State Farm Classification Coding Exercise

Part 5 - Build a Decision Tree Model

A. Construct Decision Tree Model Upon Training Data

Import numpy and pandas.

In [1]:
```
import numpy as np
import pandas as pd
```

Import data visualization libraries and set %matplotlib inline.

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Import training model data into a Pandas dataframe called train_model1.

In [3]:
```
train_model1 = pd.read_pickle('../State_Farm/Data/train_model.pickle')
```

Check number of rows and columns in train_model1 dataframe.

In [4]:
```
train_model1.shape
```

Out[4]:
```
(40000, 15)
```

View structure of train_model1 dataframe.

In [5]:
```
train_model1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 15 columns):
y          40000 non-null int64
x75        40000 non-null float64
x37        40000 non-null float64
x58        40000 non-null float64
x97        40000 non-null float64
x41_flt    40000 non-null float64
x99        40000 non-null float64
x1         40000 non-null float64
x40        40000 non-null float64
x70        40000 non-null float64
x44        40000 non-null float64
x63        40000 non-null float64
x56        40000 non-null float64
x83        40000 non-null float64
x96        40000 non-null float64
dtypes: float64(14), int64(1)
memory usage: 4.6 MB
```

View first five rows of train_model1 dataframe.

In [6]:
```
train_model1.head()
```

Out[6]:

	y	x75	x37	x58	x97	x41_flt	x99	x1	x40	
0	0	40.617107	-10.839200	2.078396	-2.125570	449.48	1.237667	74.425320	4.550518	41
1	1	-49.303165	57.917006	-2.696257	-36.030599	-525.06	1.952183	24.320711	-9.476135	36
2	1	-19.706659	-12.991058	-2.417447	26.212736	-599.50	0.558988	-66.160459	6.876065	49
3	0	-7.301283	37.658926	4.443710	19.221130	-220.71	0.214462	33.210943	-16.933984	82
4	0	-2.751656	-59.497091	-2.421952	-5.703269	-1405.59	-1.191319	-26.717872	-9.551514	-19

Gather summary statistics of train_model1 dataframe.

In [7]:
```
train_model1.describe()
```

Out[7]:

	y	x75	x37	x58	x97	x41_flt	
count	40000.00000	40000.000000	40000.000000	40000.000000	40000.000000	40000.000000	40000.0
mean	0.20360	5.417519	4.814694	-0.892583	-2.514305	0.315315	0.0
std	0.40268	35.653631	31.561154	6.319470	18.551630	1001.659950	1.1
min	0.00000	-146.967384	-127.651997	-31.395877	-73.908741	-4496.460000	-4.3
25%	0.00000	-17.971049	-16.589036	-5.175537	-15.023076	-669.295000	-0.7
50%	0.00000	5.592658	4.493183	-0.891371	-2.514305	6.040000	0.0
75%	0.00000	28.988799	25.968530	3.413382	9.882079	678.325000	0.7
max	1.00000	144.548014	126.924294	22.420511	76.120119	4062.630000	4.4

Define X and y.

In [8]:
```
X = train_model1.drop(['y'], axis=1)
y = train_model1['y']
print(X.shape)
print(y.shape)
```

```
(40000, 14)
(40000,)
```

Tune decision tree model to avoid overfitting.

In [9]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import cross_val_score
from sklearn import metrics
```

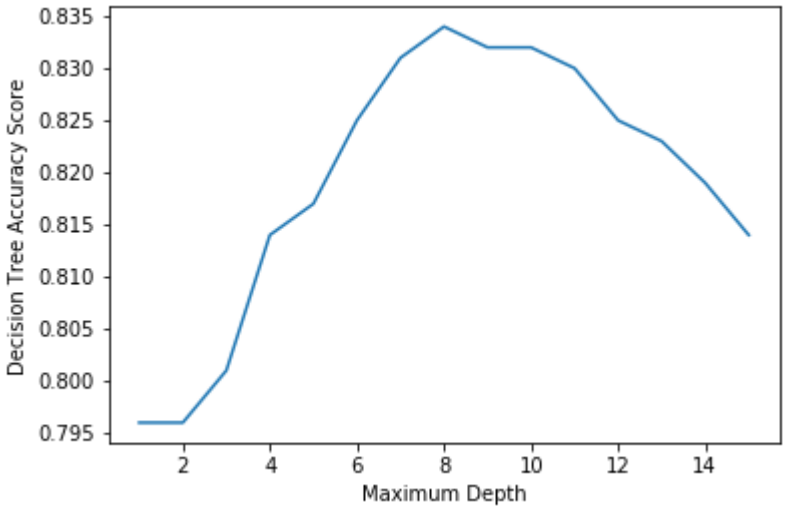
```
C:\Users\kyrma\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\cros
s_validation.py:41: DeprecationWarning: This module was deprecated in version
0.18 in favor of the model_selection module into which all the refactored class
es and functions are moved. Also note that the interface of the new CV iterator
s are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [10]:
```
max_depth_range1 = range(1, 16)

acc_scores1 = []

for depth in max_depth_range1:
    treereg = DecisionTreeClassifier(max_depth=depth, random_state=1)
    scores1 = cross_val_score(treereg, X, y, cv=14, scoring='accuracy')
    acc_scores1.append(scores1.mean().round(3))

print(acc_scores1)
```

```
[0.796, 0.796, 0.801, 0.814, 0.817, 0.825, 0.831, 0.834, 0.832, 0.832, 0.83, 0.
825, 0.823, 0.819, 0.814]
```

In [11]:
```
plt.plot(max_depth_range1, acc_scores1)
plt.xlabel('Maximum Depth')
plt.ylabel('Decision Tree Accuracy Score')
```

Out[11]:
```
Text(0,0.5,'Decision Tree Accuracy Score')
```



- The maximum depth of the decision tree should be 8 since it has a maximum accuracy score of 0.834.

Fit a decision tree with maximum depth of 8 on training model data.

In [12]:
```
treereg_train = DecisionTreeClassifier(max_depth=8, random_state=1)
treereg_train.fit(X, y)
```

Out[12]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=8,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best')
```

Compute training decision tree model feature importances.

In [13]:
```
pd.DataFrame({'feature':X.columns, 'importance':treereg_train.feature_importances_
```

Out[13]:

	feature	importance
0	x75	0.136632
3	x97	0.122258
5	x99	0.101910
9	x44	0.098431
2	x58	0.093464
1	x37	0.091063
12	x83	0.070051
4	x41_flt	0.061514
10	x63	0.049660
6	x1	0.043919
7	x40	0.043811
11	x56	0.033076
13	x96	0.030648
8	x70	0.023562

Make predictions on training model data and calculate accuracy.

In [14]:
```
y_pred_class = treereg_train.predict(X)
print(metrics.accuracy_score(y, y_pred_class))
```

```
0.8594
```

Compute null accuracy manually.

In [15]:
```
print(1 - y.mean())
```

```
0.7964
```

Calculate area under the ROC curve (AUC) value for decision tree model fitted on training model data.

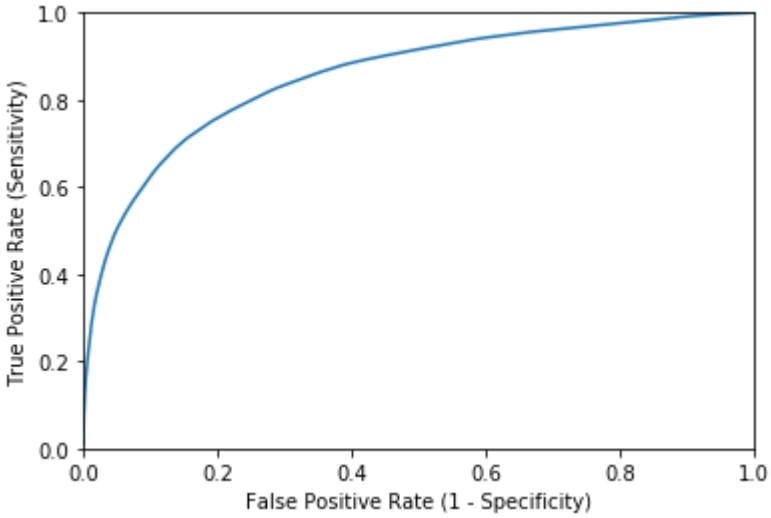
In [16]:
```
y_pred_prob = treereg_train.predict_proba(X)[:, 1]
print(metrics.roc_auc_score(y, y_pred_prob))
```

```
0.8556623532103949
```

Plot decision tree model ROC curve.

In [17]:
```
thresholds = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
fpr, tpr, thresholds = metrics.roc_curve(y, y_pred_prob)
plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
```

Out[17]:
```
Text(0,0.5,'True Positive Rate (Sensitivity)')
```



Create GraphViz file of decision tree.

In [18]:
```
from sklearn.tree import export_graphviz
from IPython.display import Image
from IPython.display import display
```

In [19]:
```
feature_cols = ['x75', 'x37', 'x58', 'x97', 'x41_flt', 'x99', 'x1', 'x40', 'x70',
export_graphviz(treereg_train, out_file='State_Farm_Classification_Decision_Tree.
```

- To display the decision tree, copy the text from the State_Farm_Classification_Decision_Tree.dot file and paste it into the text box at http://webgraphviz.com/ (http://webgraphviz.com/).

Print confusion matrix to calculate training model data accuracy and error rates plus precision and recall.

In [20]:
```
print(metrics.confusion_matrix(y, y_pred_class))
```

```
[[30456  1400]
 [ 4224  3920]]
```

Calculate training model data accuracy rate.

In [21]:
```
float(30456 + 3920) / float(30456 + 1400 + 4224 + 3920)
```

Out[21]: 0.8594

Calculate training model data misclassification / error rate.

In [22]:
```
float(4224 + 1400) / float(30456 + 1400 + 4224 + 3920)
```

Out[22]: 0.1406

Calculate precision to measure how confident the decision tree model is for capturing the positives in training model data.

In [23]:
```
float(3920) / float(3920 + 1400)
```

Out[23]: 0.7368421052631579

Calculate recall / sensitivity to measure how well the decision tree model is capturing the positives in training model data.

In [24]:
```
float(3920) / float(4224 + 3920)
```

Out[24]: 0.481335952848723

Calculate specificity to measure how well the decision tree model is capturing the negatives in training model data.

In [25]:
```
float(30456) / float(30456 + 1400)
```

Out[25]: 0.9560522350577599

Print out training model data classification report for decision tree model.

In [26]:
```
from sklearn.metrics import classification_report
```

In [27]:
```
print(classification_report(y, y_pred_class))
```

```
             precision    recall  f1-score   support

          0       0.88      0.96      0.92     31856
          1       0.74      0.48      0.58      8144

avg / total       0.85      0.86      0.85     40000
```

List out training model data decision tree model accuracy scores and their average using 10-fold cross-validation.

In [28]:
```
from sklearn.cross_validation import cross_val_score
```

In [29]:
```
acc_scores2 = cross_val_score(treereg_train, X, y, cv=10, scoring='accuracy').rou
print(acc_scores2)
print(acc_scores2.mean().round(3))
```

```
[0.83  0.828 0.836 0.83  0.834 0.836 0.836 0.831 0.835 0.835]
0.833
```

List out training model data decision tree model AUC values and their average using 10-fold cross-validation.

In [30]:
```
auc_scores1 = cross_val_score(treereg_train, X, y, cv=10, scoring='roc_auc').roun
print(auc_scores1)
print(auc_scores1.mean().round(3))
```

```
[0.791 0.797 0.797 0.797 0.799 0.798 0.806 0.799 0.816 0.813]
0.801
```

B. Generate Predictions in Cleaned Test Data

Import cleaned test data pickle file into a Pandas dataframe called test_model1.

In [31]:
```
test_model1 = pd.read_pickle('../State_Farm/Data/test_model.pickle')
```

Check number of rows and columns in test_model1 dataframe.

In [32]:
```
test_model1.shape
```

Out[32]:
```
(10000, 14)
```

View structure of test_model1 dataframe.

In [33]:
```
test_model1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
x75        10000 non-null float64
x37        10000 non-null float64
x58        10000 non-null float64
x97        10000 non-null float64
x41_flt    10000 non-null float64
x99        10000 non-null float64
x1         10000 non-null float64
x40        10000 non-null float64
x70        10000 non-null float64
x44        10000 non-null float64
x63        10000 non-null float64
x56        10000 non-null float64
x83        10000 non-null float64
x96        10000 non-null float64
dtypes: float64(14)
memory usage: 1.1 MB
```

View first five rows of test_model1 dataframe.

In [34]:
```
test_model1.head()
```

Out[34]:

	x75	x37	x58	x97	x41_flt	x99	x1	x40	
0	80.851384	-32.086998	5.093232	13.739194	2475.46	1.141799	54.479467	-8.776231	44.269
1	-21.295879	29.391786	-5.989844	-13.018951	-1109.10	0.568757	-20.244923	23.337240	5.663
2	27.719905	-30.329997	6.115105	0.791425	-187.70	-0.816682	-61.467354	-8.845933	-47.647
3	-4.053955	11.088216	-2.750484	-16.716012	525.65	0.603007	-18.454831	-57.516611	-14.010
4	21.743536	-21.955105	-4.286183	23.003355	-1113.53	1.929231	15.810515	-13.207037	13.267

Define X_test.

In [35]:
```
X_test = test_model1[feature_cols]
print(X_test.shape)
```

```
(10000, 14)
```

Make predictions on cleaned test data by calculating probabilities for belonging to positive class (labeled '1').

In [36]:
```
y_test_pred_prob = treereg_train.predict_proba(X_test)[:, 1]
```

Construct results2_dtree Pandas dataframe out of test data y predicted class probabilities.

In [37]:
```
results2_dtree = pd.DataFrame({'y_test_pred_prob': y_test_pred_prob[:]})
results2_dtree.head()
```

Out[37]:

	y_test_pred_prob
0	0.163598
1	0.423729
2	0.296970
3	0.070088
4	0.359807

Check number of rows and columns in results2_dtree dataframe.

In [38]:
```
results2_dtree.shape
```

Out[38]: (10000, 1)

Export results2_dtree Pandas dataframe containing test data y predicted class probabilities to CSV file.

In [39]:
```
results2_dtree.to_csv('../State_Farm/Data/results2_dtree.csv', sep=',', index=Fal
```