credit_card_default

June 28, 2022

1 Default of Credit Card Clients Data Set

1.1 Data Set Information

The data set consists of customers credit card information and payments over a 6 month period. This data is used to compare predictive accuracy of multiple machine learning algorithms aimed at predicting if customers would default on thier credit card payments. This type of analysis would be important for businesses to forcasting revenue and the impact on revenue that are caused by clients defaulting on thier creditcards.

Abstract	Description
Data Set Characteristics:	Multivariate
Number of Instances:	30000
Number of Attributes:	24
Associated Tasks:	Classification
Date Donated:	2016-01-26

1.1.1 Attribute Information

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:

X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit. X2: Gender (1 = male; 2 = female). X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others). X4: Marital status (1 = married; 2 = single; 3 = others). X5: Age (year). X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above. X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005. X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

1.1.2 Citation

https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

1.1.3 Imported Libararies and Packages

[1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import seaborn as sns
from scipy.stats import norm
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, GridSearchCV

#graphing tree
from sklearn import tree
from sklearn.tree import export_graphviz
# import graphviz
from six import StringIO
from IPython.display import Image
import pydotplus


# Machine Learning Models
from sklearn.linear_model import LogisticRegression as LR
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.ensemble import AdaBoostClassifier

# Metrics
from sklearn.metrics import confusion_matrix,  roc_auc_score, roc_curve,␣
↪precision_score, recall_score
```

1.2 Part 1: Load, Inspection and Cleaning of Data

[2]:
```
# Load the data
df = pd.read_csv("cc_default_data.csv")
# check shape of the dataframe
df.shape
```

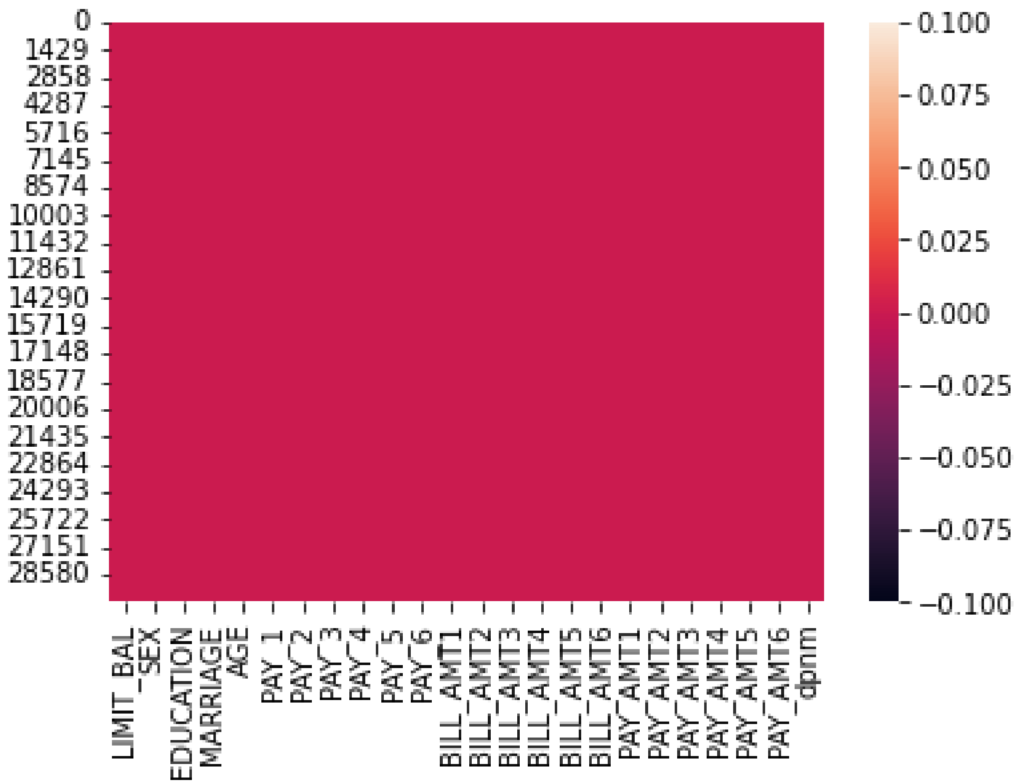
[2]:
```
(30000, 25)
```

[3]:
```
# inspect the first 5 rows of the data
df.head()
```

[3]:
```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_1  PAY_2  PAY_3  PAY_4  \
0   1      20000    2          2         1   24      2      2     -1     -1
1   2     120000    2          2         2   26     -1      2      0      0
2   3      90000    2          2         2   34      0      0      0      0
3   4      50000    2          2         1   37      0      0      0      0
4   5      50000    1          2         1   57     -1      0     -1      0

   …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0  …          0          0          0         0       689         0
1  …       3272       3455       3261         0      1000      1000
2  …      14331      14948      15549      1518      1500      1000
3  …      28314      28959      29547      2000      2019      1200
4  …      20940      19146      19131      2000     36681     10000

   PAY_AMT4  PAY_AMT5  PAY_AMT6  dpnm
0         0         0         0     1
1      1000         0      2000     1
2      1000      1000      5000     0
3      1100      1069      1000     0
4      9000       689       679     0

[5 rows x 25 columns]
```

[4]:
```
# Drop ID column.
df.drop('ID', axis=1, inplace=True)
```

[5]:
```
# Inspect if there are any null values in the dataset, by using a heatmap
sns.heatmap(df.isnull());
```



[6]:
```
# Double check on the null values
null_values = pd.DataFrame(df.isnull().sum())
null_values
```

[6]:

	0
LIMIT_BAL	0
SEX	0
EDUCATION	0
MARRIAGE	0
AGE	0
PAY_1	0
PAY_2	0
PAY_3	0
PAY_4	0
PAY_5	0
PAY_6	0
BILL_AMT1	0
BILL_AMT2	0
BILL_AMT3	0
BILL_AMT4	0
BILL_AMT5	0
BILL_AMT6	0

```
PAY_AMT1    0
PAY_AMT2    0
PAY_AMT3    0
PAY_AMT4    0
PAY_AMT5    0
PAY_AMT6    0
dpnm        0
```

[55]:

```
# Check for duplicated data
print("Duplicated rows: ", df.duplicated().sum())

# Remove Duplicated Data

print("Dataframe rows before removing duplicates: ", df.shape[0])
df = df.drop_duplicates()
print("Dataframe rows after removing duplicates: ", df.shape[0])
```

```
Duplicated rows:  0
Dataframe rows before removing duplicates:  29965
Dataframe rows after removing duplicates:  29965
```

[56]:

```
# inspect the data types of the each feature, to ensure that we dont need to␣
↪transform the data in anyway.
print(df.dtypes)
```

```
LIMIT_BAL    int64
SEX          int64
EDUCATION    int64
MARRIAGE     int64
AGE          int64
PAY_1        int64
PAY_2        int64
PAY_3        int64
PAY_4        int64
PAY_5        int64
PAY_6        int64
BILL_AMT1    int64
BILL_AMT2    int64
BILL_AMT3    int64
BILL_AMT4    int64
BILL_AMT5    int64
BILL_AMT6    int64
PAY_AMT1     int64
PAY_AMT2     int64
PAY_AMT3     int64
PAY_AMT4     int64
PAY_AMT5     int64
```

```
PAY_AMT6     int64
dpnm         int64
dtype: object
```

[7]:
```
#checking the percentage of defaulted subjects.
df['dpnm'].value_counts() / df.shape[0]
```

[7]:
```
0    0.7788
1    0.2212
Name: dpnm, dtype: float64
```

[57]:
```
#finding out the different values of each feature. There are features that␣
↪appear in the data that are not accurately labeled in the directions.
columns = ['SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1', 'PAY_2', 'PAY_3',␣
↪'PAY_4', 'PAY_5', 'PAY_6']

for feature in columns:
    print(f'{df[feature].value_counts()} \n');
```

```
2    18091
1    11874
Name: SEX, dtype: int64

2    14019
1    10563
3     4915
5      280
4      123
6       51
0       14
Name: EDUCATION, dtype: int64

2    15945
1    13643
3      323
0       54
Name: MARRIAGE, dtype: int64

29    1602
27    1475
28    1406
30    1394
26    1252
31    1213
25    1185
34    1161
32    1157
33    1146
```

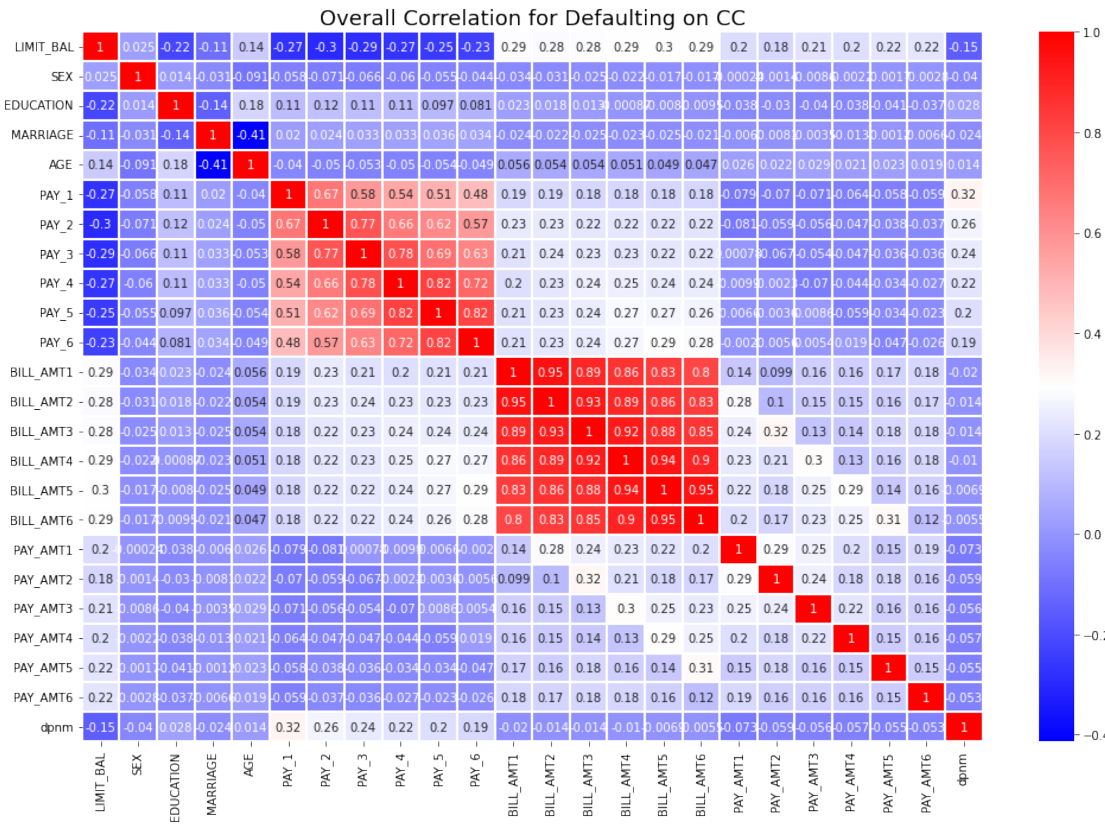
```
24    1126
35    1113
36    1107
37    1041
39     951
38     943
23     930
40     870
41     822
42     792
44     700
43     669
45     617
46     570
22     560
47     501
48     466
49     449
50     411
51     340
53     325
52     304
54     247
55     209
56     178
58     122
57     122
59      83
60      67
21      67
61      56
62      44
63      31
64      31
66      25
65      24
67      16
69      15
70      10
68       5
73       4
72       3
75       3
71       3
79       1
74       1
Name: AGE, dtype: int64
```

```
 0    14737
-1     5682
 1     3667
-2     2750
 2     2666
 3      322
 4       76
 5       26
 8       19
 6       11
 7        9
Name: PAY_1, dtype: int64

 0    15730
-1     6046
 2     3926
-2     3752
 3      326
 4       99
 1       28
 5       25
 7       20
 6       12
 8        1
Name: PAY_2, dtype: int64

 0    15764
-1     5934
-2     4055
 2     3819
 3      240
 4       75
 7       27
 6       23
 5       21
 1        4
 8        3
Name: PAY_3, dtype: int64

 0    16455
-1     5683
-2     4318
 2     3159
 3      180
 4       68
 7       58
 5       35
 6        5
```

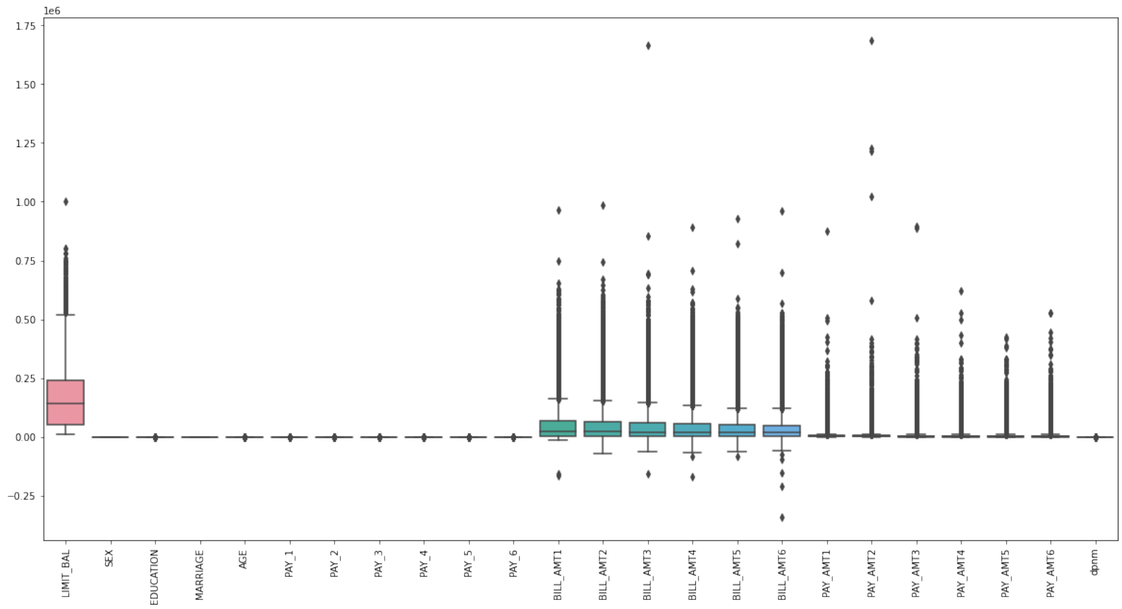
```
 1        2
 8        2
Name: PAY_4, dtype: int64

 0    16947
-1     5535
-2     4516
 2     2626
 3      178
 4       83
 7       58
 5       17
 6        4
 8        1
Name: PAY_5, dtype: int64

 0    16286
-1     5736
-2     4865
 2     2766
 3      184
 4       48
 7       46
 6       19
 5       13
 8        2
Name: PAY_6, dtype: int64
```

[58]:

```
# Correlation Matrix of all the features
corr = df.corr()
plt.figure(figsize = (17,11))
plt.title('Overall Correlation for Defaulting on CC', fontsize=18)
sns.heatmap(corr,annot=True,cmap='bwr', linewidths=.02)
plt.show();
```



[59]:
```
#box plot of all features
plt.figure(figsize=(20,10));
sns.boxplot(data=df);
plt.xticks(rotation = 90);
```



[12]:
```
# dropped the features that were highly correlated. I decided to keep PAY_1,␣
↪BILL_AMT1 because they were the most recent features in the dataset.
drop_list = ['PAY_2','PAY_3','PAY_4','PAY_5','PAY_6', 'BILL_AMT2', 'BILL_AMT3',␣
↪'BILL_AMT4', 'BILL_AMT5','BILL_AMT6']

for col in drop_list:
    df.drop(col , axis=1, inplace=True)
```

1.3 Part2: Pre-Processing

[13]:
```
#creating test and train data for ML
X = df.drop(columns=['dpnm'], axis=1)
y = df.dpnm
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.25,␣
↪random_state=0)

print(f' train shape: {X_train.shape}\ntest shape: {y_train.shape}')
```

```
 train shape: (22473, 23)
test shape: (22473,)
```

[14]:
```
# Standardization
# Standardize each feature to ensure more accurate
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

1.4 Part3: ML Model Building

ML Models to be use:

Logistic Regression Logistic Regression W/ GridSearch Decision Tree Decision Tree W/ GridSearch Random Forest Random Forest W/ GridSearch Adaptive Boost with Decision Tree

I chose these models because they were all proficient with classifying binary data sets.

1.4.1 Logistic Regression

[15]:
```
LogReg = LR(solver = 'newton-cg')
LogReg.fit(X_train,y_train)
```

[15]:
```
LogisticRegression(solver='newton-cg')
```

[60]:
```
logreg_pp = LogReg.predict_log_proba(X_test)
logreg_y_pred = LogReg.predict(X_test)
```

Confusion Matrix

[61]:
```
pd.DataFrame(confusion_matrix(y_test, logreg_y_pred, labels=[0,1]))
```

[61]:

	0	1
0	5694	150
1	1262	386

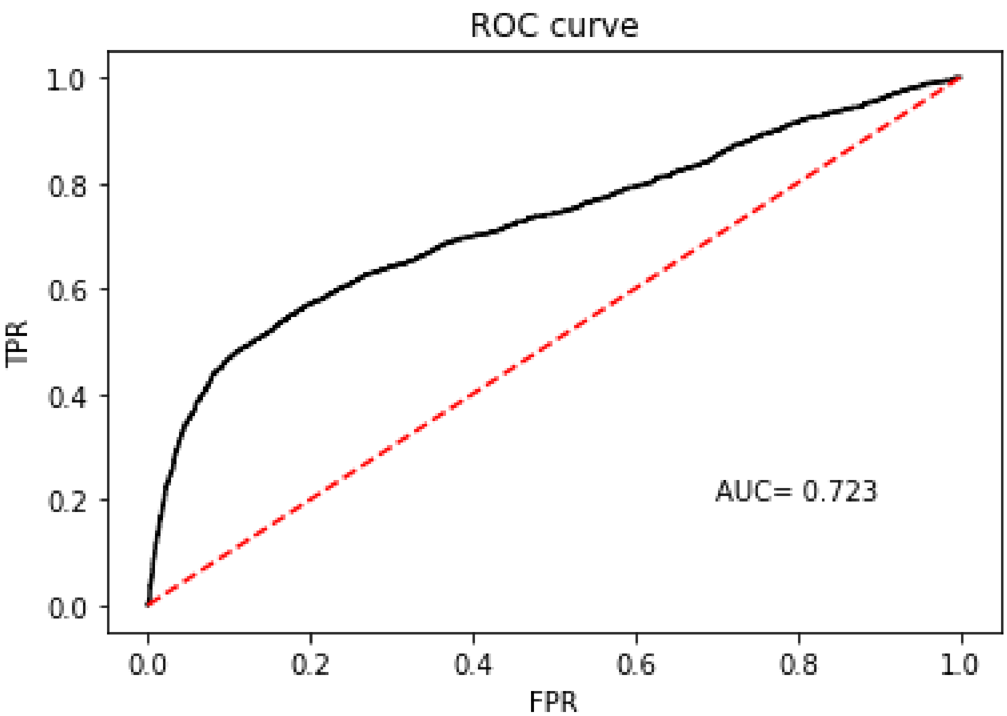
Logistic Regression Metrics

[18]:
```
logreg_acc = LogReg.score(X_test,y_test)
logreg_prec = precision_score(y_test, logreg_y_pred)
logreg_recall = recall_score(y_test, logreg_y_pred)
logreg_cv_score = cross_val_score(LogReg, X_train, y_train, cv=10).mean()
print("Logistic Regression Prediction Results: ")
# Accuracy:
print("Accuracy: {:.3f}".format(logreg_acc))

#Cross Validation:
print("Cross Validation Accuracy: {:.3f}".format(logreg_cv_score))

# Precision
print('Precision: {:.3f}'.format(logreg_prec))

# Recall
print('Recall: {:.3f}'.format(logreg_recall))
```

```
Logistic Regression Prediction Results:
Accuracy: 0.812
Cross Validation Accuracy: 0.810
```

```
Precision: 0.720
Recall: 0.234
```

[19]:
```
fpr,tpr,th = roc_curve(y_test, logreg_pp[:,1])
lg_auc = roc_auc_score(y_test, logreg_pp[:,1])
plt.plot(fpr,tpr, 'k-')
plt.plot(np.arange(0,1.1,0.1), np.arange(0,1.1,0.1), 'r--')
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.text(0.7,0.2, 'AUC= ' + "{:.3f}".format(lg_auc));
```



1.4.2 Logistic Regression w/ GridSearch

[20]:
```
params = { 'penalty': ['l2','none'], 'C': [ 0.001, 0.01, 0.1, 1, 10, 32, 100,␣
 ↪200], 'solver': ['newton-cg', 'lbfgs', 'sag', 'saga']}
lr = LR(multi_class='auto', random_state=25, n_jobs=-1)
LogReg_GS = GridSearchCV(lr,params,cv=10, n_jobs=-1, verbose=1)
LogReg_GS.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 64 candidates, totalling 640 fits
```

[20]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(n_jobs=-1, random_state=25),
             n_jobs=-1,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 32, 100, 200],
                         'penalty': ['l2', 'none'],
                         'solver': ['newton-cg', 'lbfgs', 'sag', 'saga']},
             verbose=1)
```

[21]:
```
logreg_pp_gs = LogReg_GS.predict_proba(X_test)
logreg_y_pred_gs = LogReg_GS.predict(X_test)
logreg_gs_cross_validation = LogReg_GS.best_score_
logreg_gs_prec = precision_score(y_test,logreg_y_pred_gs)
logreg_gs_acc = LogReg_GS.score(X_test,y_test)
logreg_gs_recall = recall_score(y_test,logreg_y_pred_gs)
```

Confusion Matrix

[22]:
```
pd.DataFrame(confusion_matrix(y_test, logreg_y_pred_gs, labels=[0,1]))
```

[22]:

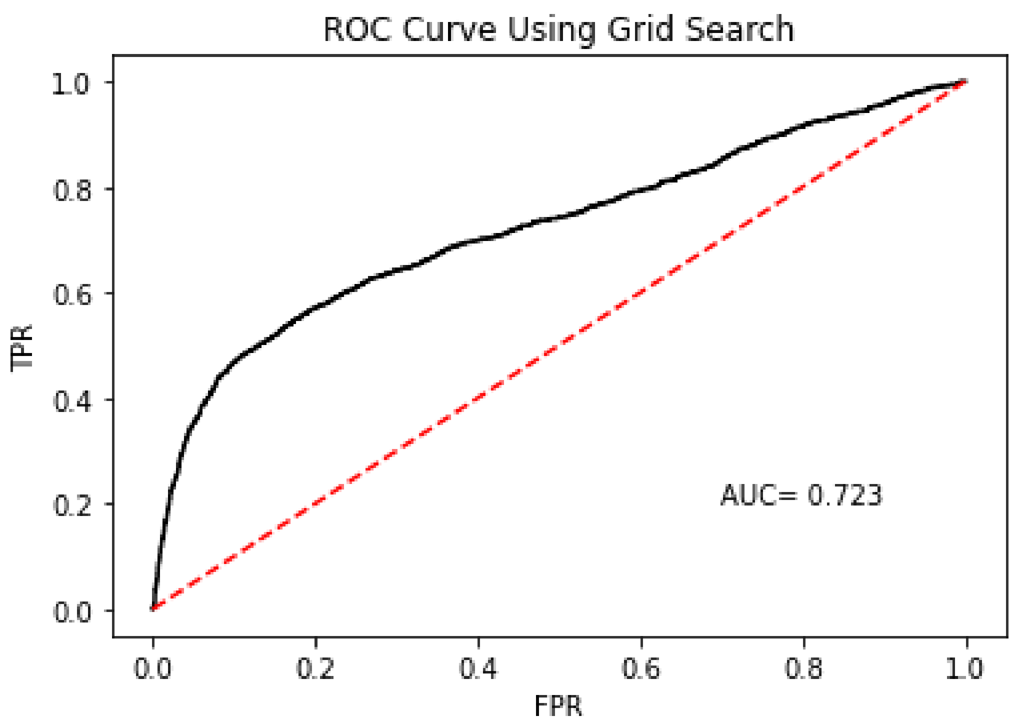
	0	1
0	5694	150
1	1262	386

Logistic Regression w/ Gridsearch Metrics

[23]:
```
print("GridSearch Logistic Regression Prediction Results: ")

#Best Predictors
print("Best Logistic Regression Parameters: {}".format(LogReg_GS.best_params_))

#Cross Validation Score
print("Cross Validation Accuracy: {:.3f}".format(logreg_gs_cross_validation))
# Accuracy:
print("Accuracy: {:.3f}".format(logreg_gs_acc))

# Precision
print('Precision: {:.3f}'.format(logreg_gs_prec))

# Recall
print('Recall: {:.3f}'.format(logreg_gs_recall))
```

```
GridSearch Logistic Regression Prediction Results:
Best Logistic Regression Parameters: {'C': 1, 'penalty': 'l2', 'solver':
'newton-cg'}
Cross Validation Accuracy: 0.810
Accuracy: 0.812
Precision: 0.720
Recall: 0.234
```

[24]:
```
fpr,tpr,th = roc_curve(y_test, logreg_pp_gs[:,1])
lg_gs_auc = roc_auc_score(y_test, logreg_pp_gs[:,1])
plt.plot(fpr,tpr, 'k-')
plt.plot(np.arange(0,1.1,0.1), np.arange(0,1.1,0.1), 'r--')
plt.title('ROC Curve Using Grid Search')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.text(0.7,0.2, 'AUC= ' + "{:.3f}".format(lg_gs_auc));
```



1.4.3 Decision Tree

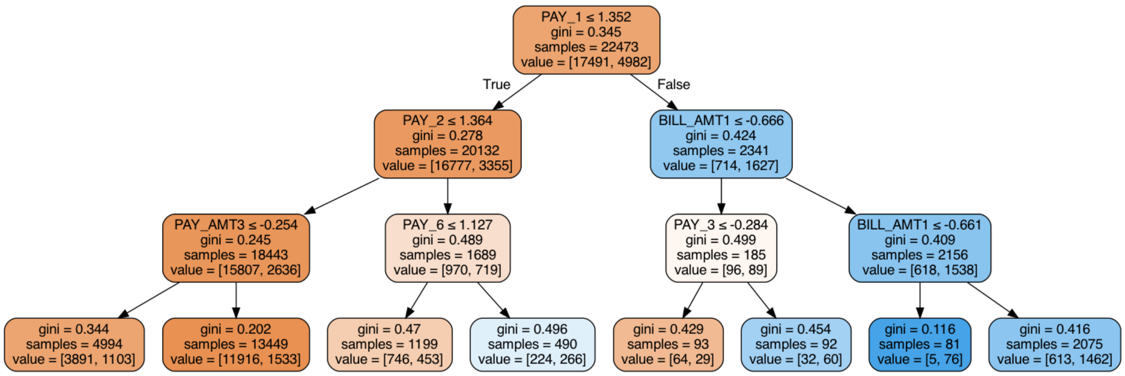
[25]:
```
dtc = DTC(criterion='gini', max_depth=3,  random_state=25)
dtc.fit(X_train, y_train)
```

[25]: DecisionTreeClassifier(max_depth=3, random_state=25)

Decision Tree Graph

[26]:
```
dot_data = StringIO()
export_graphviz(dtc, out_file=dot_data, filled=True,feature_names=X.columns,␣
 ↪rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

[26]:



[27]:
```
pp_dtc = dtc.predict_proba(X_test)
y_pred_dtc = dtc.predict(X_test)
dtc_acc = dtc.score(X_test,y_test)
dtc_prec = precision_score(y_test, y_pred_dtc)
dtc_recall = recall_score(y_test, y_pred_dtc)
dtc_cv_score = cross_val_score(dtc, X_train, y_train, cv=10).mean()
```

Confusion Matrix

[28]:
```
pd.DataFrame(confusion_matrix(y_test, y_pred_dtc, labels=[0,1]))
```

[28]:

	0	1
0	5538	306
1	1039	609

Decision Tree Metrics

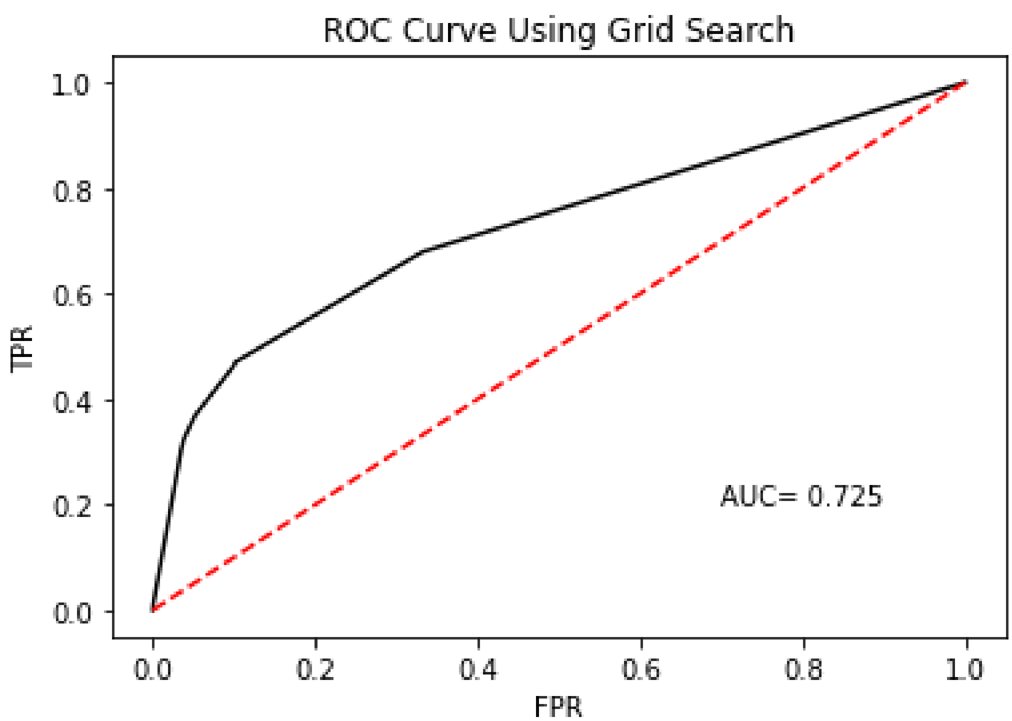
[29]:
```
print("Decision Tree Prediction Results: ")

#Cross Validation Score
print("Cross Validation Accuracy: {:.3f}".format(dtc_cv_score))

# Accuracy:
print("Accuracy: {:.3f}".format(dtc_acc))

# Precision
print('Precision: {:.3f}'.format(dtc_prec))

# Recall
print('Recall: {:.3f}'.format(dtc_recall))
```

```
Decision Tree Prediction Results:
Cross Validation Accuracy: 0.821
Accuracy: 0.820
```

```
Precision: 0.666
Recall: 0.370
```

[30]:

```
fpr,tpr,th = roc_curve(y_test, pp_dtc[:,1])
dtc_auc = roc_auc_score(y_test,pp_dtc[:,1])
plt.plot(fpr,tpr, 'k-')
plt.plot(np.arange(0,1.1,0.1), np.arange(0,1.1,0.1), 'r--')
plt.title('ROC Curve Using Grid Search')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.text(0.7,0.2, 'AUC= ' + "{:.3f}".format(dtc_auc));
```



1.4.4 Decision Tree w/ GridSearch

[31]:

```
params = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [ 2, 5, 8, 10, 15],
    'max_leaf_nodes': [ 2, 5, 8, 10, 16],
    'min_samples_leaf': [1,2,4,10,20]
}

dtc = DTC()
DTC_GS = GridSearchCV(dtc,params, cv=10, n_jobs=-1, verbose=1)
```

```
DTC_GS.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 375 candidates, totalling 3750 fits
```

[31]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [2, 5, 8, 10, 15],
                         'max_leaf_nodes': [2, 5, 8, 10, 16],
                         'min_samples_leaf': [1, 2, 4, 10, 20]},
             verbose=1)
```

[32]:
```
pp_dtc_gs = DTC_GS.predict_proba(X_test)
y_pred_dtc_gs = DTC_GS.predict(X_test)
dtc_gs_acc = DTC_GS.score(X_test,y_test)
dtc_gs_prec = precision_score(y_test, y_pred_dtc_gs)
dtc_gs_recall = recall_score(y_test, y_pred_dtc_gs)
dtc_gs_cross_validation = DTC_GS.best_score_
```

Confusion Matrix

[33]:
```
pd.DataFrame(confusion_matrix(y_test, y_pred_dtc_gs, labels=[0,1]))
```

[33]:

	0	1
0	5538	306
1	1039	609

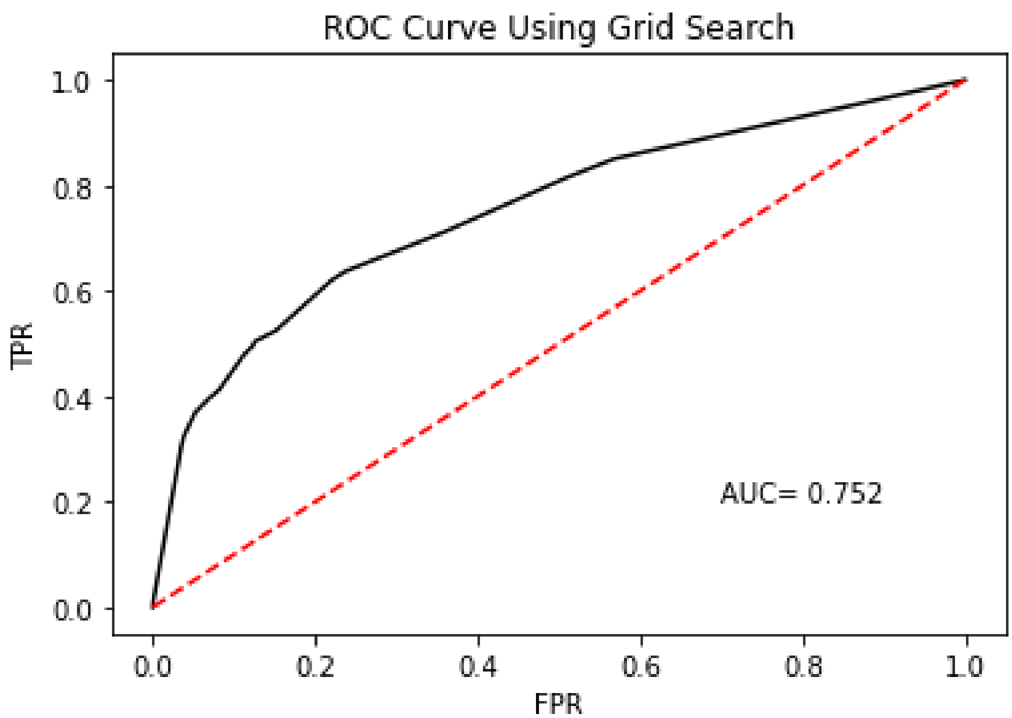
Decision Tree w/ GridSearch Metrics

[34]:
```
print("GridSearch DTC Prediction Results: ")

#Best Predictors
print("Best DTC Parameters: {}".format(DTC_GS.best_params_))

#Cross Validation Score
print("Cross Validation Accuracy: {:.3f}".format(dtc_gs_cross_validation))
# Accuracy:
print("Accuracy: {:.3f}".format(dtc_gs_acc))

# Precision
print('Precision: {:.3f}'.format(dtc_gs_prec))

# Recall
print('Recall: {:.3f}'.format(dtc_gs_recall))
```

```
GridSearch DTC Prediction Results:
Best DTC Parameters: {'criterion': 'gini', 'max_depth': 8, 'max_leaf_nodes': 16,
'min_samples_leaf': 20}
Cross Validation Accuracy: 0.821
```

```
Accuracy: 0.820
Precision: 0.666
Recall: 0.370
```

[35]:
```
fpr,tpr,th = roc_curve(y_test, pp_dtc_gs[:,1])
dtc_gs_auc = roc_auc_score(y_test, pp_dtc_gs[:,1])
plt.plot(fpr,tpr, 'k-')
plt.plot(np.arange(0,1.1,0.1), np.arange(0,1.1,0.1), 'r--')
plt.title('ROC Curve Using Grid Search')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.text(0.7,0.2, 'AUC= ' + "{:.3f}".format(dtc_gs_auc));
```



1.4.5 Random Forest Decision Tree

[36]:
```
rfc = RFC(n_estimators=20)
rfc.fit(X_train, y_train)
```

[36]: `RandomForestClassifier(n_estimators=20)`

[37]:
```
pp_rfc = rfc.predict_proba(X_test)
y_pred_rfc = rfc.predict(X_test)
rfc_acc = rfc.score(X_test,y_test)
rfc_prec = precision_score(y_test, y_pred_rfc)
```

```
rfc_recall = recall_score(y_test, y_pred_rfc)
rfc_cv_score = cross_val_score(rfc, X_train, y_train, cv=10).mean()
```

Confusion Matrix

[38]:
```
pd.DataFrame(confusion_matrix(y_test, y_pred_rfc, labels=[0,1]))
```

[38]:

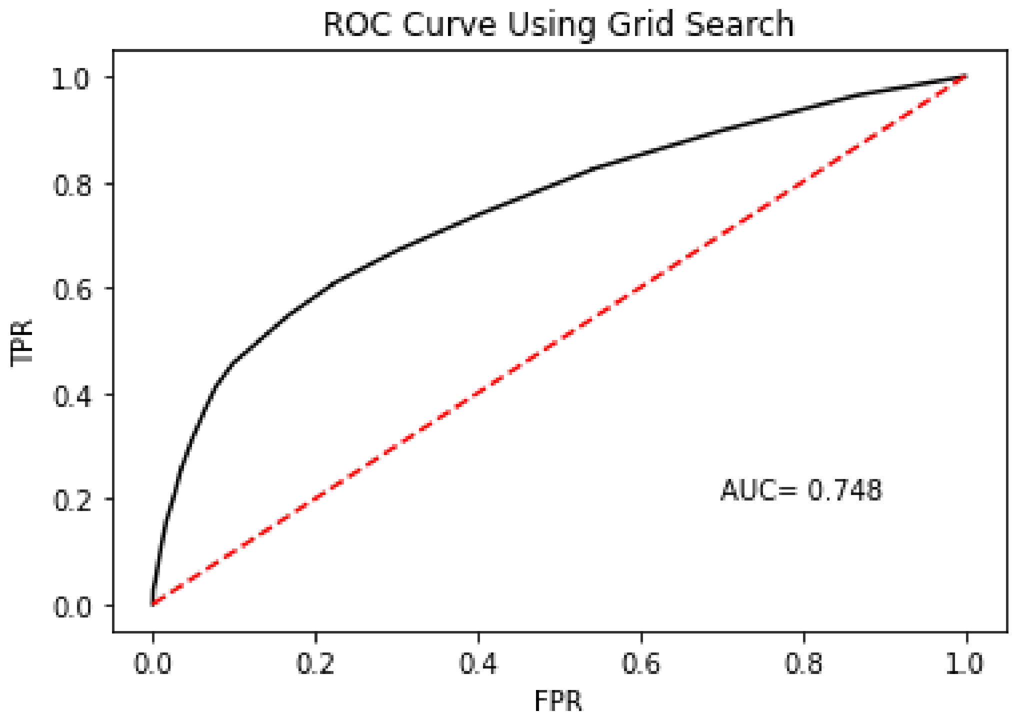
	0	1
0	5481	363
1	1052	596

Random Forest Metrics

[39]:
```
print("Random Forest Prediction Results: ")
#Cross Validation Score
print("Cross Validation Accuracy: {:.3f}".format(rfc_cv_score))
# Accuracy:
print("Accuracy: {:.3f}".format(rfc_acc))

# Precision
print('Precision: {:.3f}'.format(rfc_prec))

# Recall
print('Recall: {:.3f}'.format(rfc_recall))
```

```
Random Forest Prediction Results:
Cross Validation Accuracy: 0.812
Accuracy: 0.811
Precision: 0.621
Recall: 0.362
```

[40]:
```
fpr,tpr,th = roc_curve(y_test, pp_rfc[:,1])
rfc_auc = roc_auc_score(y_test, pp_rfc[:,1])
plt.plot(fpr,tpr, 'k-')
plt.plot(np.arange(0,1.1,0.1), np.arange(0,1.1,0.1), 'r--')
plt.title('ROC Curve Using Grid Search')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.text(0.7,0.2, 'AUC= ' + "{:.3f}".format(rfc_auc));
```



1.4.6 Random Forest w/ GridSearch

[41]:
```
params = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [ 2, 5, 8, 10, 15],
    'max_leaf_nodes': [ 2, 5, 8, 10, 15],
    'n_estimators': [4, 6, 8, 10]
}

rfc = RFC()
RFC_GS = GridSearchCV(rfc,params, cv=10, verbose=1)
RFC_GS.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 300 candidates, totalling 3000 fits
```

[41]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [2, 5, 8, 10, 15],
                         'max_leaf_nodes': [2, 5, 8, 10, 15],
                         'n_estimators': [4, 6, 8, 10]},
             verbose=1)
```

[42]:
```
pp_rfcgs = RFC_GS.predict_proba(X_test)
y_pred_rfcgs = RFC_GS.predict(X_test)
rfc_gs_acc = RFC_GS.score(X_test,y_test)
rfc_gs_prec = precision_score(y_test, y_pred_rfcgs)
rfc_gs_recall = recall_score(y_test, y_pred_rfcgs)
rfc_gs_cross_validation = RFC_GS.best_score_
```

Confusion Matrix

[43]:
```
pd.DataFrame(confusion_matrix(y_test, y_pred_rfcgs, labels=[0,1]))
```

[43]:

	0	1
0	5609	235
1	1138	510

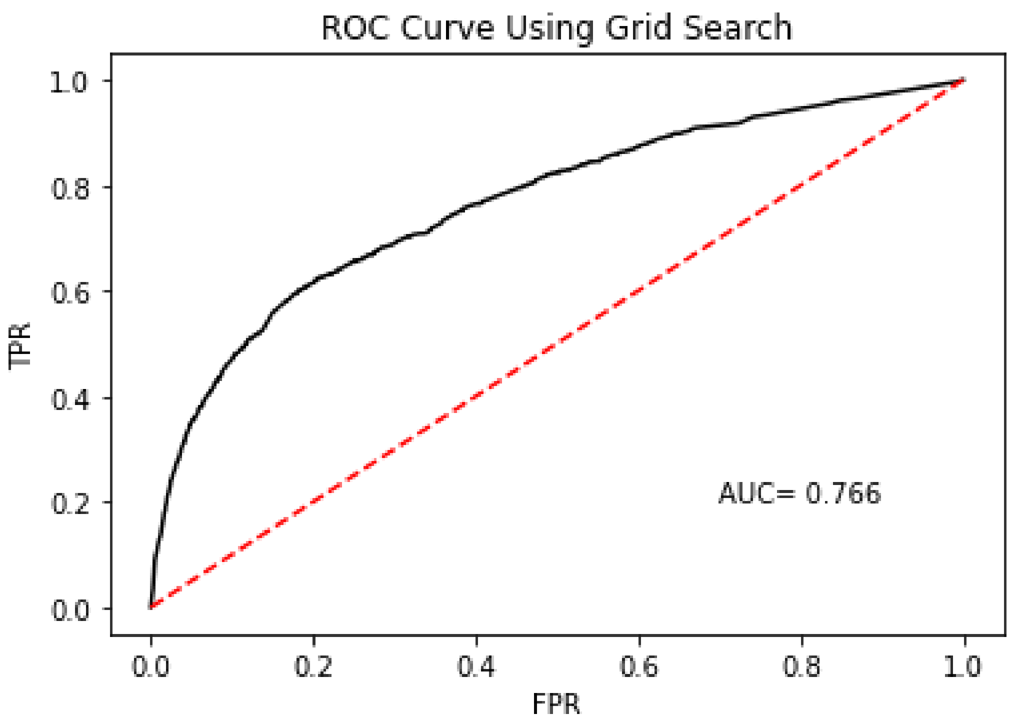
Random Forest w/ GridSearch

[44]:
```
print("GridSearch Random Forest Prediction Results: ")
#Best Predictors
print("Best Random Forest Parameters: {}".format(RFC_GS.best_params_))

#Cross Validation Score
print("Cross Validation Accuracy: {:.3f}".format(rfc_gs_cross_validation))

# Accuracy:
print("Accuracy: {:.3f}".format(rfc_gs_acc))

# Precision
print('Precision: {:.3f}'.format(rfc_gs_prec))

# Recall
print('Recall: {:.3f}'.format(rfc_gs_recall))
```

```
GridSearch Random Forest Prediction Results:
Best Random Forest Parameters: {'criterion': 'gini', 'max_depth': 8,
'max_leaf_nodes': 15, 'n_estimators': 6}
Cross Validation Accuracy: 0.818
Accuracy: 0.817
Precision: 0.685
Recall: 0.309
```

[45]:
```
fpr,tpr,th = roc_curve(y_test, pp_rfcgs[:,1])
rfc_gs_auc = roc_auc_score(y_test, pp_rfcgs[:,1])
plt.plot(fpr,tpr, 'k-')
plt.plot(np.arange(0,1.1,0.1), np.arange(0,1.1,0.1), 'r--')
plt.title('ROC Curve Using Grid Search')
plt.xlabel('FPR')
plt.ylabel('TPR')
```

```
plt.text(0.7,0.2, 'AUC= ' + "{:.3f}".format(rfc_gs_auc));
```



1.4.7 AdaBoost Classifier

[46]:
```
abc = AdaBoostClassifier(DTC(max_depth=3), n_estimators=100, random_state=25)
abc.fit(X_train, y_train)
```

