In [1]:
```
#Import Libraries
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import re
pd.options.display.max_columns=40

### Import Descision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
```

In [2]:
```
### Load
path_hw6="C:\\Users\\fbaharkoush\\IE 598 Machine Learning\\Homework\\HW 6\\"
df_ts=pd.read_csv(path_hw6+"ccdefault.csv").drop("ID",axis=1)
### Missing Values
df_ts.isnull().sum().sum()==0
```

Out[2]: True

In [3]:
```
df_ts.head()
```

Out[3]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5
0	20000	2	2	1	24	2	2	-1	-1	-2	-2	3913	3102	689	0	0
1	120000	2	2	2	26	-1	2	0	0	0	2	2682	1725	2682	3272	3455
2	90000	2	2	2	34	0	0	0	0	0	0	29239	14027	13559	14331	14948
3	50000	2	2	1	37	0	0	0	0	0	0	46990	48233	49291	28314	28959
4	50000	1	2	1	57	-1	0	-1	0	0	0	8617	5670	35835	20940	19146

In [4]:
```
## Values
X=df_ts.drop("DEFAULT",axis=1).values
#y=df_ts["squeeze"].values
y=df_ts["DEFAULT"].values
```

Part 1: Random test train splits

Fit the Model for 10 different samples by changing random_state from 1 to 10 in sequence.

In [5]:
```
list_of_models_score_dt_test=[]
list_of_models_score_dt_train=[]
list_of_y_pred_dt_test=[]
list_of_y_pred_dt_train=[]
list_of_roc_test=[]
list_of_roc_dt_train=[]
list_of_random_state=[]
for i in range(1,11):
    ### Create the Model
    Decision_Tree=DecisionTreeClassifier(max_depth=4)
    ### Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .1, random_state=i,  stratify=y)
    ### Fit values to the model
    Decision_Tree.fit(X_train,y_train)

    ### Prediction test
    y_pred_dt_test=Decision_Tree.predict(X_test)
    ### Prediction train
    y_pred_dt_train=Decision_Tree.predict(X_train)

    ### Store the prediction of each model
    #list_of_y_pred_dt_test.append(y_pred_dt_test)
    #list_of_y_pred_dt_train.append(y_pred_dt_train)

    #### Evaluate the score of each model
    dt_model_score_train=Decision_Tree.score(X_train,y_train)
    dt_model_score_test=Decision_Tree.score(X_test,y_test)

    ### ROC score
    list_of_roc_dt_train.append(roc_auc_score(y_train,y_pred_dt_train))
    list_of_roc_test.append(roc_auc_score(y_test,y_pred_dt_test))

    ### Store the Scores of each model
    list_of_models_score_dt_train.append(dt_model_score_train)
    list_of_models_score_dt_test.append(dt_model_score_test)
    ### Store Neghbour Number
    list_of_random_state.append(i)
```

Report performance

In [6]:
```
df_dt_performance=pd.DataFrame({"Random_State":list_of_random_state,
              "DT Model Accuracy% Train":list_of_models_score_dt_train,
             "DT Model Accuracy% Test":list_of_models_score_dt_test,
            "DT Model ROC Score Train":list_of_roc_dt_train,
            "DT Model ROC Score Test":list_of_roc_test})
df_dt_performance.sort_values("DT Model ROC Score Test",ascending=False)
```

Out[6]:

	Random_State	DT Model Accuracy% Train	DT Model Accuracy% Test	DT Model ROC Score Train	DT Model ROC Score Test
0	1	0.822556	0.828333	0.661102	0.669869
6	7	0.823259	0.825333	0.660535	0.664170
1	2	0.822593	0.824000	0.661366	0.662774
9	10	0.823778	0.820667	0.661467	0.658478
5	6	0.823889	0.819000	0.660160	0.656330
4	5	0.823148	0.818000	0.661842	0.655688
3	4	0.823852	0.820333	0.661815	0.654491
2	3	0.824111	0.817333	0.661442	0.644480
8	9	0.823852	0.817333	0.655280	0.641246
7	8	0.823778	0.816667	0.651396	0.639201

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

Out[7]:

	Random_State	DT Model Accuracy% Train	DT Model Accuracy% Test	DT Model ROC Score Train	DT Model ROC Score Test
count	10.00000	10.000000	10.000000	10.000000	10.000000
mean	5.50000	0.823481	0.820700	0.659641	0.654673
std	3.02765	0.000560	0.003942	0.003485	0.010157
min	1.00000	0.822556	0.816667	0.651396	0.639201
25%	3.25000	0.823176	0.817500	0.660254	0.646983
50%	5.50000	0.823778	0.819667	0.661234	0.656009
75%	7.75000	0.823852	0.823167	0.661461	0.661700
max	10.00000	0.824111	0.828333	0.661842	0.669869

Part 2: Cross validation

In [8]:
```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size = .1, random_state=4)
```

In [9]:
```
dt=DecisionTreeClassifier()
CV_ROC_Scores=cross_val_score(dt,X_train,y_train,cv=10,scoring='roc_auc',n_jobs=-1)
```

In [10]:
```
print("ROC_AUC Score of 10-folds",CV_ROC_Scores,"\n")
print("Average ROC_AUC Score of 10-folds",np.mean(CV_ROC_Scores),"\n")
print("Average ROC_AUC Score of Random test train splits",np.mean(df_dt_performance["DT Model ROC Score Test"]))
```

```
ROC_AUC Score of 10-folds [0.65005777 0.60603264 0.6173527  0.60433074 0.60929052 0.58796656
 0.61783752 0.62469064 0.6125184  0.62043683]

Average ROC_AUC Score of 10-folds 0.6150514312168099

Average ROC_AUC Score of Random test train splits 0.65467280079221
```

Conclusion

Average ROC_AUC Score of 10-folds is less than ROC_AUC Score of Random test train splits CV is said to overfit the training set. To remedy overfitting: 1. derease complexity of the model such as decreasing max mdepth, increasing min samples per leaf and gather mode data.

Part 2: Cross validation

Grid Search

In [11]:
```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size = .1, random_state=4)
```

In [12]:
```
tree_parameters = {'criterion':['gini','entropy'],'max_depth':[1,2,3,4,5,6,7,8,9,10]}
```

In [13]:
```
clf_decision_tree=GridSearchCV(DecisionTreeClassifier(), tree_parameters, cv=10)
#clf_decision_tree.fit(X, y)
```

In [14]:
```
clf_decision_tree.fit(X_train, y_train)
```

Out[14]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=None,
                                              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=None,
                                              splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [15]:
```
print("clf_decision_tree best score is" , clf_decision_tree.best_score_)
print("clf_decision_tree best score paramteres are: \n \n" , clf_decision_tree.best_estimator_)
```

```
clf_decision_tree best score is 0.8224814814814815
clf_decision_tree best score paramteres are:

 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                        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=None, splitter='best')
```

In [16]:
```
y_pred_clf_dt_train=clf_decision_tree.predict(X_train)
y_pred_clf_dt_test=clf_decision_tree.predict(X_test)
```

In [17]:
```
print("ROC Score of GirdSearch Classifier on Training set",roc_auc_score(y_train,y_pred_clf_dt_train))
print("ROC Score of GirdSearch Classifier on test set",roc_auc_score(y_test,y_pred_clf_dt_test),"\n")
print(df_dt_performance.sort_values("DT Model ROC Score Test",
                                    ascending=False).head(1).drop([
    "DT Model Accuracy% Train","DT Model Accuracy% Test"],axis=1))
```

```
ROC Score of GirdSearch Classifier on Training set 0.662724094598382
ROC Score of GirdSearch Classifier on test set 0.6464423610713496

   Random_State  DT Model ROC Score Train  DT Model ROC Score Test
0             1                  0.661102                 0.669869
```

Final conclusion

As the result show after performin grid search the ROC score of the model is not significantly improving. we most likely need more data to biuld a more robust model. On the other hand other classification methods such as Logistic Regression or Elastic Regression should be fitted and evaluated. They may produce better ROC. At this point Random test train splits is the most efficient approch and it is doing as good as other models fitted.

In [ ]:

In [ ]:

In [ ]: