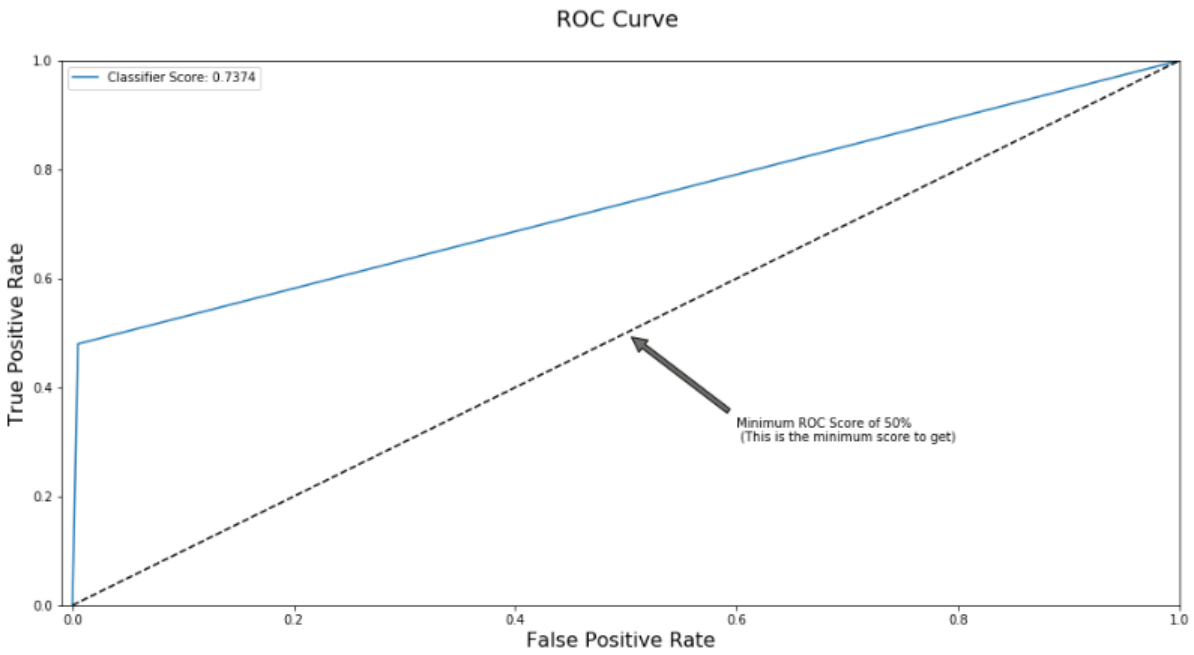
Summay:

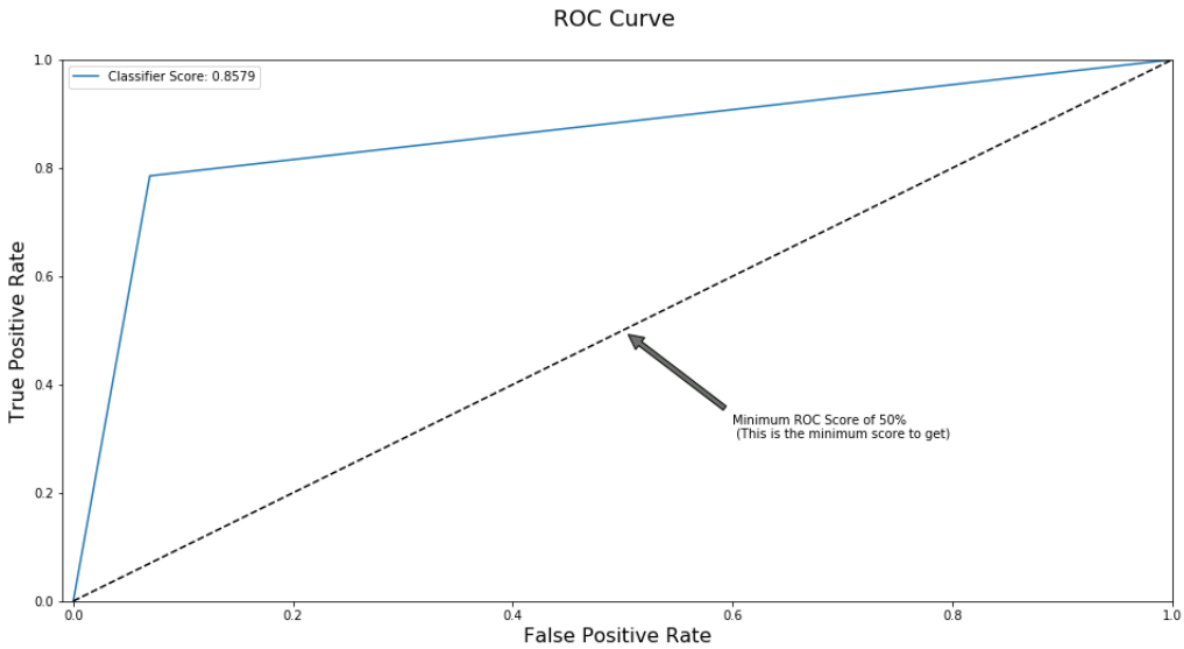
In this project on classification through predictive model development, we use two models - Random forest and logistic regression to train data and ultimately choose the one – Random Forest model with better performance through comparison on model performance.

Considering each model has its pros and cons, logistic regression can provide probabilities for outcomes and low variance as well as working well with feature decision boundaries, but it has high bias; random forest model can reduce variance and decorrelate  trees, but it is not easy to visually interpret, for this classification on imbalanced data, to measure the model performance, we not only need to consider accuracy and precision, but also need to look at the FP rate and TP rate through AUC.

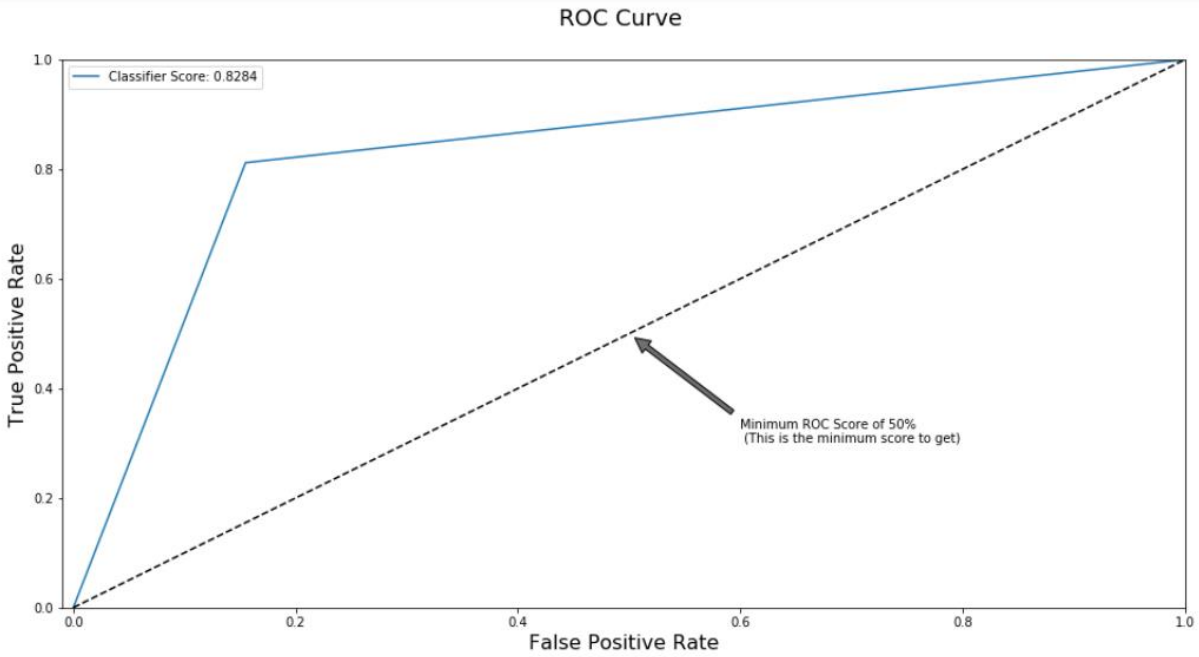
1. Random Forest with No sampling data (accuracy=88.95%,precision = 99.52%)




2. Random Forest with SMOTE sampling technique to transform imbalanced data to balanced data(accuracy = 90% Precision = 93%)




3. Logistic regression with SMOTE(accuracy = 83.83% precision = 84.5%)




Through comparison about accuracy, precision as well as AUC area, I ultimately select the random forest model as well as SMOTE technique that is used to transform the imbalanced data to balanced data.

Code:

```
%matplotlib inline

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from imblearn.over_sampling import SMOTE

from imblearn.under_sampling import NearMiss

from imblearn.over_sampling import RandomOverSampler

from imblearn.under_sampling import RandomUnderSampler

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import confusion_matrix

from sklearn.impute import SimpleImputer

from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import LinearRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier
```

**#1.Data Preprocessing**

```
train = pd.read_csv('exercise_05_train.csv')
```

```
test = pd.read_csv('exercise_05_test.csv')

#transform x45

def pct_to_float(x):

    if type(x) == float:

        return None

    else:

        return float(x.strip('%'))/100

train['x45'] = train['x45'].apply(lambda x:pct_to_float(x))

test['x45'] = test['x45'].apply(lambda x:pct_to_float(x))

#transform x41

def amt_to_float(x):

    if type(x) == float:

        return None

    else:

        return float(x.strip('$'))

train['x41'] = train['x41'].apply(lambda x:amt_to_float(x))

test['x41'] = test['x41'].apply(lambda x:amt_to_float(x))

#transform x35

train['x35']= train['x35'].replace("fri","friday", regex=False)

train['x35']= train['x35'].replace("wed","wednesday", regex=False)

train['x35']= train['x35'].replace("thur","thurday", regex=False)

train['x35']= train['x35'].replace("thurday","thursday", regex=False)

test['x35']= test['x35'].replace("fri","friday", regex=False)

test['x35']= test['x35'].replace("wed","wednesday", regex=False)

test['x35']= test['x35'].replace("thur","thurday", regex=False)
```

```
test['x35']= test['x35'].replace("thurday","thursday", regex=False)

#Deal with missing values and imputed missing values

missing_data =
pd.DataFrame({'total_missing':train.isnull().sum(),'perc_missing':(train.isnull().sum()/len(train)*100)})

def missing_value_imputation(x,features):

    x = x.reset_index(drop=True)

    numeric_features = x[features]._get_numeric_data().columns

    cat_features = list(set(features) - set(numeric_features))

    imputer = SimpleImputer()

    imputed_features = imputer.fit_transform(x[numeric_features])

    imputed_featuredf = pd.DataFrame(imputed_features, columns = numeric_features)

    imputed_featuredf[cat_features] = x[cat_features].fillna('Null')

    return imputed_featuredf

imputed_train = missing_value_imputation(train,train.columns)

imputed_test = missing_value_imputation(test,test.columns)

sum(imputed_train.isnull().sum()),sum(imputed_test.isnull().sum())

# Look at variable statistics

train.describe()

#class imbalance rate

print('0', round(train['y'].value_counts()[0]/len(train)*100,2), '% of the dataset')

print('1', round(train['y'].value_counts()[1]/len(train)*100,2), '% of the dataset')

#look at the class distribution

colors = ["#0101DF","#DF0101"]


sns.countplot('y', data=train, palette = colors)

plt.title('class distribution \n (0:0 || 1:1)', fontsize=14)
```

```
plt.show()

#encode categorical variables

def transform(x,y):

   cat_cols = list(set(x.columns) - set(x._get_numeric_data().columns))

   encoded_x = x.copy()

   for col in cat_cols:

      le = preprocessing.LabelEncoder()

      encoded_x[col] = le.fit_transform(encoded_x[col])

      y[col] = le.transform(y[col])

   return encoded_x,y

encoded_train,encoded_test = transform(imputed_train,imputed_test)

#2.Split train and test dataset from train data

predictors = list(encoded_train.columns)

predictors.remove('y')

x_train, x_test, y_train, y_test = train_test_split(encoded_train[predictors], encoded_train['y'],
test_size=0.3, random_state=42)

test_x = encoded_test[predictors]

#3.Feature selection – Select significant features

#option1 RFECV

from sklearn.feature_selection import RFECV

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

rfecv_logreg = RFECV(estimator=logreg, step=20, cv=5, # may need to change cv

          scoring='accuracy', n_jobs=-1)

rfecv_logreg.fit(x_train, y_train)

names = x_train.columns.values
```

```
ranks = rfecv_logreg.ranking_

logreg_rfe_names_ranks=list(zip(names, ranks))

logreg_rfe_ns_rs = pd.DataFrame(data = logreg_rfe_names_ranks, columns=['Feat_names', 'F_ranks'])

logreg_rfe_ns_rs_sorted = logreg_rfe_ns_rs.sort_values(['F_ranks', 'Feat_names'], ascending = [True, True])

print(logreg_rfe_ns_rs_sorted)

mygs_logreg_rfecv_cvshuffle = GridSearchCV(estimator=logreg, param_grid={}, cv=5,scoring ='accuracy')

mygs_logreg_rfecv_cvshuffle.fit(x_train[logreg_rfe_ns_rs_sorted[logreg_rfe_ns_rs_sorted['F_ranks']<=1 ].Feat_names],y_train)

print(mygs_logreg_rfecv_cvshuffle.best_score_)

#4.Model 1: Random Forest

#4.1 model parameter tuning

rfc = RandomForestClassifier(random_state =42, n_jobs = -1)

randomforest_params = {'max_depth':np.arange(5,10),'n_estimators':[500, 1000],

'criterion':['gini','entropy'], 'max_features':[0.3,0.4,0.5,0.6],'max_leaf_nodes':[100,200,300], 'class_weight':[None,'balanced']}

CV_rfc = GridSearchCV(estimator = rfc, param_grid =randomforest_params,cv=5 )

CV_rfc.fit(x_train_feature_selected,y_train)

CV_rfc.best_params_

#4.2 Sampling  or not

rf_classifier = RandomForestClassifier(random_state =42, n_jobs = -1, n_estimators = 500, max_depth = 9, criterion ='entropy',max_features = 0.6, max_leaf_nodes = 300)

sampling_techniques = ['randomUnderSampling','randomOverSampling','SMOTE']

samplers = [RandomUnderSampler(return_indices=True),RandomOverSampler(),SMOTE(ratio='minority')]

for w in range(len(samplers)):

    sampler = samplers[w]
```

```
    if w==0:
        x_sampled,y_sampled,_, = sampler.fit_sample(x_train_feature_selected,y_train)
    else:
        x_sampled,y_sampled = sampler.fit_sample(x_train_feature_selected,y_train)
    rf_classifier.fit(x_sampled,y_sampled)
    pred = rf_classifier.predict(x_test_feature_selected)
    print(sampling_techniques[w])
    print(confusion_matrix(y_test,pred))
#no sampling
rf_classifier.fit(x_train_feature_selected,y_train)
pred_1 = rf_classifier.predict(x_test_feature_selected)
confusion_matrix(y_test,pred_1)
#4.3 ROC curve
tree_fpr, tree_tpr, tree_threshold = roc_curve(y_test, pred)
def graph_roc_curve(fpr, tpr):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n', fontsize=18)
    plt.plot(fpr, tpr, label='Classifier Score: {:.4f}'.format(roc_auc_score(y_test, pred)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5),
xytext=(0.6, 0.3),
                arrowprops=dict(facecolor='#6E726D', shrink=0.05),
                )
```

```
    plt.legend()

graph_roc_curve(tree_fpr, tree_tpr)

#4.4 Prediction with SMOTE Sampling

submission1 = pd.DataFrame(columns=['Prediction'],index=test_x_feature_selected.index,
data=rf_classifier.predict_proba(test_x_feature_selected)[:,1])

# set column name

submission1.index.name='Id'

# Write out the submission into csv file

submission1.to_csv("Model1_RF_proba.csv")

# Sanity check on the submission

submission1.describe().astype(float)

#5.Model2: Logistic Regression

#5.1 Model parameter tuning

LogR = LogisticRegression(random_state =0)

LogR_params = {"penalty": ['l1', 'l2'], "solver":['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], 'C': [0.001,
0.01, 0.1, 1, 10, 100, 1000], 'multi_class': ['ovr','auto'], 'class_weight':[None,'balanced']}

CV_LogRC = GridSearchCV(estimator = LogR, param_grid =LogR_params,cv=5 )

CV_LogRC.fit(x_train_feature_selected,y_train)

CV_LogRC.best_params_

#5.2 Sampling or not

logR = LogisticRegression(random_state =0, solver='liblinear', C=0.01, multi_class ='ovr')

sampling_techniques = ['randomUnderSampling','randomOverSampling','SMOTE']

samplers =
[RandomUnderSampler(return_indices=True),RandomOverSampler(),SMOTE(ratio='minority')]

for w in range(len(samplers)):

    sampler = samplers[w]

    if w==0:
```

```
        x_sampled,y_sampled,_, = sampler.fit_sample(x_train_feature_selected,y_train)
    else:
        x_sampled,y_sampled = sampler.fit_sample(x_train_feature_selected,y_train)
    LogR.fit(x_sampled,y_sampled)
    pred = LogR.predict(x_test_feature_selected)
    print(sampling_techniques[w])
    print(confusion_matrix(y_test,pred))
#5.3 ROC Curve
tree_fpr, tree_tpr, tree_threshold = roc_curve(y_test, pred)
def graph_roc_curve(fpr, tpr):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n', fontsize=18)
    plt.plot(fpr, tpr, label='Classifier Score: {:.4f}'.format(roc_auc_score(y_test, pred)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5),
xytext=(0.6, 0.3),
                arrowprops=dict(facecolor='#6E726D', shrink=0.05),
                )
    plt.legend()
graph_roc_curve(tree_fpr, tree_tpr)
#5.4 Prediction with SMOTE Sampling
submission2 = pd.DataFrame(columns=['Prediction'],index=test_x_feature_selected.index,
data=LogR.predict_proba(test_x_feature_selected)[:,1])
```

```
# set column name

submission2.index.name='Id'

# Write out the submission into csv file

submission2.to_csv("model2_logReg_proba.csv")

# Sanity check on the submission

submission2.describe().astype(float)
```