In [4]:
```
import os
import numpy as np
import pandas as pd
from sklearn import model_selection
from sklearn.metrics import roc_curve
from sklearn.metrics import auc,confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.grid_search import GridSearchCV
from xgboost import XGBClassifier
```

In [5]:
```
BSDF_dev = pd.read_csv("DEV_SAMPLE.csv")
X_cont =  BSDF_dev[['Age', 'Gender', 'Balance', 'Occupation',
                'No_OF_CR_TXNS', 'SCR', 'Holding_Period']]
y = BSDF_dev["Target"]

BSDF_holdout = pd.read_csv("HOLDOUT_SAMPLE.csv")
X_holdout =  BSDF_holdout[['Age', 'Gender', 'Balance', 'Occupation',
                'No_OF_CR_TXNS', 'SCR', 'Holding_Period']]

y_test = BSDF_holdout["Target"]
```

In [6]:
```
#Categorical Variable to Numerical Variables
X = pd.get_dummies(X_cont)
X_test = pd.get_dummies(X_holdout)
X.columns
```

Out[6]:
```
Index(['Age', 'Balance', 'No_OF_CR_TXNS', 'SCR', 'Holding_Period', 'Gender_F',
       'Gender_M', 'Gender_O', 'Occupation_PROF', 'Occupation_SAL',
       'Occupation_SELF-EMP', 'Occupation_SENP'],
      dtype='object')
```

In [7]:
```
model = XGBClassifier()
model.fit(X, y)

pred_y_train = model.predict(X)
pred_y_train
```

Out[7]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [8]:
```
## Let us see the classification accuracy of our model
score = accuracy_score(y, pred_y_train)
score
```

Out[8]:
```
0.91735714285714287
```

In [9]:
```
pred_y_test = model.predict(X_test)
pred_y_test
```

Out[9]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [10]:
```
## Let us see the classification accuracy of our model
score_test = accuracy_score(y_test, pred_y_test)
score_test   ##0.92066666666666663
```

Out[10]:
```
0.92066666666666663
```

In [11]:
```
y_train_prob = model.predict_proba(X)
fpr, tpr, thresholds =  roc_curve(y, y_train_prob[:,1])
auc(fpr, tpr)    ##0.82503403949628207
```

Out[11]:
```
0.82503403949628207
```

In [12]:
```
y_test_prob = model.predict_proba(X_test)
fpr, tpr, thresholds =  roc_curve(y_test, y_test_prob[:,1])
auc(fpr, tpr)
```

Out[12]: 0.77417284550780385

In [13]:
```
scores = model_selection.cross_val_score(model, X, y, cv = 10, scoring='roc_auc
')
scores.mean()
scores.std()
```

Out[13]: 0.017636272635206406

In [14]:
```
scores.mean()
```

Out[14]: 0.78465241016649534

In [15]:
```
param_dist = {"n_estimators":np.arange(10,20),
              "learning_rate": [0.3,0.5,0.6,0.7,0.8,0.9,1,1.1,1.2,1.3],
             }
```

In [16]:
```
tree = XGBClassifier()
tree_cv  = GridSearchCV(tree, param_dist,
                        scoring = 'roc_auc', verbose = 100)
tree_cv.fit(X, y)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[CV] learning_rate=0.3, n_estimators=10 ..............................
[CV] ..... learning_rate=0.3, n_estimators=10, score=0.754108 -   0.1s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=10 ..............................
[CV] ..... learning_rate=0.3, n_estimators=10, score=0.786784 -   0.1s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=10 ..............................
[CV] ..... learning_rate=0.3, n_estimators=10, score=0.769692 -   0.1s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=11 ..............................
[CV] ..... learning_rate=0.3, n_estimators=11, score=0.753828 -   0.1s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=11 ..............................
[CV] ..... learning_rate=0.3, n_estimators=11, score=0.786083 -   0.1s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.5s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=11 ..............................
[CV] ..... learning_rate=0.3, n_estimators=11, score=0.772818 -   0.1s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.7s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=12 ..............................
[CV] ..... learning_rate=0.3, n_estimators=12, score=0.760068 -   0.2s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.8s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=12 ..............................
[CV] ..... learning_rate=0.3, n_estimators=12, score=0.785876 -   0.1s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    1.0s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=12 ..............................
[CV] ..... learning_rate=0.3, n_estimators=12, score=0.773209 -   0.1s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    1.1s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=13 ..............................
[CV] ..... learning_rate=0.3, n_estimators=13, score=0.759941 -   0.1s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    1.2s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=13 ..............................
[CV] ..... learning_rate=0.3, n_estimators=13, score=0.783707 -   0.2s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    1.4s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=13 ..............................
[CV] ..... learning_rate=0.3, n_estimators=13, score=0.776145 -   0.1s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    1.5s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=14 ..............................
[CV] ..... learning_rate=0.3, n_estimators=14, score=0.760359 -   0.2s
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    1.7s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=14 ..............................
[CV] ..... learning_rate=0.3, n_estimators=14, score=0.781375 -   0.1s
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    1.8s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=14 ..............................
[CV] ..... learning_rate=0.3, n_estimators=14, score=0.775887 -   0.1s
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:    2.0s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=15 ..............................
[CV] ..... learning_rate=0.3, n_estimators=15, score=0.761971 -   0.2s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:    2.1s remaining:    0.0
s
[CV] learning_rate=0.3, n_estimators=15 ..............................
[CV] ..... learning_rate=0.3, n_estimators=15, score=0.784570 -   0.1s
[Parallel(n_jobs=1)]: Done  17 out of  17 | elapsed:    2.3s remaining:    0.0
s
```

Out[16]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_byl
evel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': array([10, 11, 12, 13, 14, 15, 16, 17, 18,
19]), 'learning_rate': [0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3]},
       pre_dispatch='2*n_jobs', refit=True, scoring='roc_auc', verbose=100)
```

In [17]:
```
## Building the model using best combination of parameters
print("Tuned Decision Tree parameter : {}".format(tree_cv.best_params_))
classifier = tree_cv.best_estimator_
classifier.fit(X,y)
```

```
Tuned Decision Tree parameter : {'learning_rate': 0.3, 'n_estimators': 19}
```

Out[17]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.3, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=19,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [18]:
```
y_train_prob = classifier.predict_proba(X)
fpr, tpr, thresholds = roc_curve(y, y_train_prob[:,1])
auc_d = auc(fpr, tpr)
auc_d
```

Out[18]:
```
0.80536931228006747
```

In [19]:
```
y_test_prob = classifier.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_test_prob[:,1])
auc_h = auc(fpr, tpr)
auc_h
```

Out[19]:
```
0.77117557835850858
```

In [20]:
```
Prediction = classifier.predict_proba(X)
BSDF_dev["prob_score"] = Prediction[:,1]
```

In [21]:
```
#scoring step
#decile code
def deciles(x):
    decile = pd.Series(index=[0,1,2,3,4,5,6,7,8,9])
    for i in np.arange(0.1,1.1,0.1):
        decile[int(i*10)]=x.quantile(i)
    def z(x):
        if x<decile[1]: return(1)
        elif x<decile[2]: return(2)
        elif x<decile[3]: return(3)
        elif x<decile[4]: return(4)
        elif x<decile[5]: return(5)
        elif x<decile[6]: return(6)
        elif x<decile[7]: return(7)
        elif x<decile[8]: return(8)
        elif x<decile[9]: return(9)
        elif x<=decile[10]: return(10)
        else:return(np.NaN)
    s=x.map(z)
    return(s)
```

In [22]:
```
def Rank_Ordering(X,y,Target):
    X['decile']=deciles(X[y])
    Rank=X.groupby('decile').apply(lambda x: pd.Series([
        np.min(x[y]),
        np.max(x[y]),
        np.mean(x[y]),
        np.size(x[y]),
        np.sum(x[Target]),
        np.size(x[Target][x[Target]==0]),
        ],
        index=(["min_resp","max_resp","avg_resp",
                "cnt","cnt_resp","cnt_non_resp"])
        )).reset_index()
    Rank = Rank.sort_values(by='decile',ascending=False)
    Rank["rrate"] = Rank["cnt_resp"]*100/Rank["cnt"]
    Rank["cum_resp"] = np.cumsum(Rank["cnt_resp"])
    Rank["cum_non_resp"] = np.cumsum(Rank["cnt_non_resp"])
    Rank["cum_resp_pct"] = Rank["cum_resp"]/np.sum(Rank["cnt_resp"])
    Rank["cum_non_resp_pct"]=Rank["cum_non_resp"]/np.sum(Rank["cnt_non_resp"])
    Rank["KS"] = Rank["cum_resp_pct"] - Rank["cum_non_resp_pct"]
    Rank
    return(Rank)

Rank = Rank_Ordering(BSDF_dev,"prob_score","Target")
Rank
```

Out[22]:

	decile	min_resp	max_resp	avg_resp	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	c
9	10	0.193776	0.829256	0.301227	1400.0	502.0	898.0	35.857143	502.0	8
8	9	0.134253	0.193697	0.159821	1400.0	230.0	1170.0	16.428571	732.0	2
7	8	0.099884	0.134190	0.115957	1400.0	145.0	1255.0	10.357143	877.0	3
6	7	0.078141	0.099829	0.087936	1400.0	110.0	1290.0	7.857143	987.0	4
5	6	0.060255	0.078123	0.069074	1402.0	88.0	1314.0	6.276748	1075.0	5
4	5	0.045603	0.060239	0.052517	1401.0	56.0	1345.0	3.997145	1131.0	7
3	4	0.034140	0.045591	0.039973	1399.0	50.0	1349.0	3.573981	1181.0	8
2	3	0.025592	0.034094	0.029515	1401.0	33.0	1368.0	2.355460	1214.0	9
1	2	0.018438	0.025588	0.021969	1400.0	13.0	1387.0	0.928571	1227.0	1
0	1	0.007246	0.018438	0.013583	1397.0	8.0	1389.0	0.572656	1235.0	1

In [23]:
```
## Let us see the Rank Ordering on Hold-Out Dataset
Prediction_h = classifier.predict_proba(X_test)
BSDF_holdout["prob_score"] = Prediction_h[:,1]

Rank_h = Rank_Ordering(BSDF_holdout,"prob_score","Target")
Rank_h
```

Out[23]:

	decile	min_resp	max_resp	avg_resp	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cu
9	10	0.188203	0.829256	0.289158	602.0	168.0	434.0	27.906977	168.0	43
8	9	0.130801	0.188202	0.155600	601.0	91.0	510.0	15.141431	259.0	94
7	8	0.096769	0.130751	0.112449	597.0	72.0	525.0	12.060302	331.0	14
6	7	0.075429	0.096704	0.085079	600.0	45.0	555.0	7.500000	376.0	20
5	6	0.057761	0.075419	0.066308	600.0	52.0	548.0	8.666667	428.0	25
4	5	0.043982	0.057721	0.050109	600.0	19.0	581.0	3.166667	447.0	31
3	4	0.033456	0.043975	0.038631	600.0	18.0	582.0	3.000000	465.0	37
2	3	0.025313	0.033369	0.029041	600.0	17.0	583.0	2.833333	482.0	43
1	2	0.018487	0.025301	0.021835	602.0	11.0	591.0	1.827243	493.0	49
0	1	0.007246	0.018438	0.013456	598.0	5.0	593.0	0.836120	498.0	55