Home Credit default prediction - Model

In [1]:
```
import datetime
print(datetime.datetime.now())
```

```
2020-02-26 20:42:34.477888
```

In [2]:
```
import sklearn
print('The scikit-learn version is {}.'.format(sklearn.__version__))
```

```
The scikit-learn version is 0.22.
```

In [3]:
```
import pandas as pd
import numpy as np

import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn import metrics
from sklearn.model_selection import GridSearchCV
```

In [4]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib as mpl
import seaborn as sns
```

read training and testing datasets

In [5]:
```
X_train_final = pd.read_csv(r"C:\Users\mamta\MMAI 2020\MMAI823_AI in Finance\Team Assignments\Team Project\X_train_redo.csv",sep=',')
#X_val_final = pd.read_csv(r"C:\Users\mamta\MMAI 2020\MMAI823_AI in Finance\Team Assignments\Team Project\X_test_final.csv",sep=',')
X_test_final = pd.read_csv(r"C:\Users\mamta\MMAI 2020\MMAI823_AI in Finance\Team Assignments\Team Project\X_test_redo.csv",sep=',')
y_train = np.loadtxt('y_train_redo.txt', dtype=int)
y_test = np.loadtxt('y_test_redo.txt', dtype=int)
#y_val = np.loadtxt('y_val.txt', dtype=int)
```

In [6]:
```
X_train_final.shape
```
Out[6]:
```
(215257, 126)
```

In [7]:
```
X_test_final.shape
```
Out[7]:
```
(92254, 126)
```

In [8]:
```
y_train.shape
```
Out[8]:
```
(215257,)
```

In [9]:
```
y_test.shape
```
Out[9]:
```
(92254,)
```

Upsampling using SMOTE

In [10]:
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1
)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train ==
0)))

# import SMOTE module from imblearn library
# pip install imblearn (if you don't have imblearn in your system)
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
X_train_up, y_train_up = sm.fit_sample(X_train_final, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_up.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_up.shap
e))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_up == 1
)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_up == 0
)))
```

```
Before OverSampling, counts of label '1': 17377
Before OverSampling, counts of label '0': 197880


Using TensorFlow backend.

After OverSampling, the shape of train_X: (395760, 126)
After OverSampling, the shape of train_y: (395760,)

After OverSampling, counts of label '1': 197880
After OverSampling, counts of label '0': 197880
```

Down sampling using Near Miss

In [11]:
```
print("Before Undersampling, counts of label '1': {}".format(sum(y_train == 1
)))
print("Before Undersampling, counts of label '0': {} \n".format(sum(y_train ==
0)))

# apply near miss
from imblearn.under_sampling import NearMiss
nr = NearMiss()

X_train_down, y_train_down = nr.fit_sample(X_train_final, y_train.ravel())

print('After Undersampling, the shape of train_X: {}'.format(X_train_down.shap
e))
print('After Undersampling, the shape of train_y: {} \n'.format(y_train_down.s
hape))

print("After Undersampling, counts of label '1': {}".format(sum(y_train_down =
= 1)))
print("After Undersampling, counts of label '0': {}".format(sum(y_train_down =
= 0)))
```

```
Before Undersampling, counts of label '1': 17377
Before Undersampling, counts of label '0': 197880

After Undersampling, the shape of train_X: (34754, 126)
After Undersampling, the shape of train_y: (34754,)

After Undersampling, counts of label '1': 17377
After Undersampling, counts of label '0': 17377
```

Plot ROC_AUC_Curve

In [12]:

```
def plot_roc(clf, X_test_final, y_test, name, ax, show_thresholds=False):
    y_pred_ada = clf.predict_proba(X_test_final)[:, 1]
    fpr, tpr, thr = roc_curve(y_test, y_pred_ada)

    #ax.plot([0, 1], [0, 1], 'k--');
    ax.plot([0, 1], [0, 1]);
    ax.plot(fpr, tpr, label='{}, AUC={:.5f}'.format(name, auc(fpr, tpr)));
    #ax.scatter(fpr, tpr,marker='*');

    if show_thresholds:
        for i, th in enumerate(thr):
            ax.text(x=fpr[i], y=tpr[i], s="{:.2f}".format(th), fontsize=9,
                     horizontalalignment='left', verticalalignment='top', colo
r='black',
                     bbox=dict(facecolor='white', edgecolor='black', boxstyle=
'round,pad=0.1', alpha=0.1));

    ax.set_xlabel('False positive rate', fontsize=18);
    ax.set_ylabel('True positive rate', fontsize=18);
    ax.tick_params(axis='both', which='major', labelsize=18);
    ax.grid(True);
    ax.set_title('ROC Curve', fontsize=18)
```

