUM                      int64
ORIGIN                     object
Weather                     int64
DAY_WEEK                 category
DAY_OF_MONTH                int64
TAIL_NUM                   object
Flight Status              object
binned_CRS_DEP_TIME      category
dtype: object
```

In [120…
```
# remove DAY_OF_MONTH variable
delays_df = delays_df.drop('DAY_OF_MONTH', axis=1)
delays_df.head()
```

Out[120]:

	CRS_DEP_TIME	CARRIER	DEP_TIME	DEST	DISTANCE	FL_DATE	FL_NUM	ORIGIN	Weather	D/
0	1455	OH	1455	JFK	184	01/01/2004	5935	BWI	0	
1	1640	DH	1640	JFK	213	01/01/2004	6155	DCA	0	
2	1245	DH	1245	LGA	229	01/01/2004	7208	IAD	0	
3	1715	DH	1709	LGA	229	01/01/2004	7215	IAD	0	
4	1039	DH	1035	LGA	229	01/01/2004	7792	IAD	0	

- Think about this carefully.. the table below shows all the relevant predictors. Consider why you are removing the ones that may not make the cut.

- Do not include DEP_TIME (actual departure time) in the model because it is unknown at the time of prediction (unless we are generating our predictions of delays after the plane takes off, which is unlikely).
- Partition the data into training (70%) and validation (30%) sets.
- Use a tree with maximum depth 8 and minimum impurity decrease = 0.01. Express the resulting tree as a set of rules.

In [121…

```
delays_df2 = delays_df[['CARRIER', 'DEST', 'DISTANCE', 'ORIGIN', 'Weather', 'DAY_WEEK'
delays_df2.head()
```

Out[121]:

	CARRIER	DEST	DISTANCE	ORIGIN	Weather	DAY_WEEK	binned_CRS_DEP_TIME
0	OH	JFK	184	BWI	0	4	4
1	DH	JFK	213	DCA	0	4	5
2	DH	LGA	229	IAD	0	4	4
3	DH	LGA	229	IAD	0	4	5
4	DH	LGA	229	IAD	0	4	3

In [122…

```
# create dummies for categorical variables
X = delays_df2.drop(columns= ['CARRIER', 'DEST', 'ORIGIN'])
y = delays_df2[['CARRIER', 'DEST', 'ORIGIN']]
```

In [123…

```
# partition the data into training (70%) and validation (30%) sets. set random_state=1
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [124…

```
# fit a tree model and draw tree with maximum depth 8, minimum sample split as 50, and
DelayTree = DecisionTreeClassifier(max_depth=8, min_samples_split=50, min_impurity_dec
DelayTree.fit(train_X, train_y)
```

Out[124]:

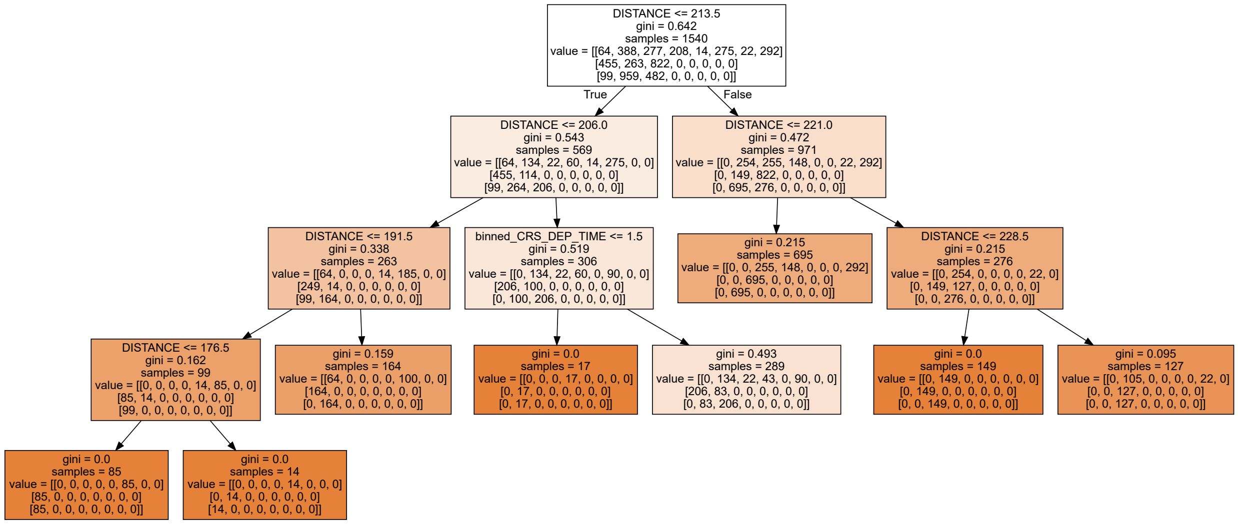
```
DecisionTreeClassifier(max_depth=8, min_impurity_decrease=0.01,
                       min_samples_split=50)
```

In [127…

```
dot_data = tree.export_graphviz(DelayTree, out_file=None,
                                feature_names=train_X.columns,
                                filled=True)

graph = graphviz.Source(dot_data, format="png")
graph
```

Out[127]:



Discuss the rules created from the tree below:

- The max_depth parameter limits how big the tree can be. It does not show all possibilities. The min_samples_split is the least amount samples required to split a node and branch out. The min_impurity_decrease is for when a node starts to lose its impurity and causes it to branch out once the certain value is hit.

Example: If you needed to fly between DCA and EWR on a Monday at 7:00 AM, would you be able to use the tree you created? What other information would you need? Is it available in practice? What information is redundant?

- I think you could use the tree I created. It would require you to know the distance of the flight as the tree bases its findings on how far the flight is. You would need to how busy the airport is and whether the departing or arrival airport is experienceing any delays from non-airline related causes. Some of that information is available as most airlines tend to keep up to date informatin on their site regarding such delays but it is almost impossible to account for all potential causes of delays such as if a plane unexpectedly crashes on the landing pad when all day the flights were ontime and operations were running smoothly. The information that is redundant is a lot of the values. There are a lot of '0' in the mix on the tree.

Fit the same tree as in the previous example, this time excluding the Weather predictor. Display both the resulting (small) tree and the full-grown tree. You will find that the small tree contains a single terminal node.

In [134…

```
# remove variable Weather from the analysis
delays_df3 = delays_df[['CARRIER', 'DEST', 'DISTANCE', 'ORIGIN', 'DAY_WEEK', 'binned_(
delays_df3.head()
```

Out[134]:

	CARRIER	DEST	DISTANCE	ORIGIN	DAY_WEEK	binned_CRS_DEP_TIME
0	OH	JFK	184	BWI	4	4
1	DH	JFK	213	DCA	4	5
2	DH	LGA	229	IAD	4	4
3	DH	LGA	229	IAD	4	5
4	DH	LGA	229	IAD	4	3

In [135…

```
# full-grown tree using training data
X = delays_df3.drop(columns= ['CARRIER', 'DEST', 'ORIGIN'])
y = delays_df3[['CARRIER', 'DEST', 'ORIGIN']]
```

In [136…

```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [137…

```
newDelayTree = DecisionTreeClassifier(max_depth=8, min_samples_split=50, min_impurity_
newDelayTree.fit(train_X, train_y)
```

Out[137]:

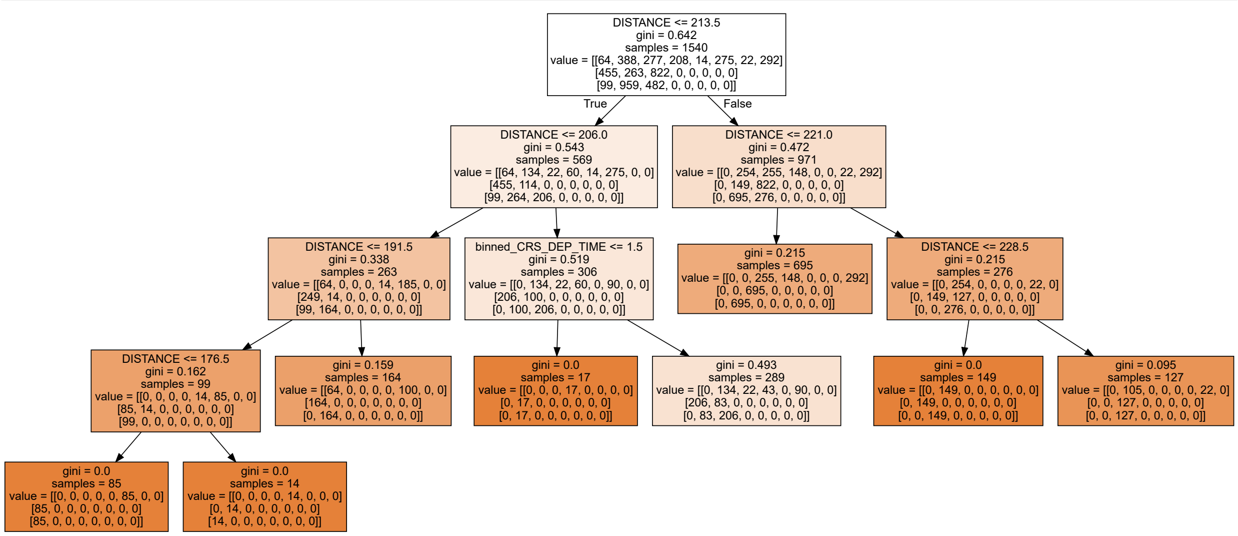
```
DecisionTreeClassifier(max_depth=8, min_impurity_decrease=0.01,
                       min_samples_split=50)
```

In [138…

```
dot_data = tree.export_graphviz(newDelayTree, out_file=None,
                                feature_names=train_X.columns,
                                filled=True)

graph = graphviz.Source(dot_data, format="png")
graph
```

Out[138]:



Examine the full-grown tree. What are the top three predictors according to this tree?

- The top 3 predictors from this tree is distance, day of the week, and destination.

Compare the initial tree with the full grown tree. What do you notice? Speak to the top-level of the fully grown tree as opposed to the small tree.

- I noticed that the produced similar outputs but the weather did not have as big of an impact as expected. I figured it would play a bigger role but it appears that it usually is not a fact baring a huge storm.

Please run a classification summary of the tree and discuss the output.

In [7]:
```
# print the training and test output. Anything you would adjust after seeing the resul
```

- From the results below it appears that there are too many object variables to classifiy if properly. If I had to change anything I would try and include more data that could help me get better results.

In [152…
```
y_train_pred = newDelayTree.predict(train_X)
y_train_pred
```

Out[152]:
```
array([['DH', 'EWR', 'IAD'],
       ['DH', 'EWR', 'IAD'],
       ['DH', 'EWR', 'IAD'],
       ...,
       ['US', 'LGA', 'DCA'],
       ['DH', 'JFK', 'IAD'],
       ['RU', 'EWR', 'BWI']], dtype=object)
```

In [153…
```
cm = confusion_matrix(train_y, y_train_pred)

accuracy = accuracy_score(train_y, y_train_pred)

print("Confusion Matrix:")
print(cm)
print("Accuracy:", accuracy)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [153], in <cell line: 1>()
----> 1 cm = confusion_matrix(train_y, y_train_pred)
      3 accuracy = accuracy_score(train_y, y_train_pred)
      5 print("Confusion Matrix:")

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:307, in confusi
on_matrix(y_true, y_pred, labels, sample_weight, normalize)
    222 def confusion_matrix(
    223     y_true, y_pred, *, labels=None, sample_weight=None, normalize=None
    224 ):
    225     """Compute confusion matrix to evaluate the accuracy of a classification.
    226 
    227     By definition a confusion matrix :math:`C` is such that :math:`C_{i, j}`
   (...)
    305     (0, 2, 1, 1)
    306     """
--> 307     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    308     if y_type not in ("binary", "multiclass"):
    309         raise ValueError("%s is not supported" % y_type)

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:104, in _check_
targets(y_true, y_pred)
    102 # No metrics support "multiclass-multioutput" format
    103 if y_type not in ["binary", "multiclass", "multilabel-indicator"]:
--> 104     raise ValueError("{0} is not supported".format(y_type))
    106 if y_type in ["binary", "multiclass"]:
    107     y_true = column_or_1d(y_true)

ValueError: multiclass-multioutput is not supported
```

In [154…

```
y_pred = newDelayTree.predict(valid_X)
y_pred
```

Out[154]:

```
array([['DH', 'JFK', 'IAD'],
       ['US', 'LGA', 'DCA'],
       ['MQ', 'JFK', 'DCA'],
       ...,
       ['DH', 'EWR', 'IAD'],
       ['US', 'LGA', 'DCA'],
       ['DH', 'EWR', 'IAD']], dtype=object)
```

In [155…

```
cm = confusion_matrix(train_y, y_train_pred)

accuracy = accuracy_score(train_y, y_train_pred)

print("Confusion Matrix:")
print(cm)
print("Accuracy:", accuracy)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [155], in <cell line: 1>()
----> 1 cm = confusion_matrix(train_y, y_train_pred)
      3 accuracy = accuracy_score(train_y, y_train_pred)
      5 print("Confusion Matrix:")

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:307, in confusi
on_matrix(y_true, y_pred, labels, sample_weight, normalize)
    222 def confusion_matrix(
    223     y_true, y_pred, *, labels=None, sample_weight=None, normalize=None
    224 ):
    225     """Compute confusion matrix to evaluate the accuracy of a classification.
    226 
    227     By definition a confusion matrix :math:`C` is such that :math:`C_{i, j}`
   (...)
    305     (0, 2, 1, 1)
    306     """
--> 307     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    308     if y_type not in ("binary", "multiclass"):
    309         raise ValueError("%s is not supported" % y_type)

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:104, in _check_
targets(y_true, y_pred)
    102 # No metrics support "multiclass-multioutput" format
    103 if y_type not in ["binary", "multiclass", "multilabel-indicator"]:
--> 104     raise ValueError("{0} is not supported".format(y_type))
    106 if y_type in ["binary", "multiclass"]:
    107     y_true = column_or_1d(y_true)

ValueError: multiclass-multioutput is not supported
```

In [ ]: