Project Report

PROJECT DESCRIPTION

Domain Background - Which customers are happy customers?

This project is trying to solve a problem from Kaggle sponsored by Santander Bank.
https://www.kaggle.com/c/santander-customer-satisfaction
Santander Bank posts hundreds of lines of data with anonymized features onto Kaggle and ask to help them predict if a customer is satisfied or dissatisfied based on their banking experience (data with anonymized features). The analysis on the provided data would be valuable for the bank. Based on the predicted result, the bank could probably use the result and take several approaches to improve their customer experience to keep their customers and make their customer happy.

Problem Statement

Since Santander Bank only wants to understand if their customers are satisfied or dissatisfied with the banking experience, this should be a classification problem to classify the customers into two different groups: the satisfied group and the dissatisfied group, which we could use 1 and 0 respectively to represent. In order to solve the problem and get an optimized classification result, it would require the effort to follow the procedure below:
Performance Metrics Selection -> Data Exploration and Cleaning -> Feature Selection and Transformation -> Model Training and Selection -> Model Tuning -> Performance Analysis

By following the procedure, we could use the train dataset to train the model that we select and use the test dataset the evaluate the performance. The optimized trained model could be used to predict if the customers are satisfied once we feed it with new customers' data.

GET THE DATA

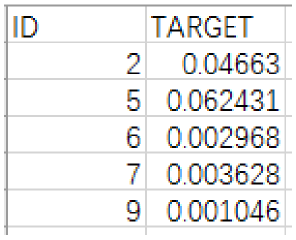
The data can be downloaded from Kaggle site. The datasets are also included during the project submission.
https://www.kaggle.com/c/santander-customer-satisfaction/data
The anonymized dataset containing a large number of numeric variables.

The data package includes

- train.csv - the training set including the target. The train.csv includes the target column which is the variable to predict. It equals 1 for unsatisfied customers and 0 for satisfied customers.
- test.csv – the test set without the target
- sample_submission.csv – a sample submission file in the correct format

PERFORMANCE METRICS

Area Under the ROC Curve is selected to evaluate the performance of model in this project and this metric was used to evaluate submissions on Kaggle.
https://www.kaggle.com/c/santander-customer-satisfaction/details/evaluation
The task for this project is to predict the probability that each customer in the test set is an unsatisfied customer. The final submission.csv file should have the following format.



ID	TARGET
2	0.04663
5	0.062431
6	0.002968
7	0.003628
9	0.001046

Area Under the ROC Curve

TP & FP

Given a classifier and an instance, there are four possible outcomes. If the instance is positive and it is classified as positive, it is counted as TP(true positive, also called hit rate and recall); if it is classified as negative, it is counted as a FN(false negative). If the instance is negative and it is classified as negative, it is counted as a TN(true negative); if it is classified as positive, it is counted as a FP(false positive, also called false alarm rate).

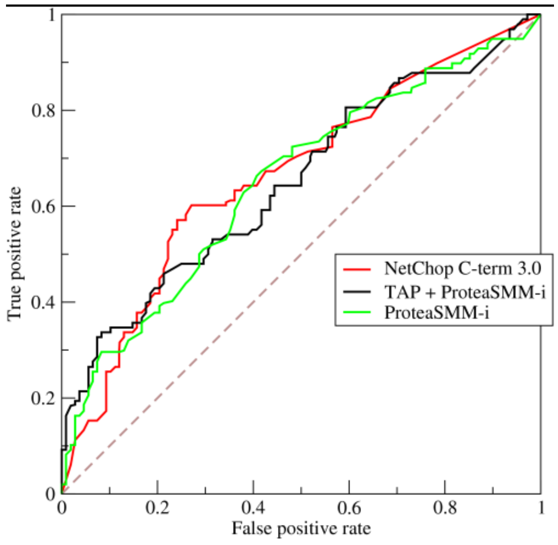
TP rate = TP/total positives, FP rate = FP/total negatives

ROC Curve

ROC Graphs are two-dimensional graphs in which TP rate is plotted on the Y axis and FP rate is plotted on the X axis. The ROC curve shows the tradeoff between TP(benefits) and FP(costs).

For the given classifier, we set multiple thresholds to split a test set into several parts. For each sub-test set, we could get the TP and FP rate with the classifier. Then the ROC curve could be created by plotting the points (accumulative TP and FP rate).

Please find the sample ROC curve below from (https://en.wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:Roccurves.png).



Area under ROC

The diagonal line y=x represents the randomly guessing of a test set. The AUC is only 0.5, which means random guessing, which is the worst case.

The closer the curve follows the left-hand border and the top border, the more accurate the test. The optimal AUC value would be 1.

The result we get from the solution for this project is measured based on AUC score.

BENCHMARK

There are overall 5000 competitors joined the Santander bank competition and their final results (AUC score) are listed on the leaderboard on Kaggle. Our goal is to get an AUC score that would place us among the top 25% of the competitors. The benchmark score is 0.826120.

IMPLEMENTATION

We mainly follow the below procedure to implement the code.

1. Preliminary Data Exploration
    a. Read in both training data and testing by using Pandas Dataframe.
    ```
    train = pd.read_csv("train/train.csv")
    test = pd.read_csv("test/test.csv")
    ```
    b. Use matplotlib.pyplot histogram to visualize the number of satisfied customers versus. unsatisfied customers
    ```
    plt.hist(train['TARGET'],np.arange(0,1.5,0.3),
    alpha=0.5,color='r')
    plt.title("Customer Satisfaction")
    plt.xticks(np.arange(3)+0.10,('Satisfied','Unsatisfied'))
    plt.ylabel("Number of Customers")
    plt.show()
    ```
    c. Go through each column to check if the column has only one constant value or the column is the same as other columns, then drop the columns from the initial tables
    ```
    #Look for columns with constant value
    for col in train.columns:
      first = train.ix[0,[col]].values
      if (train[col].values == first * np.ones(len(train)).astype(float)).all():
        removed_col.append(col)

    #Look for duplicated columns
    for i in xrange(len(columns)-1):
      for j in range(i+1,len(columns)):
        if (train[columns[i]].values == train[columns[j]].values).all():
          removed_same_col.append(columns[i+1])

    train.drop(removed_same_col,axis=1,inplace=True)
    test.drop(removed_same_col,axis=1,inplace=True)
    ```

2. Feature Scaling

    I found that implementing feature scaling is helpful for improving the outcome of training the model. Initially, I didn't implement feature scaling and the AUC score is low since the model is affected by features with larger number.

    Go through each column and use MinMaxScaler function to normalize the dataset to (0,1)
    ```
    mms = MinMaxScaler()
    X_all_norm = pd.DataFrame(mms.fit_transform(X_all),columns=X_all.columns)
    ```

```
X_train_norm = pd.DataFrame(mms.fit_transform(X_train),columns=X_train.columns)
X_test_norm = pd.DataFrame(mms.fit_transform(X_test),columns=X_test.columns)
kaggle_test_norm = pd.DataFrame(mms.fit_transform(kaggle_test),columns=kaggle_test.col
umns)
display(X_train_norm.describe())
```

3. Feature Selection

The code below implements feature selection by using Random Forest Classifier to decide if a feature is important measured by entropy.

```
feat_labels = X_train.columns
forest = RandomForestClassifier(n_estimators=130,criterion="entropy",random_state=0,n_j
obs=-1)
forest.fit(X_train_norm,y_train)
importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]
```

SelectFromModel() function select the most important features among all the features.

```
featSelected = SelectFromModel(forest, prefit=True)
feature_boolean_array = featSelected.get_support()
feature_selected = pd.DataFrame(np.array(X_all_norm.columns[feature_boolean_array]),col
umns=["Selected Features"])
```

Then use featSelected.transform() function to transform the datasets based on the new sets of features. **Complications: it is also important here to also transform the dataset type to pd.DataFrame to make sure everything has the same data type. FeatSelected.transform() generate a list of data instead of Dataframe, which will cause problems later when calculating the AUC score by using roc_auc_score. I spent some time here to debug the problem.**

```
X_all_norm = pd.DataFrame(featSelected.transform(X_all_norm.values))
X_train_norm = pd.DataFrame(featSelected.transform(X_train_norm.values))
X_test_norm = pd.DataFrame(featSelected.transform(X_test_norm.values))
kaggle_test_norm = pd.DataFrame(featSelected.transform(kaggle_test_norm.values))
```

4. Model Selection

Predefine a function called train_predict() to calculate different models' running time and their AUC score. Initialize and run different models and compare their performance.

```
Dtree = DecisionTreeClassifier(random_state = 42)
KNN = neighbors.KNeighborsClassifier()
RandomForest = RandomForestClassifier(random_state = 42)
LogisticReg = linear_model.LogisticRegression(random_state = 42)
GBoost = GradientBoostingClassifier(random_state = 42)
GNB = GaussianNB()
Adaboost = AdaBoostClassifier(random_state = 42)
Xgboost = xgb.XGBClassifier()
```

Plot ROC curves for different models

```
clfList[i].fit(X_train_norm,y_train)
```

```
y_pred = clfList[i].predict_proba(X_test_norm)
fpr, tpr, thresholds = roc_curve(y_test, y_pred[:,1])
```

5. Model Tuning

We focus on tuning XGBoost model on this part. The model is selected based on the performance result from the previous model selection section. We use GridSearchCV function to go through every combination of parameters to find out the optimal parameters for the model which may get us a higher AUC result. **There are a lot of complications here. Even though we use grid search method, it is still not guaranteed to find the optimal sets of parameters. We have to go through a lot of online resources to really dig into mathematical theorem of XGBoost and learn from other people's experience to find out the tricks of tuning XGBoost models. We will discuss this part on the later section.**

Define initial values for each parameter and the XGBoost classifier.

```
param_initial = {'n_estimators':[200],
          'learning_rate': [0.1],
          'max_depth':[5],
          'min_child_weight':[1],
          'subsample': [0.80],
          'colsample_bytree': [0.80],
          'gamma':[0]
         }
clf_initial = xgb.XGBClassifier(n_estimators=200, seed=42, sile
nt=False)
```

Then implement GridSearchCV to go through each combination of parameters to find out the best estimator and the best AUC score

```
tuned_XGB_initial = GridSearchCV(clf_initial,param_initial,scor
ing="roc_auc",cv=10)
tuned_XGB_initial.fit(X_train_norm,y_train.values)
print "XBG_1 Best Estimator: ", tuned_XGB_initial.best_estimato
r_
print "best score: ",tuned_XGB_initial.best_score_
```

Each iteration we will try different sets of parameters and check the outcome AUC score for both training dataset and testing dataset. Please see the sample code below

```
param_1 = {'n_estimators':[50, 100, 200, 300]}

#AUC score for training data
y_train_pred = tuned_XGB_initial.predict_proba(X_train_norm)
score = roc_auc_score(y_train.values,y_train_pred[:,1])
print "score for training set: ", score

#AUC score for testing data
y_test_pred = tuned_XGB_initial.predict_proba(X_test_norm)
score_test = roc_auc_score(y_test.values, y_test_pred[:,1])
print "score for testing set: ", score_test
```

6. <u>Performance Evaluation</u>

Once we get the final classifier with the optimal parameter values we think, we try to evaluate the model by using different sizes of training/testing datasets and check the AUC scores to see if the model is overfitting and able to generalize the datasets.

We define a function called performance_metric to calculate the AUC score.

```
def performance_metric(y_true, y_pred):
    score = roc_auc_score(y_true,y_pred[:,1])
    return score
```

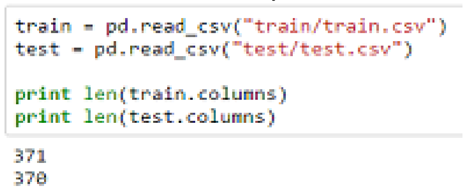
And define another function called leaning_curves to calculate to feed the model with different sizes of datasets and calculate the AUC scores, then plot the graph.

```
def learning_curves(X, y, X_test, y_test, clf):
    print ("Creating learning curve graphs for train and test d
ata set")
    # Create the figure
    plt.figure(figsize=(12,8))

   # Create different sizes of datasets
    sizes = [1000,5000,10000,15000,20000,34000,48000,55000,6200
0,76020]
    train_score = np.zeros(len(sizes)) # score of the training
set
    test_score = np.zeros(len(sizes)) # score of the test set

    # Create different models
    for i, s in enumerate(sizes):
            print "size: ",s
            #scores = cross_validation.cross_val_score(clf, X[:
s], y[:s],scoring='roc_auc', cv=10)


            # Find the performance on the testing set
            # use all the training data
            clf.fit(X[:s], y[:s])
            y_pred = clf.predict_proba(X[:s])
            #print y_pred
            train_score[i] = performance_metric(y[:s], y_pred)
            print ("train roc_auc: {}".format(train_score[i]))

            y_pred_test = clf.predict_proba(X_test)
            test_score[i] = performance_metric(y_test, y_pred_t
est)
            print ("test roc_auc: {}".format(test_score[i]))

    plt.plot(sizes, train_score, lw = 2, label = 'Training set
')
    plt.plot(sizes, test_score, lw = 2, label = 'Testing set')
    plt.legend(loc="best",fontsize=15)
    plt.title('Learning Curve', fontsize=18)
    plt.xlabel('Number of Data Points')
    plt.ylabel('AUC score')
```

PRELIMINARY DATA EXPLORATION

1. Looking at the raw data in the excel sheet, the meaning of each variable is unclear since the names are not given.

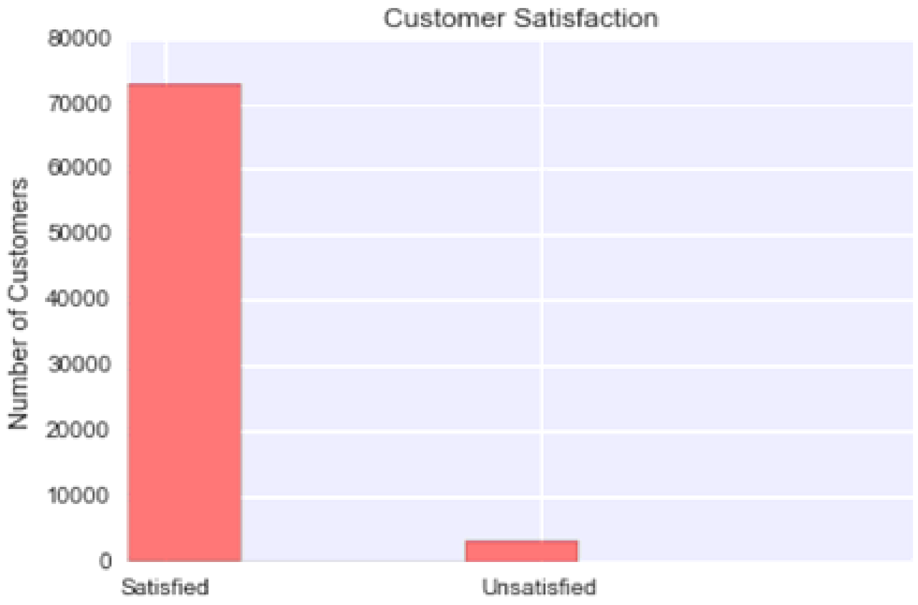
   Load both training data and testing data. We can see that the training data has 371 columns and the testing data has 370 columns, which means that we have to work on the feature Engineering to decrease the number of columns and find out the golden features to increase the accuracy of our model.



```
train = pd.read_csv("train/train.csv")
test = pd.read_csv("test/test.csv")

print len(train.columns)
print len(test.columns)
```

```
371
370
```

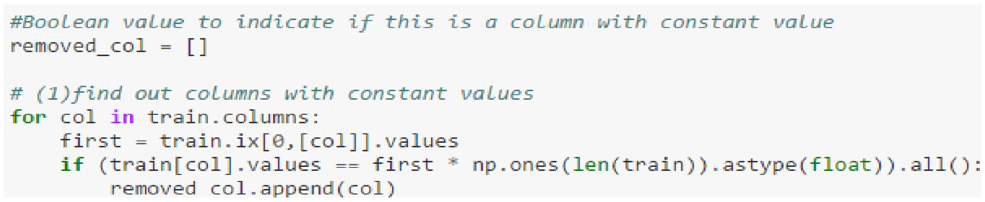
2. Below is the basic information of the data. Total number of training data set is 76020. Number of features is 369 (exclude “TARGET” and “ID”).
   The histogram below shows the number of satisfied customers vs. number of unsatisfied customers. We can see that the majority of the customers are satisfied customers. Only a small fraction of customers who are not satisfied.



3. **Through the preliminary observation of the training dataset, we can find that some columns have constant values (i.e. value of 0).** We can find out all these columns and

remove them since their information gain is 0 if we use them as features to classify the data (i.e. decision tree classifier). They have no input to the classification model.

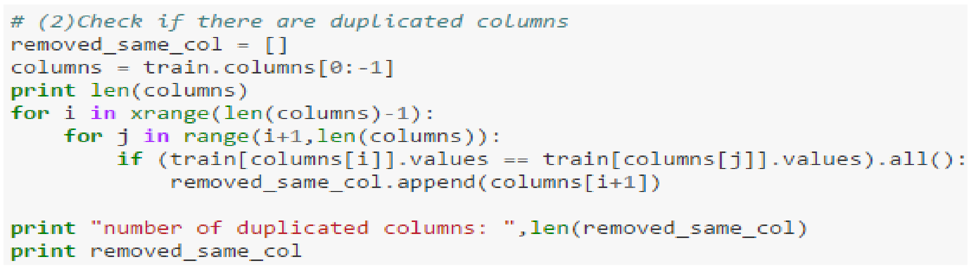
Find out all the columns with constant values.



```
#Boolean value to indicate if this is a column with constant value
removed_col = []

# (1)find out columns with constant values
for col in train.columns:
    first = train.ix[0,[col]].values
    if (train[col].values == first * np.ones(len(train)).astype(float)).all():
        removed_col.append(col)
```

We can see that there are 34 columns with constant value. We will remove all these columns from the datasets.

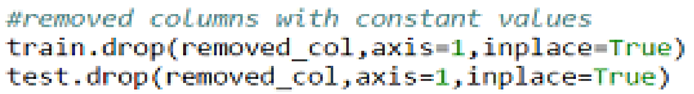
4. **We can also check if there are any columns which are duplicated.** Find out all these duplicated columns and remove them would be helpful to decrease the number of features and lower the computing time.



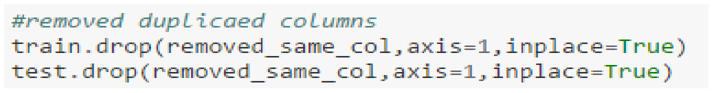
```
# (2)Check if there are duplicated columns
removed_same_col = []
columns = train.columns[0:-1]
print len(columns)
for i in xrange(len(columns)-1):
    for j in range(i+1,len(columns)):
        if (train[columns[i]].values == train[columns[j]].values).all():
            removed_same_col.append(columns[i+1])

print "number of duplicated columns: ",len(removed_same_col)
print removed_same_col
```

We can see that there are 29 columns are duplicated columns. We will remove all these columns from the datasets.

5. **Remove columns found in step 3 and 4 above.**



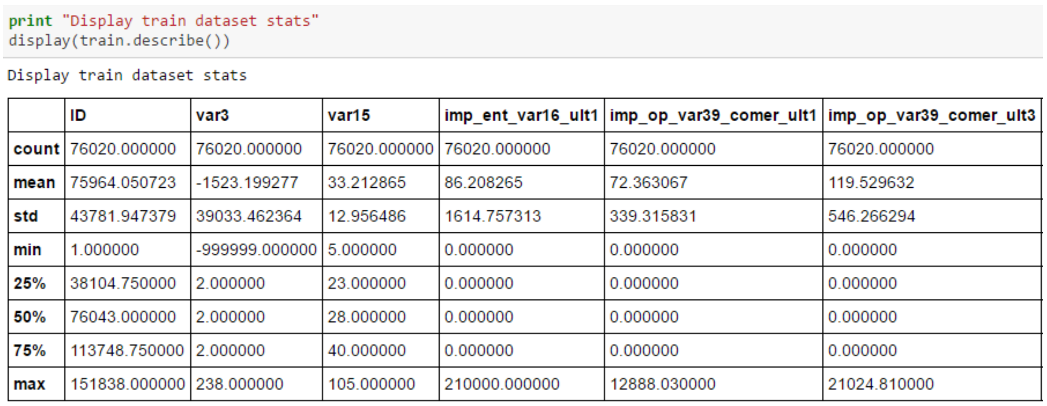
```
#removed columns with constant values
train.drop(removed_col,axis=1,inplace=True)
test.drop(removed_col,axis=1,inplace=True)
```



```
#removed duplicaed columns
train.drop(removed_same_col,axis=1,inplace=True)
test.drop(removed_same_col,axis=1,inplace=True)
```

FEATURE SCALING

Display the statistics of the datasets. Please find part of the data description below.



```
print "Display train dataset stats"
display(train.describe())
```

```
Display train dataset stats
```

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3
count	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000
mean	75964.050723	-1523.199277	33.212865	86.208265	72.363067	119.529632
std	43781.947379	39033.462364	12.956486	1614.757313	339.315831	546.266294
min	1.000000	-999999.000000	5.000000	0.000000	0.000000	0.000000
25%	38104.750000	2.000000	23.000000	0.000000	0.000000	0.000000
50%	76043.000000	2.000000	28.000000	0.000000	0.000000	0.000000
75%	113748.750000	2.000000	40.000000	0.000000	0.000000	0.000000
max	151838.000000	238.000000	105.000000	210000.000000	12888.030000	21024.810000

We can find that the mean and median values among all these features vary significantly, which means the data is not normally distributed and indicates a large skew. Perform the feature scaling would be helpful to reduce skewness.

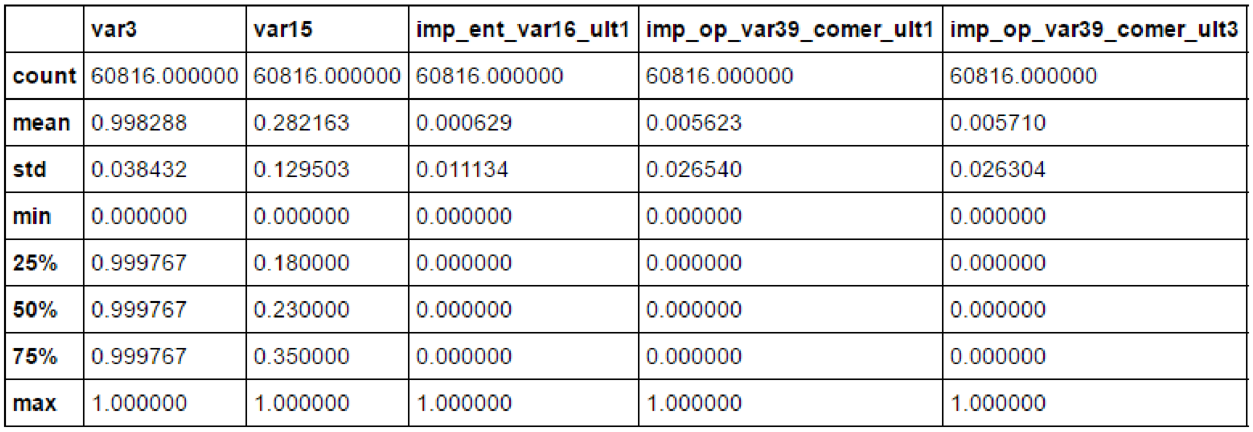
For example, if we compare “var3” with “var15”,

- “var3” max value is 238, min value is -999999
- “var15” max value is 105, min value is 5

The difference is huge so that it would be better for us to normalize the data into the same scale to speed up the training time.

The dataset is then normalized to range (0,1) by using MinMaxScaler() function. MinMaxScaler() scales and translates each features individually such that it is in the given range on the training set (between 0 and 1 here).

Please find the formula $x' = \frac{x - Min}{Max - Min}$



	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3
count	60816.000000	60816.000000	60816.000000	60816.000000	60816.000000
mean	0.998288	0.282163	0.000629	0.005623	0.005710
std	0.038432	0.129503	0.011134	0.026540	0.026304
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.999767	0.180000	0.000000	0.000000	0.000000
50%	0.999767	0.230000	0.000000	0.000000	0.000000
75%	0.999767	0.350000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

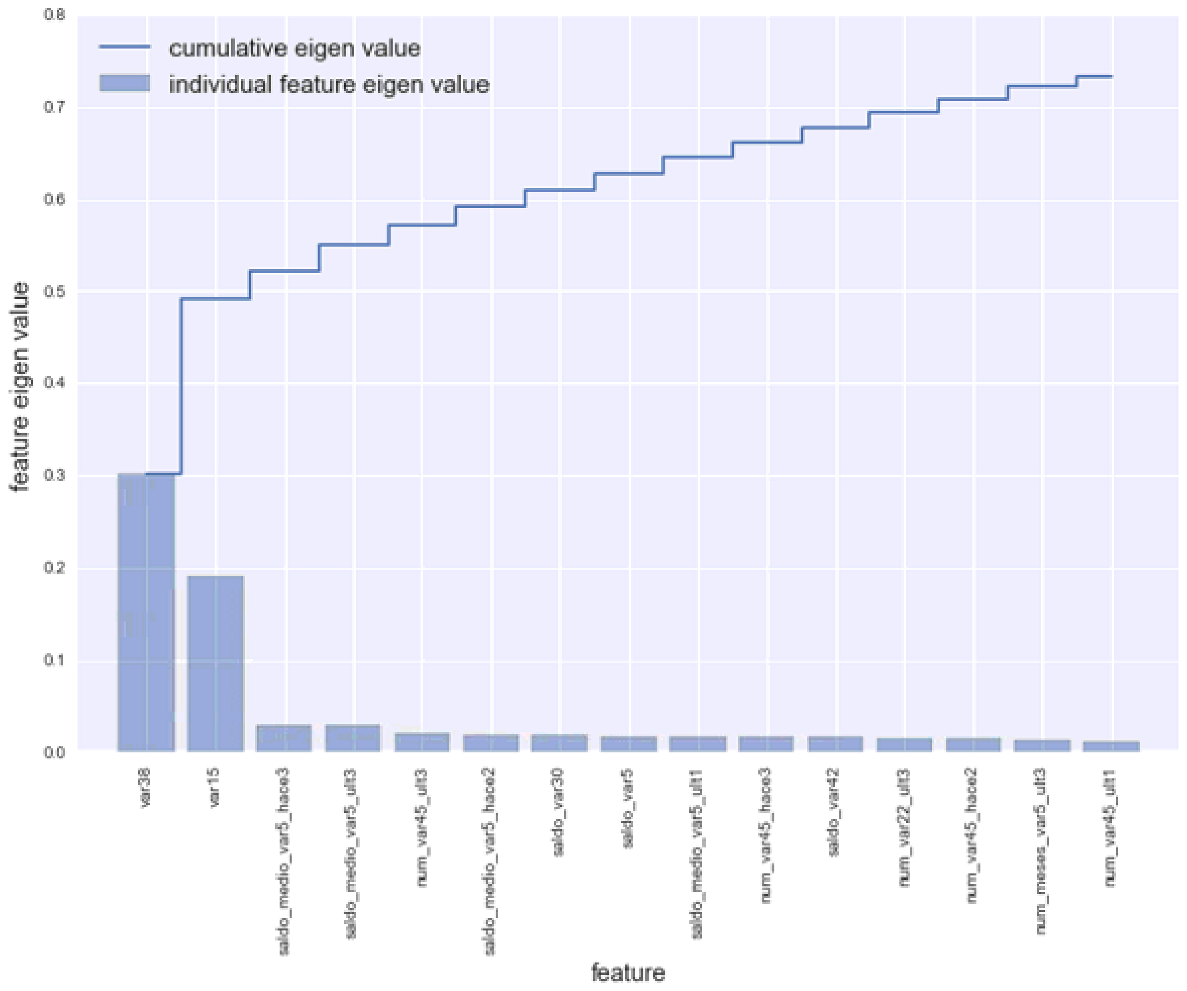
FEATURE SELECTION AND TRANSFORMATION

Perform a basic Random Forest Classifier to try to classify the dataset and find out the importance of features.

RandomForestClassifier(n_estimators=130, criterion="entropy",random_state=0,n_jobs=-1)

The number of trees is set to 130 in order to get a lower error rate. Since we don't put an emphasis on the time of speed, we could set a relatively larger number of trees. The criterion is set to "entropy" which allows to measure the average information gain of each feature in each tree.

The importance of a feature is computed as the (normalized) total reduction of the criterion (information gains here) brought by the feature. Please find the feature importance graph below



Random Forest Classifier provides a way to determine the importance of each feature. We can find that

- the most two important features are var38 and var15, which explain 0.300 and 0.186 variance respectively.
- Features, which rank lower than 246th, explain 0 importance. The first 50 features explain over 0.909 importance.

Sklearn SelectFromModel is used for feature selection. According to sklearn documentation, (http://scikit-learn.org/stable/modules/feature_selection.html) SelectFromModel is a meta-transformer that can be used along with any estimator that has a feature_importances_ attribute. If the feature importance value is below the provided threshold parameter, the features are considered unimportant and removed.

In our case, the threshold is set to default value (None). According to the source code of SelectFromModel, if the threshold is "None", the threshold used is 1e-5. "prefit" parameter is set to "True" meaning the estimator is trained already.

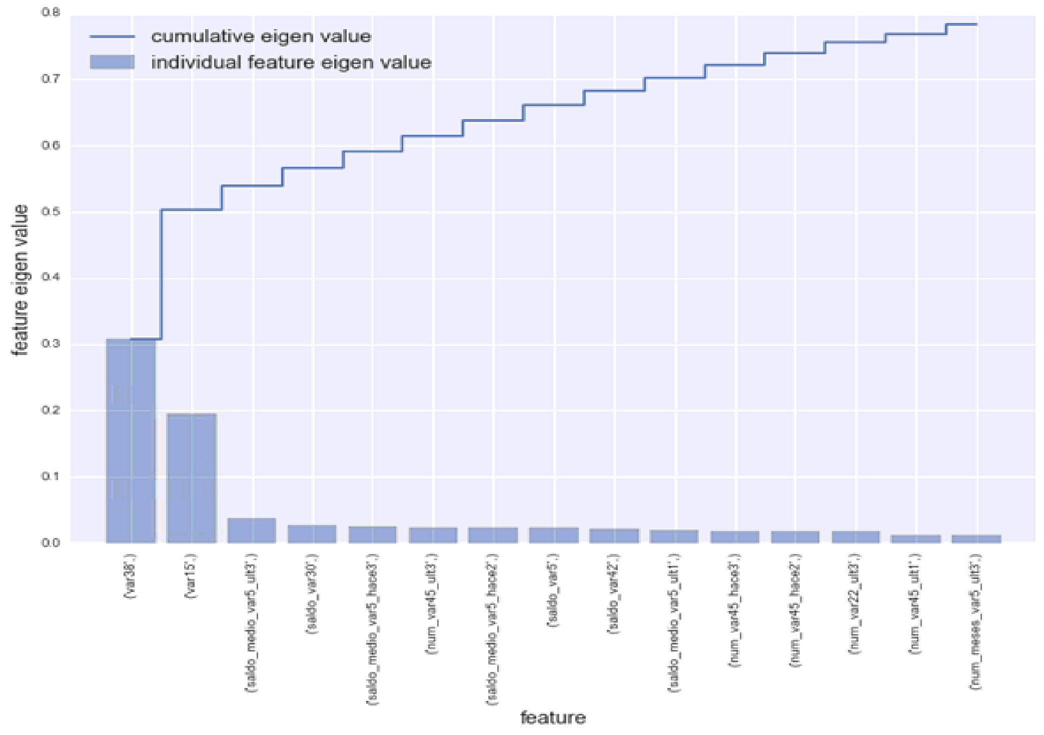
```
featSelected = SelectFromModel(forest, prefit=True)
```

The transformed dataset now has only 43 columns.

```
print X_all_norm.shape
```

(76020L, 43L)

Please find Feature Importance Graph after feature transformation below



Model Selection

Different classifiers have been tested and fit with the dataset and we compare the performance of these classifiers. Following are the chosen classifiers.

- Decision Tree Classifier
- K-Nearest Neighbors
- Random Forest Classifier
- Logistic Regression
- Gradient Boosting Classifier
- Gaussian Naïve Bayes Classifier
- Adaboost Classifier
- Xtreme Gradient Boosting Classifier

All of these classifiers are untuned. They run on all 76020 training data with features selected and calculate AUC score each with 10 cross-validations.

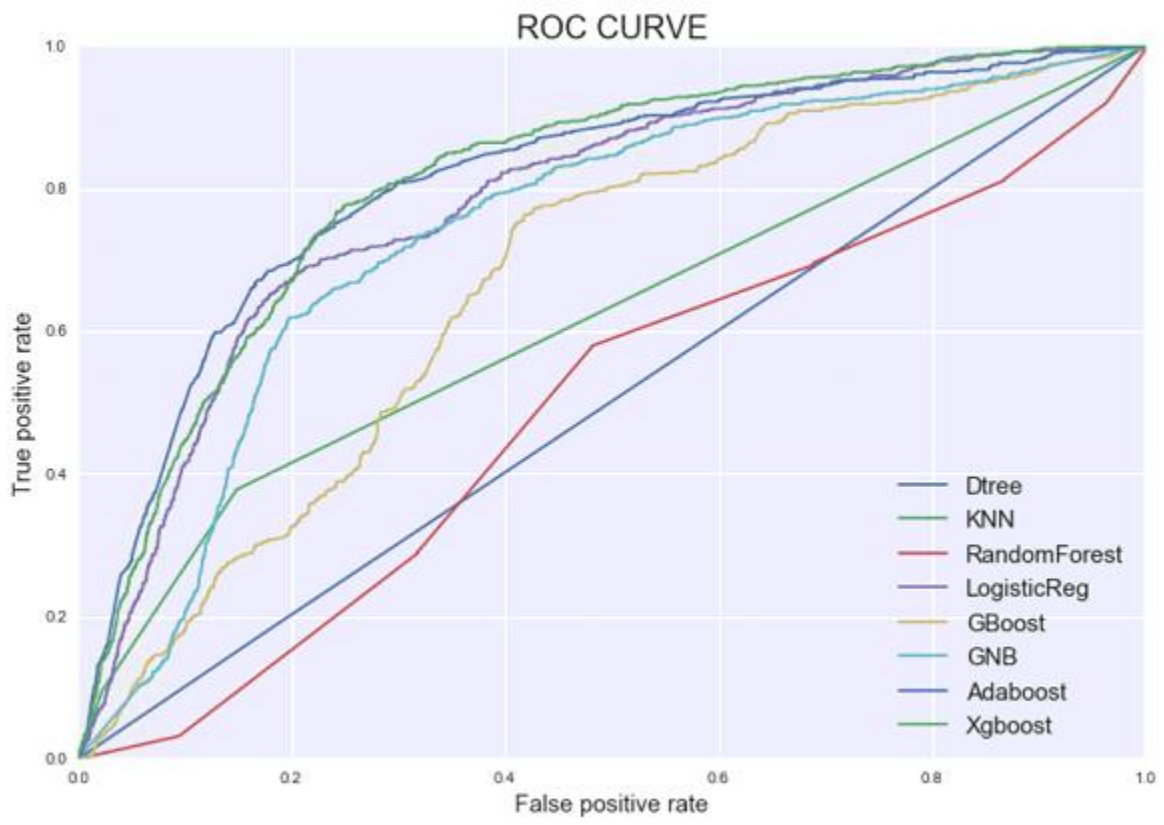
Please find the running result below.

Classifier	Running Time (sec)	AUC Score (training set)
DecisionTreeClassifier	11.9210000038	0.568994373645
KNeighborsClassifier	177.731000185	0.630864338013
RandomForestClassifier	12.0920000076	0.676971558132
LogisticRegression	7.78699994087	0.775200210694
GradientBoostingClassifier	345.523999929	0.83516586737
GaussianNB	1.32899999619	0.748369007158
AdaBoostClassifier	52.4869999886	0.824331657752
XGBClassifier	19.5509998798	0.837810418277

According to the result table above, running with 10 cross-validations, we can find that the top three classifiers with good AUC score are Xgboost Classifier, Gradient Boosting Classifier, and Adaboosting Classifier. Xgboost Classifier gets the best AUC score with 0.8378.

Decision Tree, Random Forest and GaussianNB have short running time. However, the AUC score for these three classifiers are very low. XGBClassifier has a shorter running time compared to Adaboost Classifier and Gradient Boosting Classifier.

Below is the ROC curve for different model.



Also, from the ROC curve above, we can find that the Decision Tree has the worst performance. Xgboost, Gradient Boosting and Adaboost has the relatively good performance.

XGBClassifier is chosen as the classifier for further exploration. Below is the information for basic untuned XGBClassifier.

XGBClassifier(base_score=0.5,colsample_bylevel=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,nthread=-1,objective='binary:logistic',reg_alpha=0,reg_lambda=1,scale_pos_weight=1, seed=42, silent=False, subsample=1)

Untuned AUC score: 0.8378

XGBOOST CLASSIFIER

XGBoost is a scalable tree boosting system which is proposed by Tianqi Chen and Carlos Guestrin from University of Washington (https://arxiv.org/pdf/1603.02754v3.pdf). XGBoost is available on github as an open source package (https://github.com/dmlc/xgboost).

XGBoost is widely recognized in a number of machine learning challenges and many of the winning solutions use XGBoost. It is a gradient tree boosting with regularization (variant of original GBM). XGBoost, compared to other tree-based model, is a faster tool for learning better models.

Several things that enable XGBoost to give such great results are as follows,

1. **Regularization term is included in the objective function** while most of the common objective functions in most tree packages only contain training loss function.
   Obj(Θ)=L(θ)+Ω(Θ)

   Regularization term **Ω(Θ)** controls the complexity of the model and helps smooth the final learnt weights to avoid overfitting, so that XGBoost can achieve a better generalization result. In XGBoost, the complexity is defined as

   $$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

   Where **w** is the vector of scores on leaves, **T** is the number of leaves

2. **Introducing additive training (boosting) in the formula**, where t is the number of steps

   $$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

   taking the Taylor expansion of the loss function up to the second order

   $$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

   Where ***gi*** is the first derivative of the loss function and ***hi*** is the second derivative of the loss function. And we can find that the optimization for the new tree only depends on ***gi*** and ***hi****.*

3. **Searching for single tree with optimal tree structure.** Enumerate the possible tree structures and find the best tree structure with the optimal leaf weight. Giving structure score to measure how good a tree structure is (the smaller the score the better), assuming the structure is fixed.

   $$\text{obj}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

4. **In a single tree, find the best split in each tree that gives the most reduction to the objective function.** Similar to the information gain (entropy) in decision tree. The gain function is defined as follow

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

   A new split is added based on the fact that if the gain is larger than γ. This would help to find the optimal split.

5. **XGBoost also do systems optimizations,** including out of core computing, cache optimization, distributed computing to speed up the model training time.

XGBOOST PARAMETERS

Parameter Tuning is a very critical step during model training. It is very important for us to understand the tradeoff between bias and variance.

Here are some of the parameters which are considered during tuning in order to get a better result.

n_estimators: number of boosted trees to fit

learning_rate (eta): range [0,1], step size shrinkage used in update

max_depth: maximum depth of a tree, the model is more complex when increase this value

min_child_weight: minimum sum of instance weight needed in a child node

subsample: randomly collecting certain percentage of data instances to grow trees

colsample_bytree: subsample ratio of columns when constructing each tree

gamma: range [0, ∞], minimum loss reduction required to make a further partition on a leaf node of the tree. The model will be more conservative with larger gamma value

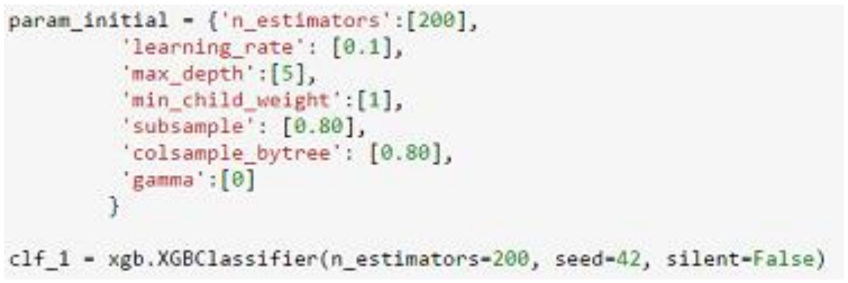
According to XGBoost documentation, in general, there are two ways to control overfitting in XGBoost. First, to tune max_depth, min_child_weight and gamma parameters to control model complexity. Second, to tune subsample, colsample_bytree parameters to add randomness

MODEL TUNING

We need to tune the Xgboost parameters to get a better AUC score.

1) The initial values of parameters are set below. These are just the initial values and will be changed during the tuning process.
   1. learning_rate: using the default value 0.1
   2. max_depth: initial depth is 5. This usually should be between 3-10.

3. min_child_weight: initial weight is 1. Too high value can lead to under-fitting.
4. gamma: using default value 0. This will be tuned later.
5. subsample: starting at 0.8. This value usually ranges between 0.5-0.9
6. colsample_bytree: starting at 0.8.
7. n_estimators: starting at relatively low as 200



```
param_initial = {'n_estimators':[200],
        'learning_rate': [0.1],
        'max_depth':[5],
        'min_child_weight':[1],
        'subsample': [0.80],
        'colsample_bytree': [0.80],
        'gamma':[0]
       }

clf_1 = xgb.XGBClassifier(n_estimators=200, seed=42, silent=False)
```

<u>The initial AUC scores for both training dataset and testing dataset are shown as below:</u>

score for training set: 0.911157766469
score for testing set: 0.750680178946
We can find that the training set's score is quite high with over 0.91 while the score for the testing set is relatively low. We need to further tune the model to increase the score for testing set and lower the gap between both scores.

2) **We then try to apply GridSearchCV along with 10-fold cross-validation to find the optimal number of estimators which affects the boosting operation.**

   param_1 = {'n_estimators':[50, 100, 200, 300]}

```
{'n_estimators': 50} 0.839413728405
{'n_estimators': 100} 0.83967865784
{'n_estimators': 200} 0.835254733612
{'n_estimators': 300} 0.830430274021
```

When learning_rate=0.1, we can find that the optimal number of estimators is close to 100.
<u>The AUC scores are shown as below:</u>

score for training set: 0.888047100139
score for testing set: 0.757664395599
The score for training set is lower than before but the score of the testing is increasing. The gap is getting smaller

3) **Next we will start tuning the trees parameters (max_depth and min_child_weight). First we will perform a wider range, then we perform other iterations to search the optimal parameters. Tree parameters may have the highest impact on the final scores.**

1st step:

param_2 = {'max_depth':range(3,10,2), 'min_child_weight':range(1,6,2)}

we can find that the best combination of parameters are shown as below.

{'max_depth': 5, 'min_child_weight': 5} **0.841661002809**

2nd step:

Based on the parameters we received from the 1st step, we perform another iteration to find the optimal ones.

param_2b = {'max_depth': [4,5,6], 'min_child_weight': [4,5,6]}

The result for each combination of parameters are shown as below.

```
{'max_depth': 4, 'min_child_weight': 4} 0.841745385028
{'max_depth': 4, 'min_child_weight': 5} 0.841984147933
{'max_depth': 4, 'min_child_weight': 6} 0.842063680243
{'max_depth': 5, 'min_child_weight': 4} 0.841316533328
{'max_depth': 5, 'min_child_weight': 5} 0.841661002809
{'max_depth': 5, 'min_child_weight': 6} 0.841233196642
{'max_depth': 6, 'min_child_weight': 4} 0.840472391771
{'max_depth': 6, 'min_child_weight': 5} 0.841538565983
{'max_depth': 6, 'min_child_weight': 6} 0.841218308058
```

The best score is **0.842063680243** with max_depth = 4, min_child_weights = 6

The AUC scores are shown as below:

```
score for training set:  0.867979876835
score for testing set:  0.764197194832
```

We can see that the AUC score for the testing set is further improved compared to the previous results.

3rd step:

According to the final result from the previous step, we can see that when max_depth = 4, the AUC increases by increasing the min_child_weight:

```
{'max_depth': 4, 'min_child_weight': 4} 0.841745385028
{'max_depth': 4, 'min_child_weight': 5} 0.841984147933
{'max_depth': 4, 'min_child_weight': 6} 0.842063680243
```

Then we could further try values more than 6 for max_child_weight since we haven't tried any values larger than 6. The max_child_weight parameter is used to control over-fitting and make t he model more conservative.

param_2c = {'min_child_weight':[6,8,10,12]}

Here we can find that when min_child_weight = 8, the model gets the highest score.
{'min_child_weight': 6} 0.842063680243
{'min_child_weight': 8} **0.842826262093**
{'min_child_weight': 10} 0.842395701415
{'min_child_weight': 12} 0.842407922168

The resulting score for both training data and testing data are shown as below.
score for training set: 0.867626782284
score for testing set: 0.771030767826
We can also see that the AUC score for testing set is further improved.

4) **Then we will start tuning gamma.**
param_3 = {'gamma':[0,0.1,0.2,0.3,0.4,0.5]}

Below are the AUC scores for different gamma values.
{'gamma': 0} **0.842826262093**
{'gamma': 0.1} 0.842584795051
{'gamma': 0.2} 0.842565085909
{'gamma': 0.3} 0.842736708966
{'gamma': 0.4} 0.842555866268
{'gamma': 0.5} 0.842315214036

We can find that the model gets the highest AUC score when gamma = 0. Here are the AUC score for both training set and testing set.
score for training set: 0.867626782284
score for testing set: 0.771030767826

5) **Next we will tune subsample and colsample_bytree. We will tune the parameters by thre e steps.**
1st step:
Try the range of values with large intervals.
param_4 = {'subsample':[0.6,0.7,0.8,0.9,1.0], 'colsample_bytree':[0.6,0.7,0.8,0.9,1.0]}

The best combination is shown as below
{'subsample': 0.8, 'colsample_bytree': 0.6} **0.842860555346**

AUC score for both training dataset and testing dataset are shown as below.
score for training set: 0.866110533714
score for testing set: 0.783992912901

2st step:

Based on the result from the 1st step, try values in 0.05 interval around the result.
param_4b = {'subsample':[0.75,0.8,0.85],'colsample_bytree':[0.55,0.6,0.65]}

The best combination is still subsample = 0.8, colsample_bytree = 0.6

AUC score for both training dataset and testing dataset are shown as below.

3rd step:

Based on the result from the 2nd step, try values in 0.02 interval around the result.
param_4b = {'subsample':[0.78,0.8,0.82],'colsample_bytree':[0.58,0.6,0.62]}

The best combination is still **subsample = 0.8, colsample_bytree = 0.6**

AUC score for both training dataset and testing dataset are shown as below.
score for training set: 0.866110533714
score for testing set: 0.783992912901

6) Lastly, we will lower the learning rate and increase the number of estimators

param_5 = {'learning_rate':[0.01, 0.02, 0.1], 'n_estimators':[100, 550, 750]}

The best combination is
{'n_estimators': 550, 'learning_rate': 0.02} 0.842961036332

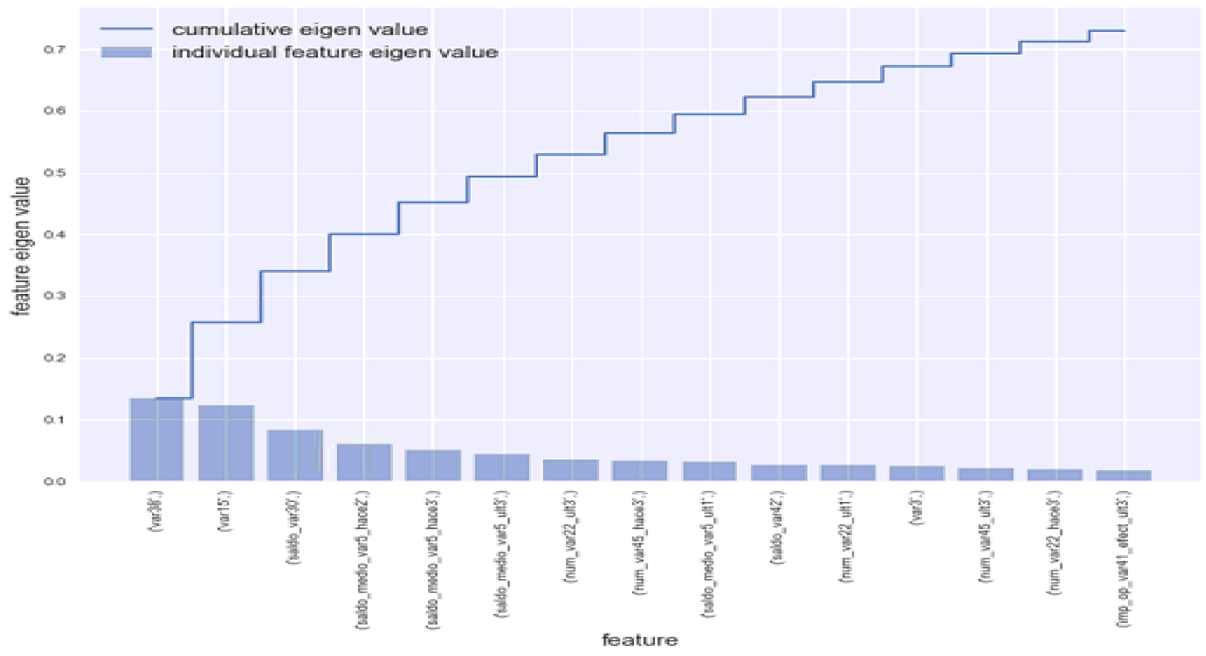
AUC score for both training dataset and testing dataset are shown as below.
score for training set: 0.869095774068

score for testing set: 0.77726798594

The final tuned XGBClassifier is shown as follow

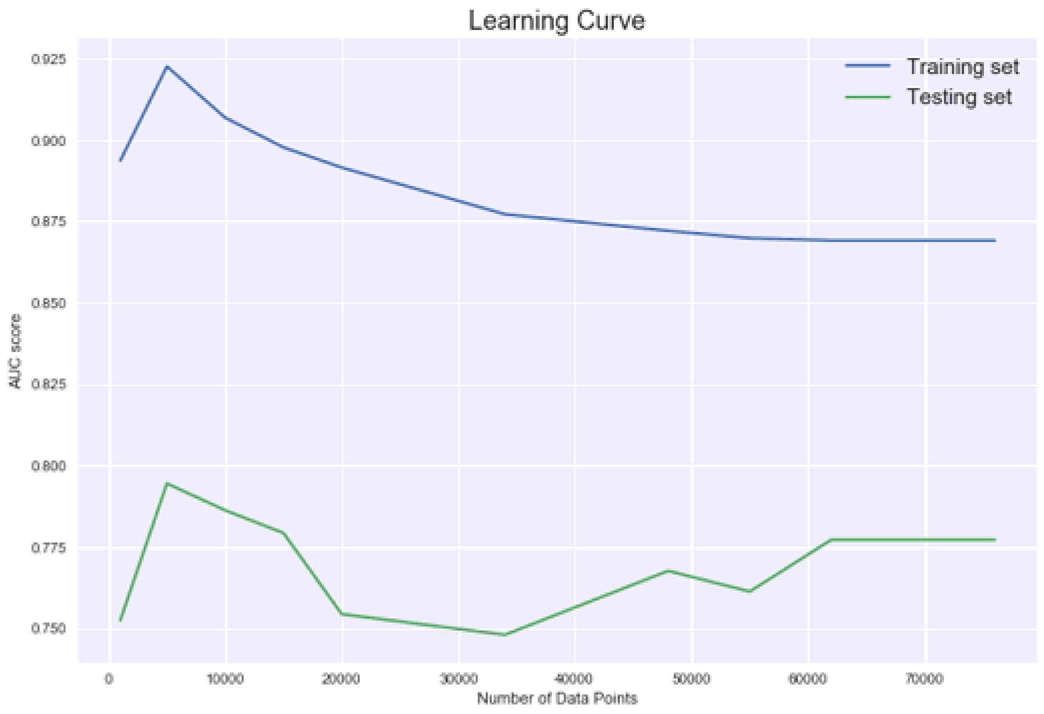
```
XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=0.6,
              gamma=0, learning_rate=0.02, max_delta_step=0, max_depth=4,
              min_child_weight=8, missing=None, n_estimators=550, nthread=-1,
              objective='binary:logistic', reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, seed=42, silent=False, subsample=0.8)
```

The feature importance graph based on the tuned XGBClassifier is shown as below. The top t wo features are less important than before.



ANALYZING MODEL PERFORMANCE

We need to check the model's learning and testing error rates on various subsets of training data. See the graph below of the model's performance.



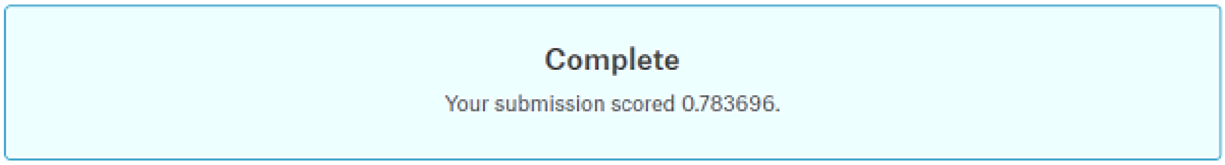
From the graph above, we can see that as the number of data increases, the difference between training set's AUC score and testing set's AUC score is getting smaller, which indicates that the models's bias and variance are getting low. However, it will still need further tuning to narrow the gap.

FINAL RESULT

We use 10-fold cross validation method to check the robustness of our final model. The scores are shown below.

```
[0.83816066  0.83869049  0.81618342  0.8401855   0.84787434  0.8407311
  0.85229894  0.85849388  0.85035749  0.82596608]
```

We don't see a huge difference among all the scores, which means the model is quite stable. The final result generated by the tuned classifier is submitted to Kaggle based on the test dataset provided by Kaggle. The submission gets a score for 0.783696, which is lower than the targeted top 25% score of 0.826120. The result we get requires further improvement to achieve the benchmark.



Complete

Your submission scored 0.783696.

SUMMARIZATION

In this project, we try to solve the problem of dissatisfied customer prediction raised by Santander Bank on Kaggle with the datasets provided by Santander Bank. As mentioned at the beginning of the report, we follow the steps below to come up with a solution for this problem.

Performance Metrics Selection -> Data Exploration and Cleaning -> Feature Selection and Transformation -> Model Training and Selection -> Model Tuning -> Performance Analysis

1. At the step of Performance Metrics Selection, we chose AUC score as the metrics for model evaluation.
2. In the Data Exploration and Cleaning part, we looked into each column and found out which columns have constant value or are duplicated with other columns. Then we remove those columns from the datasets.
3. For the Feature Selection and Transformation part, we checked the mean value and standard deviation for each column and scaled all the columns to range (0,1).
4. Then we implemented the Feature Selection and Transformation by using Random Forest Classifier to find the importance weight for each feature based on the entropy values from each decision tree. Then we transformed the whole datasets based on the selected features.

5. Next is Model Training and Selection. We built and trained different models, then compared the performance among all the models. We found out XGBoost classifier had the best AUC score among all the classifiers we chose.
6. Model Tuning is the most difficult part for this project we think. In order to get a better performance by tuning model's parameters, we need to really understand underlying theorem behind XGBoost. We used GridSearchCV technique to enumerate all selected possible parameters and tried to find out the combination that provided the highest AUC score. Note that GridSearchCV doesn't guarantee that we can find the optimal model. Most of the time we can only get to the local minimal point.
7. In the end, we implemented Performance Analysis to evaluate our final model by feeding it different sizes of datasets to observe if the difference of the final results between both training dataset and testing dataset was getting smaller as the size of dataset is increasing.

However, the final result is lower than the top 25% score which we set as the benchmark for this project.

REFLECTION

In this project, even though we try to use the tuned XGBoost model to predict the data, the result is still a bit disappointed when we compare the result with the top 25% teams. By collecting different sources of information and looking at the discussion board on Kaggle about this Santander competition, we find that there are several areas where we may need to look into to get a significant improvement on our model.

1. Feature Engineering, using different models to further analyze the features and finding out the outliers and replacing them.
2. Creating ensemble of models or stacking models. Only using one XGBoost model seems not enough, we have to use ensemble of models in order to get a leap.