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**

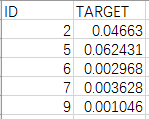
* train.csv - the training set including the target. The train.csv includes the target column which is the variable to predict. It equals 1 for unsatisfied customers and 0 for satisfied customers.
* test.csv – the test set without the target
* sample\_submission.csv – a sample submission file in the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



**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).

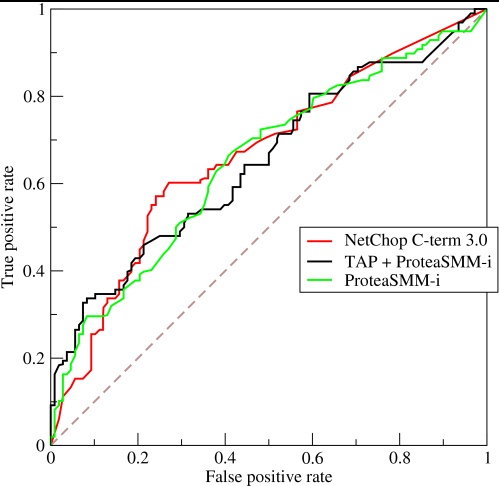
**TP rate = TP/total positives, FP rate = FP/total negatives**

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.

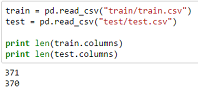
# Benchmark

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.

# Preliminary Data Exploration

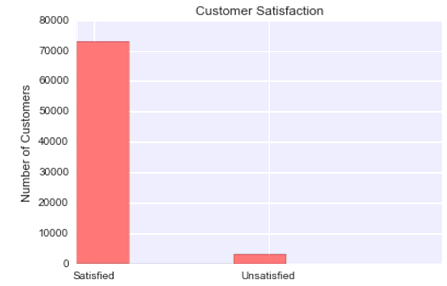
1. Looking at the raw data in the excel sheet, the meaning of each variable is unclear since the names are not given.

**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.



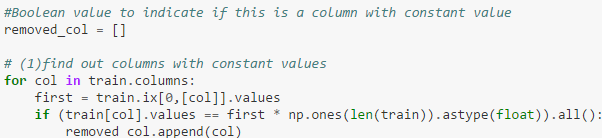
1. Below is the basic information of the data. Total number of training data set is 76020. Number of features is 369 (exclude “TARGET” and “ID”).

**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.



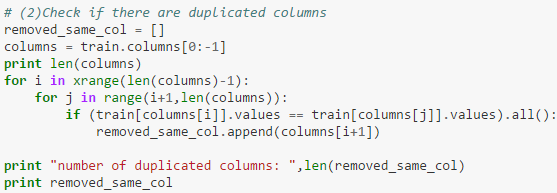
1. **Through the preliminary observation of the training dataset, we can find that some columns have constant values (i.e. value of 0).** We can find out all these columns and remove them since their information gain is 0 if we use them as features to classify the data (i.e. decision tree classifier). They have no input to the classification model.

Find out all the columns with constant values.



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

1. **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.



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

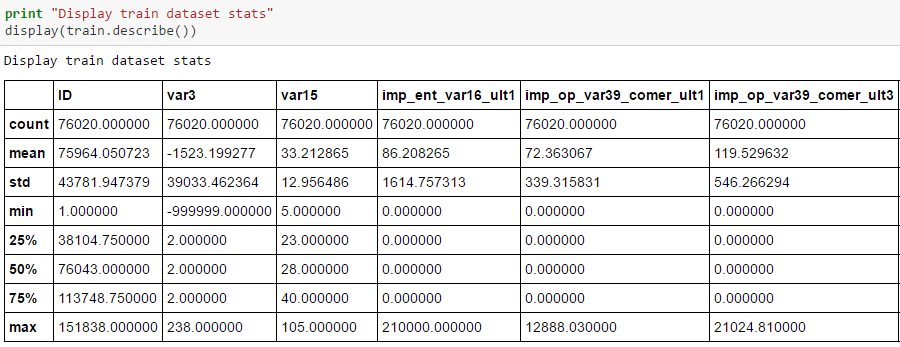
1. **Remove columns found in step 3 and 4 above.**





# Feature Scaling

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



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.

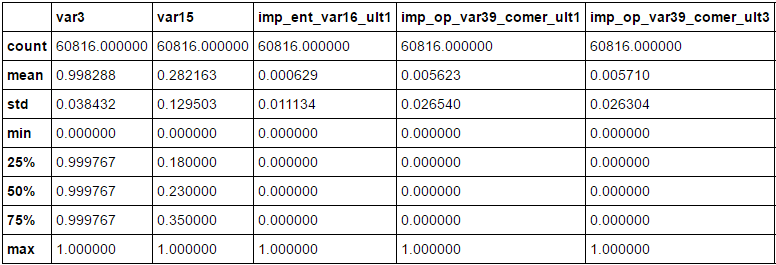
For example, if we compare “var3” with “var15”,

* “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**



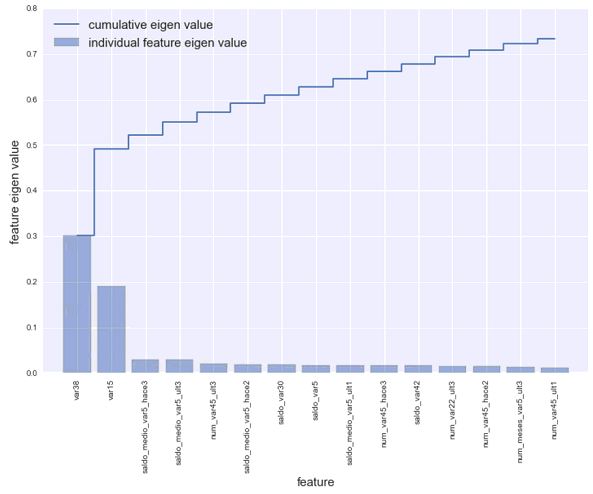
# Feature Selection And Transformation

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.

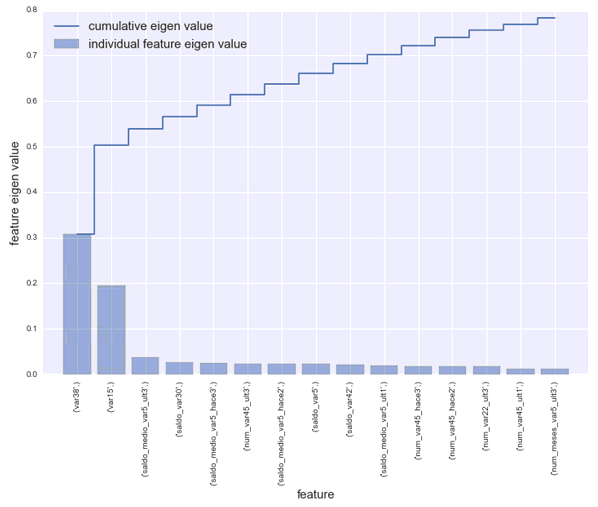
**In our case, the threshold is set to default value (None). According to the source code of SelectFromModel, if the threshold is “None”, the threshold used is 1e-5.** “prefit” parameter is set to “True” meaning the estimator is trained already.



The transformed dataset now has only 43 columns.



(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.

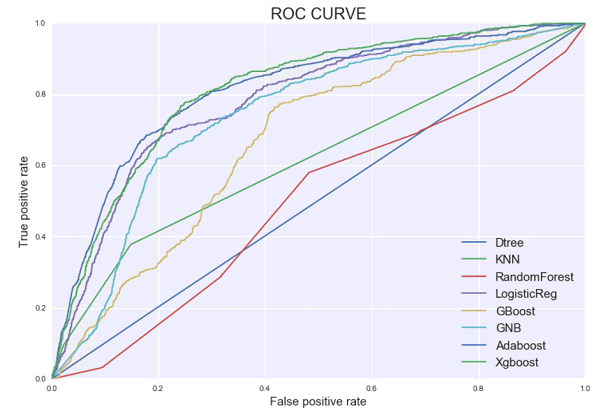
**Please find the running result below.**

|  |  |  |
| --- | --- | --- |
| 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>).

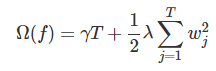
XGBoost is widely recognized in a number of machine learning challenges and many of the winning solutions use XGBoost. It is a gradient tree boosting with regularization (variant of original GBM). XGBoost, compared to other tree-based model, is a faster tool for learning better models.

**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

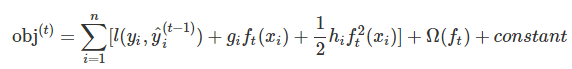


Where **w** is the vector of scores on leaves, **T** is the number of leaves

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

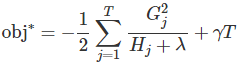


taking the Taylor expansion of the loss function up to the second order

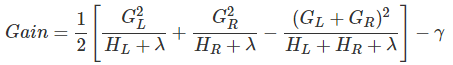


Where ***gi*** is the first derivative of the loss function and ***hi*** is the second derivative of the loss function. And we can find that the optimization for the new tree only depends on ***gi*** and ***hi****.*

1. **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.



1. **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



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.

1. **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

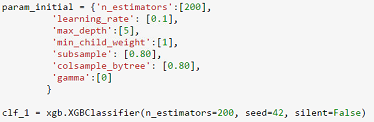
**gamma:** range [0, ∞], minimum loss reduction required to make a further partition on a leaf node of the tree. The model will be more conservative with larger gamma value

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.**
2. learning\_rate: using the default value 0.1
3. max\_depth: initial depth is 5. This usually should be between 3-10.
4. min\_child\_weight: initial weight is 1. Too high value can lead to under-fitting.
5. gamma: using default value 0. This will be tuned later.
6. subsample: starting at 0.8. This value usually ranges between 0.5-0.9
7. colsample\_bytree: starting at 0.8.
8. n\_estimators: starting at relatively low as 200



The initial AUC scores for both training dataset and testing dataset are shown as below:

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.

1. **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. The AUC scores are shown as below:

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

1. **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 the 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.

1. **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

1. **Next we will tune subsample and colsample\_bytree. We will tune the parameters by three 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

1. **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,

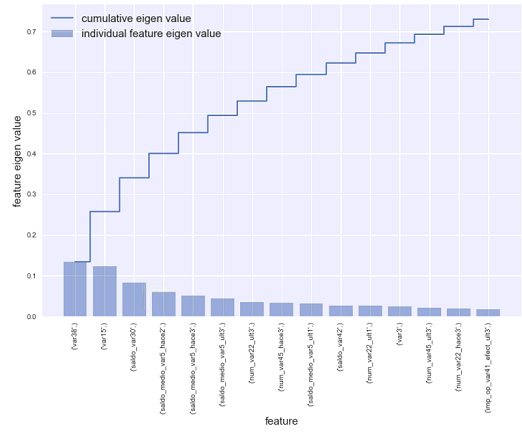
gamma=0, learning\_rate=0.02, max\_delta\_step=0, max\_depth=4,

min\_child\_weight=8, missing=None, n\_estimators=550, nthread=-1,

objective='binary:logistic', reg\_alpha=0, reg\_lambda=1,

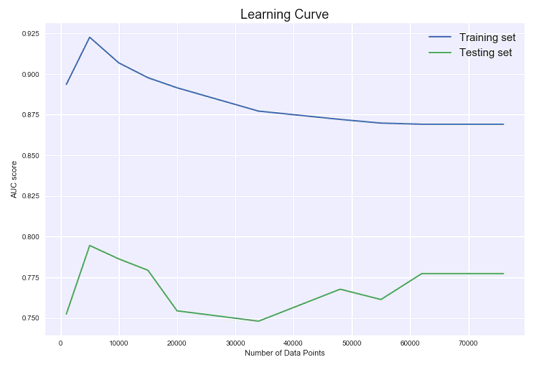
scale\_pos\_weight=1, seed=42, silent=False, subsample=0.8)

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



# Analyzing Model Performance

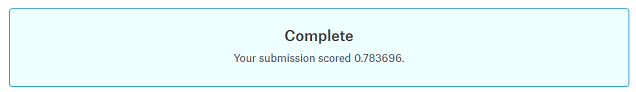
We need to check the model’s learning and testing error rates on various subsets of training data. See the graph below of the model’s performance.



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

The final result generated by the tuned classifier is submitted to Kaggle based on the test dataset provided by Kaggle. The submission can 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.



# Conclusion

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. AUC is selected as the performance metrics. We work on the data cleaning, feature Engineering, model selection and model tuning to build a model which can predict the dissatisfied customer.

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.