[46]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   n_estimators=100, random_state=25)
```

[47]:
```
pp_abc = abc.predict_proba(X_test)
y_pred_abc = abc.predict(X_test)
abc_acc = abc.score(X_test,y_test)
abc_prec = precision_score(y_test, y_pred_abc)
abc_recall = recall_score(y_test, y_pred_abc)
abc_cv_score = cross_val_score(abc, X_train, y_train, cv=10).mean()
```

Confusion Matrix

[48]:
```
pd.DataFrame(confusion_matrix(y_test, y_pred_abc, labels=[0,1]))
```

[48]:

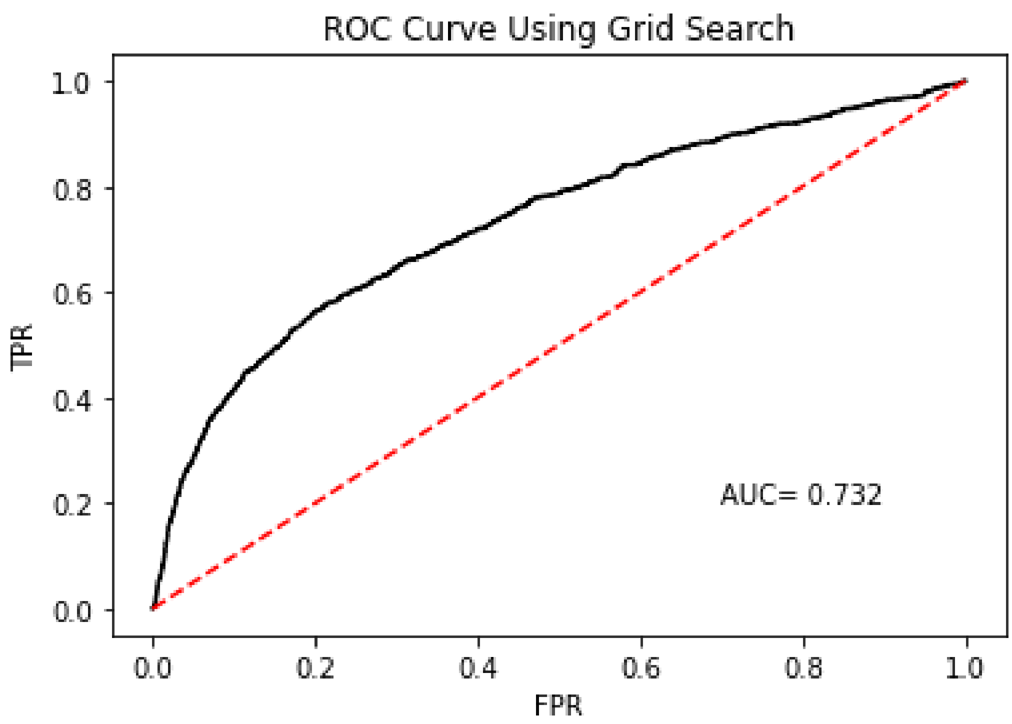
	0	1
0	5396	448
1	1037	611

AdaBoost Classifier Metrics

[49]:
```
print("Adaboost Prediction Results: ")

#Cross Validation Score
print("Cross Validation Accuracy: {:.3f}".format(abc_cv_score))

# Accuracy:
print("Accuracy: {:.3f}".format(abc_acc))

# Precision
print('Precision: {:.3f}'.format(abc_prec))

# Recall
print('Recall: {:.3f}'.format(abc_recall))
```

```
Adaboost Prediction Results:
Cross Validation Accuracy: 0.801
Accuracy: 0.802
Precision: 0.577
Recall: 0.371
```

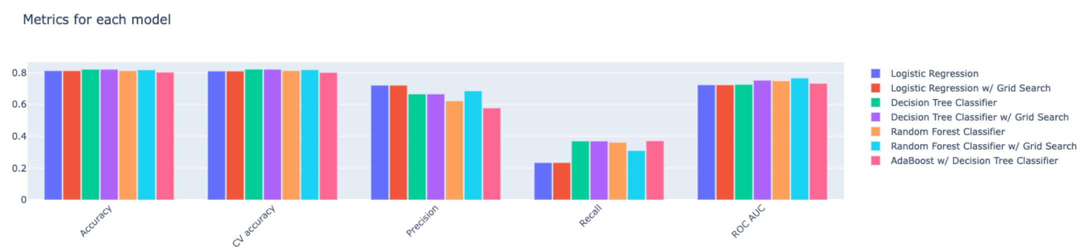
[50]:
```
fpr,tpr,th = roc_curve(y_test, pp_abc[:,1])
ada_auc = roc_auc_score(y_test, pp_abc[:,1])
plt.plot(fpr,tpr, 'k-')
plt.plot(np.arange(0,1.1,0.1), np.arange(0,1.1,0.1), 'r--')
plt.title('ROC Curve Using Grid Search')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.text(0.7,0.2, 'AUC= ' + "{:.3f}".format(ada_auc));
```



1.4.8 Aggregated Results

[51]:
```
metrics=['Accuracy', 'CV accuracy', 'Precision','Recall','ROC AUC']

#plots

fig = go.Figure(data=[
    go.Bar(name='Logistic Regression', x=metrics,␣
 ↪y=[logreg_acc,logreg_cv_score,logreg_prec,logreg_recall, lg_auc]),
    go.Bar(name='Logistic Regression w/ Grid Search', x=metrics,␣
 ↪y=[logreg_gs_acc,logreg_gs_cross_validation,logreg_gs_prec,logreg_gs_recall,␣
 ↪lg_gs_auc]),
    go.Bar(name='Decision Tree Classifier', x=metrics, y=[dtc_acc,dtc_cv_score␣
 ↪,dtc_prec,dtc_recall, dtc_auc]),
    go.Bar(name='Decision Tree Classifier w/ Grid Search', x=metrics,␣
 ↪y=[dtc_gs_acc,dtc_gs_cross_validation,dtc_gs_prec,dtc_gs_recall,␣
 ↪dtc_gs_auc]),
    go.Bar(name='Random Forest Classifier', x=metrics, y=[rfc_acc,rfc_cv_score,␣
 ↪rfc_prec,rfc_recall, rfc_auc]),
    go.Bar(name='Random Forest Classifier w/ Grid Search', x=metrics,␣
 ↪y=[rfc_gs_acc,rfc_gs_cross_validation,rfc_gs_prec,rfc_gs_recall,␣
 ↪rfc_gs_auc]),
```

```
    go.Bar(name='AdaBoost w/ Decision Tree Classifier', x=metrics,␣
 ↪y=[abc_acc,abc_cv_score,abc_prec,abc_recall, ada_auc]),
])

fig.update_layout(title_text='Metrics for each model',
                  barmode='group',xaxis_tickangle=-45,bargroupgap=0.05)
fig.show()
```



1.4.9 Discussion

Out of the all the models chosen for this project, there wasn't one that predicted if a person would default on their credit card debt significantly better than the rest. The Decision Tree Classifier with and without grid search had the best accuracy, cross validation accuracy and recall. Where Logistic Regression had the highest precision, and the Random Forest had the best AUC score. I would say none of the models did particularly well. My thoughts on the reason for such a low score for almost every metric is that there were not enough features within the dataset. The data set had only 24 features to start with and 2 of the features were the ID and classifier label. So, from the start, we had to drop 2 features from the data set. After doing some analysis, I found that 12 of the features were highly correlated with each other and 10 features needed to be dropped. So, by the time I got to running the machine learning models, I was left with only 10 features to work with. Next time, I should keep in mind the number of features the dataset has, since most of these models needs more datapoints and features to be successful.

```
[ ]:
```