GBM STARTS HERE

GBM on regular dataset

In [15]:
```
# Train the GBM Model for Classification with hyperparameter tuning
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV

gbm_model = GradientBoostingClassifier()

loss = ["deviance", "exponential"]
learning_rate = [0.1, 0.05, 0.01, 0.005]
n_estimators = range(25,2000,50)
max_depth = range(5,16,2)
max_features = range(7,20,2)
min_samples_split = range(500,3001,250)
min_samples_leaf = range(30, 101, 10)
subsample = [0.6,0.7,0.75,0.8,0.85,0.9]
random_state = [21]

param_grid = dict(
                    loss=loss,
                    learning_rate=learning_rate,
                    n_estimators=n_estimators,
                    max_depth=max_depth,
                    max_features = max_features,
                    min_samples_split=min_samples_split,
                    min_samples_leaf=min_samples_leaf,
                    subsample = subsample,
                    random_state=random_state
)

gbm_grid = RandomizedSearchCV(gbm_model, param_grid, scoring = 'roc_auc', n_jo
bs = -1, cv=3, verbose = 1)

gbm_model_result = gbm_grid.fit(X_train_final, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 122.7min finished
```

Print best parameters

In [16]:
```
# Dictionary of best parameters
print(gbm_model_result.best_params_)
```

```
{'subsample': 0.6, 'random_state': 21, 'n_estimators': 1875, 'min_samples_spl
it': 2750, 'min_samples_leaf': 100, 'max_features': 17, 'max_depth': 11, 'los
s': 'exponential', 'learning_rate': 0.01}
```

In [17]:
```
# Train the GBM Model for Classification with hyperparameter tuning
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
```

Rebuild model with best parameters

In [18]:
```
#gbm_classifier = GradientBoostingClassifier(n_estimators=160, max_depth=3, lo
ss='exponential', learning_rate=0.1, random_state=42)

gbm_classifier = GradientBoostingClassifier(n_estimators=1875, max_depth=11, l
oss='exponential', learning_rate=0.01, random_state=21,subsample=0.6,min_sampl
es_split=2750,min_samples_leaf=100)

gbm_classifier.fit(X_train_final, y_train)

gbm_pred = gbm_classifier.predict(X_test_final)
```

Print Confusion Matrix

In [19]:
```
from sklearn.metrics import confusion_matrix

gbm_cm = confusion_matrix(y_test,gbm_pred)
print(gbm_cm)
```

```
[[84674   132]
 [ 7275   173]]
```

GBM Classification Report

In [20]:
```
import sklearn.metrics as metrics
print(metrics.classification_report(y_test, gbm_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     84806
           1       0.57      0.02      0.04      7448

    accuracy                           0.92     92254
   macro avg       0.74      0.51      0.50     92254
weighted avg       0.89      0.92      0.88     92254
```

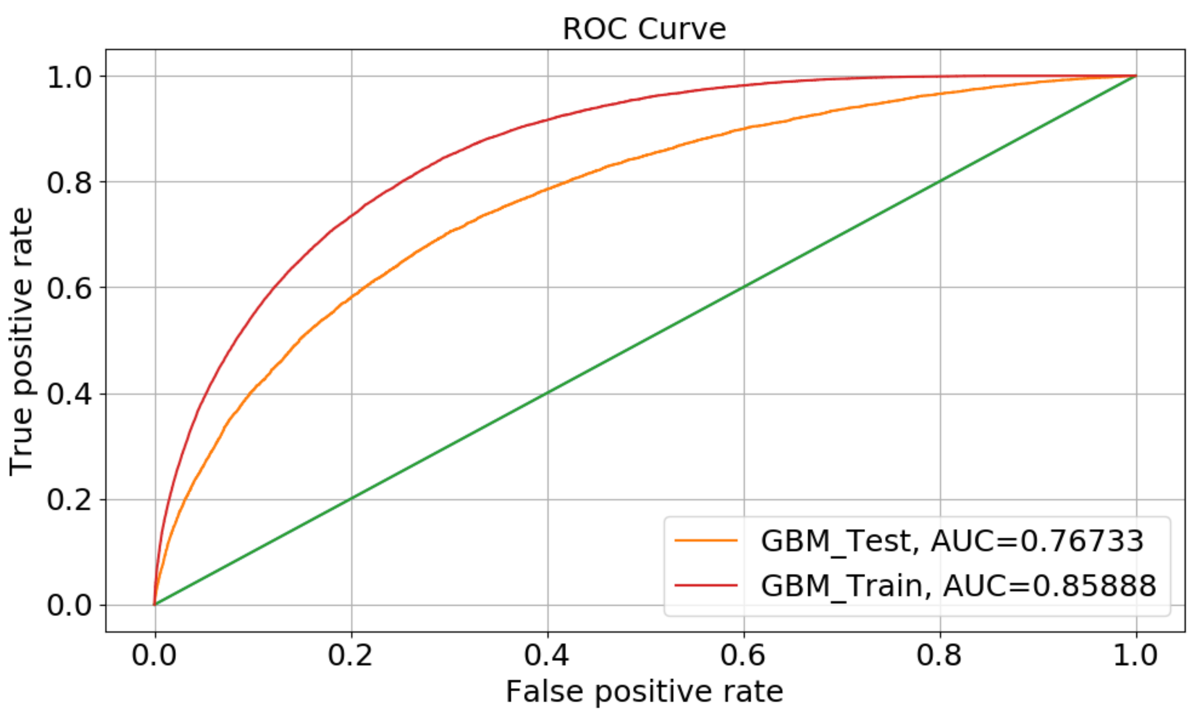
Performance Measure

In [21]:
```
print("GBM f1 score : %5.5f" %(round(f1_score(y_test,gbm_pred),3)))
print("GBM accuracy : %5.5f" %(round(accuracy_score(y_test,gbm_pred),3)))
print("GBM log loss : %5.5f" %(round(log_loss(y_test,gbm_pred),3)))
print("GBM Recall : %5.5f" %(round(recall_score(y_test,gbm_pred),3)))
print("GBM Precision : %5.5f" %(round(precision_score(y_test,gbm_pred),3)))
gbm_auc = roc_auc_score(y_test, gbm_pred)
print("GBM AUC : %5.5f" %(gbm_auc))
```

```
GBM f1 score : 0.04500
GBM accuracy : 0.92000
GBM log loss : 2.77300
GBM Recall : 0.02300
GBM Precision : 0.56700
GBM AUC : 0.51084
```

ROC AUC Curve

In [22]:
```
plt.style.use('default');
figure = plt.figure(figsize=(10, 6));
ax4 = plt.subplot(1, 1, 1);
plot_roc(gbm_classifier, X_test_final, y_test, "GBM_Test", ax4)
plot_roc(gbm_classifier, X_train_final, y_train, "GBM_Train", ax4)
plt.legend(loc='lower right', fontsize=18);
plt.tight_layout();
```



GBM Downsampled Data

In [23]:
```
gbm_down_result = gbm_grid.fit(X_train_down, y_train_down)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.3min finished
```

Print best parameters

In [24]:
```
# Dictionary of best parameters
print(gbm_down_result.best_params_)
```

```
{'subsample': 0.7, 'random_state': 21, 'n_estimators': 1275, 'min_samples_spl
it': 500, 'min_samples_leaf': 30, 'max_features': 17, 'max_depth': 11, 'los
s': 'exponential', 'learning_rate': 0.01}
```

Rebuild with best parameters

In [25]:
```
gbm_down_classifier = GradientBoostingClassifier(n_estimators=1275, max_depth=
11, loss='exponential', learning_rate=0.01, random_state=21,subsample=0.7,min_
samples_split=500,min_samples_leaf=30)


gbm_down_classifier.fit(X_train_down, y_train_down)

gbm_down_pred = gbm_down_classifier.predict(X_test_final)
```

Print Confusion Matrix

In [26]:
```
gbm_down_cm = confusion_matrix(y_test,gbm_down_pred)
print(gbm_down_cm)
```

```
[[58900 25906]
 [ 2283  5165]]
```

Print Classification report

```
In [27]: print(metrics.classification_report(y_test, gbm_down_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.69      0.81     84806
           1       0.17      0.69      0.27      7448

    accuracy                           0.69     92254
   macro avg       0.56      0.69      0.54     92254
weighted avg       0.90      0.69      0.76     92254
```

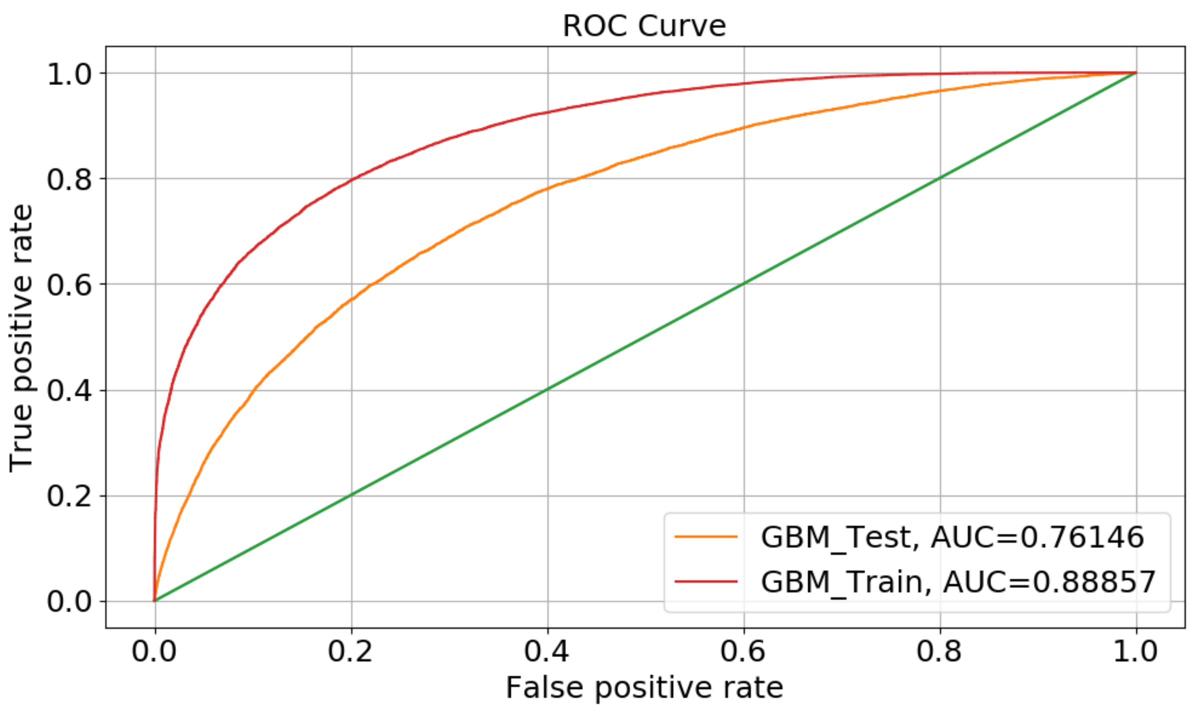
Print Performance Measures

```
In [28]: print("GBM f1 score : %5.5f" %(round(f1_score(y_test,gbm_down_pred),3)))
print("GBM accuracy : %5.5f" %(round(accuracy_score(y_test,gbm_down_pred),3)))
print("GBM log loss : %5.5f" %(round(log_loss(y_test,gbm_down_pred),3)))
print("GBM Recall : %5.5f" %(round(recall_score(y_test,gbm_down_pred),3)))
print("GBM Precision : %5.5f" %(round(precision_score(y_test,gbm_down_pred),3
)))
gbm_down_auc = roc_auc_score(y_test, gbm_down_pred)
print("GBM AUC : %5.5f" %(gbm_down_auc))
```

```
GBM f1 score : 0.26800
GBM accuracy : 0.69400
GBM log loss : 10.55400
GBM Recall : 0.69300
GBM Precision : 0.16600
GBM AUC : 0.69400
```

Print ROC AUC

In [29]:
```
plt.style.use('default');
figure = plt.figure(figsize=(10, 6));
ax4 = plt.subplot(1, 1, 1);
plot_roc(gbm_down_classifier, X_test_final, y_test, "GBM_Test", ax4)
plot_roc(gbm_down_classifier, X_train_down, y_train_down, "GBM_Train", ax4)
plt.legend(loc='lower right', fontsize=18);
plt.tight_layout();
```



GBM on Upsampled Data

In [30]:
```
gbm_up_result = gbm_grid.fit(X_train_up, y_train_up)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 303.2min finished
```

Print best parameters

In [31]:
```
# Dictionary of best parameters
print(gbm_up_result.best_params_)
```

```
{'subsample': 0.9, 'random_state': 21, 'n_estimators': 925, 'min_samples_spli
t': 500, 'min_samples_leaf': 90, 'max_features': 19, 'max_depth': 13, 'loss':
'deviance', 'learning_rate': 0.01}
```

Rebuild with best parameters

In [37]:
```
gbm_up_classifier = GradientBoostingClassifier(n_estimators=925, max_depth=13,
loss='deviance', learning_rate=0.01, random_state=21,subsample=0.9,min_samples
_split=500,min_samples_leaf=90)

gbm_up_classifier.fit(X_train_up, y_train_up)

gbm_up_pred = gbm_up_classifier.predict(X_test_final)
```

Print Confusion Matrix

In [38]:
```
gbm_up_cm = confusion_matrix(y_test,gbm_up_pred)
print(gbm_up_cm)
```

```
[[84640   166]
 [ 7267   181]]
```

Print Classification report

In [39]:
```
print(metrics.classification_report(y_test, gbm_up_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     84806
           1       0.52      0.02      0.05      7448

    accuracy                           0.92     92254
   macro avg       0.72      0.51      0.50     92254
weighted avg       0.89      0.92      0.88     92254
```

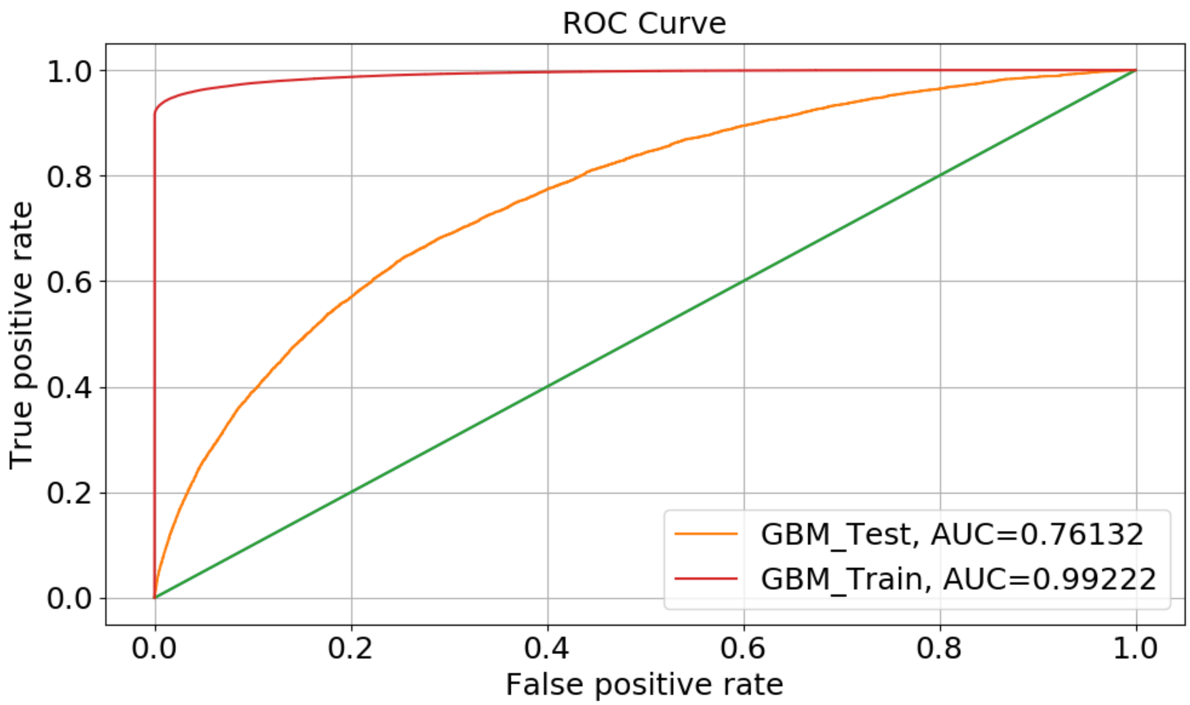
Print Performance measure

In [40]:
```
print("GBM f1 score : %5.5f" %(round(f1_score(y_test,gbm_up_pred),3)))
print("GBM accuracy : %5.5f" %(round(accuracy_score(y_test,gbm_up_pred),3)))
print("GBM log loss : %5.5f" %(round(log_loss(y_test,gbm_up_pred),3)))
print("GBM Recall : %5.5f" %(round(recall_score(y_test,gbm_up_pred),3)))
print("GBM Precision : %5.5f" %(round(precision_score(y_test,gbm_up_pred),3)))
gbm_up_auc = roc_auc_score(y_test, gbm_up_pred)
print("GBM AUC : %5.5f" %(gbm_up_auc))
```

```
GBM f1 score : 0.04600
GBM accuracy : 0.91900
GBM log loss : 2.78300
GBM Recall : 0.02400
GBM Precision : 0.52200
GBM AUC : 0.51117
```

Plot ROC AUC Curve

In [41]:
```
plt.style.use('default');
figure = plt.figure(figsize=(10, 6));
ax4 = plt.subplot(1, 1, 1);
plot_roc(gbm_up_classifier, X_test_final, y_test, "GBM_Test", ax4)
plot_roc(gbm_up_classifier, X_train_up, y_train_up, "GBM_Train", ax4)
plt.legend(loc='lower right', fontsize=18);
plt.tight_layout();
```



In [ ]: