# Data Lab Fintech Project

Bowen Liu

LendingClub.com Loan Status Prediction

Aug 26th, 2016

# Definition

## Project Overview

Like most companies, lendingclub.com released a lot of information about the loan status over time. Many features could be used in the underwriting modeling for risk management. In this project, I created a classification algorithm that accurately identified which customers tend to go charged-off, based on their characteristics and background. With an improved prediction model in place, lendingclub.con investors will be able to minimize investment risks more than before.

## Problem Statement

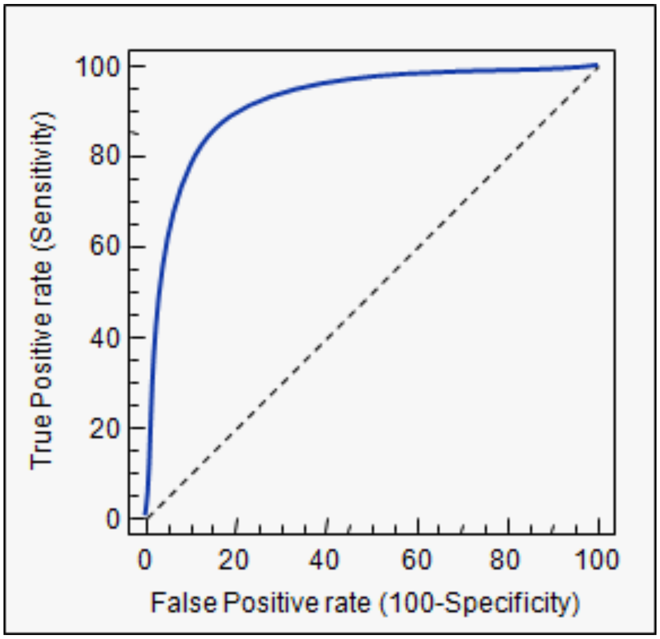
The problem was to predict the possible result of one loan initiated by a registered debtor. The loan outcome was defined by a charged-off/paid field attached to each unique loan. The outcome indicated whether or not each person paid the loan within a fixed window of time after each unique loan was released. The objectives were built upon the followings pipeline:

1. Downloaded the data set of 2014 and performed the exploratory data analysis
2. Preprocessed data and performed data engineering and benchmarked the classifier
3. Improved the performance using logistic regression model
4. Improved the performance using an ensemble method of XGBoost

## Metrics

In this project, receiver operating characteristic (ROC) is used to evaluate the model performance. According to Wikipedia definition, a receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 - specificity). The ROC curve is thus the sensitivity as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability in x-axis.

Fig. 1 A example plotting of ROC curve

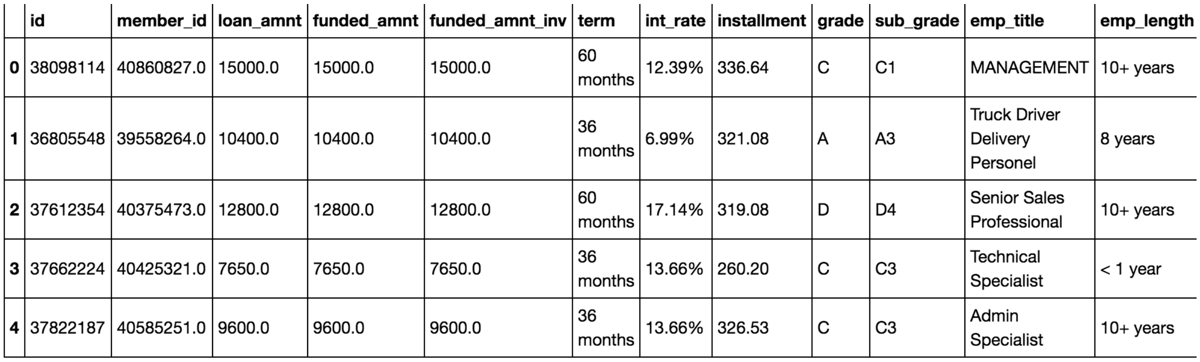


# Analysis

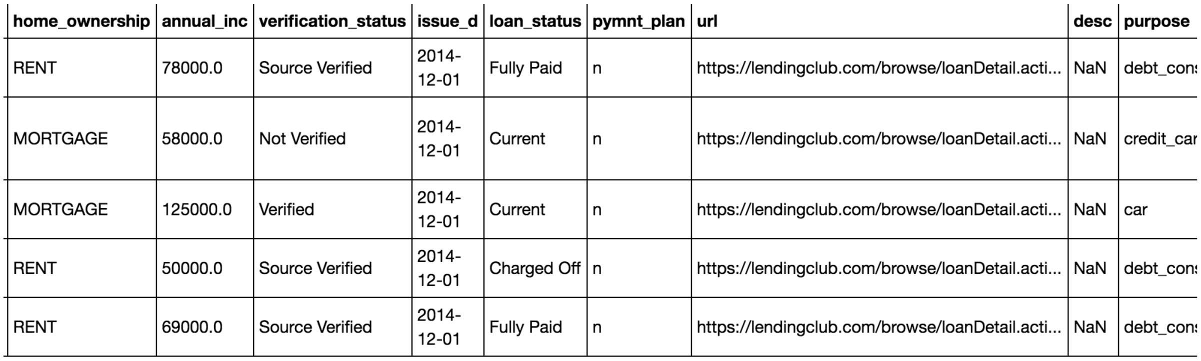
## Data Exploration

This data set includes only one data file: “LoanStats3c.csv”. The file contains 235,631 unique loans, 111 corresponding characteristics, and memory usage of 199.5+ MB. To develop a predictive model, the “loan status” field was chosen as the label. (see EDA part)

Table 1. A sample data set, including the first 5 rows of 21 columns.



| | id | member_id | loan_amnt | funded_amnt | funded_amnt_inv | term | int_rate | installment | grade | sub_grade | emp_title | emp_length |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 38098114 | 40860827.0 | 15000.0 | 15000.0 | 15000.0 | 60 months | 12.39% | 336.64 | C | C1 | MANAGEMENT | 10+ years |
| **1** | 36805548 | 39558264.0 | 10400.0 | 10400.0 | 10400.0 | 36 months | 6.99% | 321.08 | A | A3 | Truck Driver Delivery Personel | 8 years |
| **2** | 37612354 | 40375473.0 | 12800.0 | 12800.0 | 12800.0 | 60 months | 17.14% | 319.08 | D | D4 | Senior Sales Professional | 10+ years |
| **3** | 37662224 | 40425321.0 | 7650.0 | 7650.0 | 7650.0 | 36 months | 13.66% | 260.20 | C | C3 | Technical Specialist | < 1 year |
| **4** | 37822187 | 40585251.0 | 9600.0 | 9600.0 | 9600.0 | 36 months | 13.66% | 326.53 | C | C3 | Admin Specialist | 10+ years |



| home_ownership | annual_inc | verification_status | issue_d | loan_status | pymnt_plan | url | desc | purpose |
|---|---|---|---|---|---|---|---|---|
| RENT | 78000.0 | Source Verified | 2014-12-01 | Fully Paid | n | https://lendingclub.com/browse/loanDetail.acti... | NaN | debt_cons |
| MORTGAGE | 58000.0 | Not Verified | 2014-12-01 | Current | n | https://lendingclub.com/browse/loanDetail.acti... | NaN | credit_car |
| MORTGAGE | 125000.0 | Verified | 2014-12-01 | Current | n | https://lendingclub.com/browse/loanDetail.acti... | NaN | car |
| RENT | 50000.0 | Source Verified | 2014-12-01 | Charged Off | n | https://lendingclub.com/browse/loanDetail.acti... | NaN | debt_cons |
| RENT | 69000.0 | Source Verified | 2014-12-01 | Fully Paid | n | https://lendingclub.com/browse/loanDetail.acti... | NaN | debt_cons |

The data sets have four types of fields:

- Indexing fields: “id”, “member_id”
- Datetime fields: “issue_d”, “last_pymnt_d”, “next_pymnt_d”, “last_credit_pull_d”
- Numeric fields: “loan_amnt”, “funded_amnt”, “funded_amnt_inv”, “dti”, etc.
- Categorical fields: the most part of features including low and high cardinality
- Label: “loan status”, 7 levels including “Current”, “Fully Paid”, “Charged Off”, “Default”, “Late(31-120)”, “In Grace Period”, ”Late (16-31)”

## Exploratory Visualization

The data visualization will show how the label "loan status" distributes and how each predictor correlates with the label, which is helpful to understand each field and build the expectation of prediction. 10% of total data points is sampled to make the exploratory plotting. (see EDA part)

Fig. 21 A plot showing how the label outcome distributes. The biggest part of loans is in progress. Among the past loans, fully paid and charged off are the top two status compared with others.

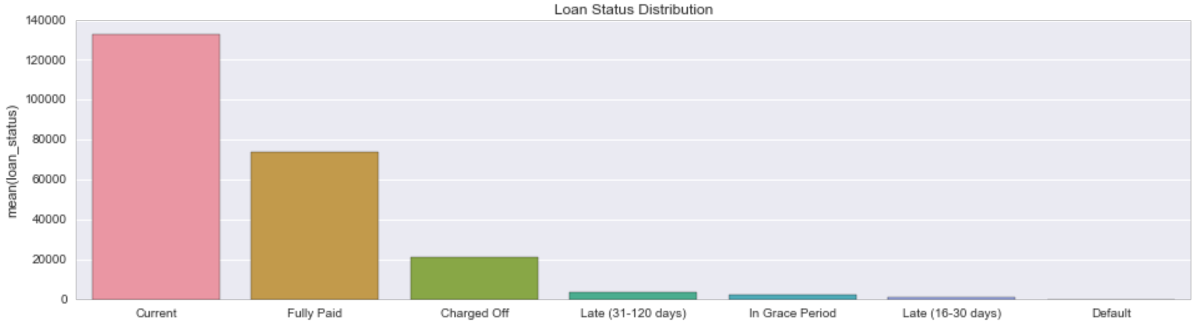


Fig.2 A plotting showing the loan grades distribution released by lendingclub.com between charged off and others. It looks like charged off rates increased along the alphabetical grades.

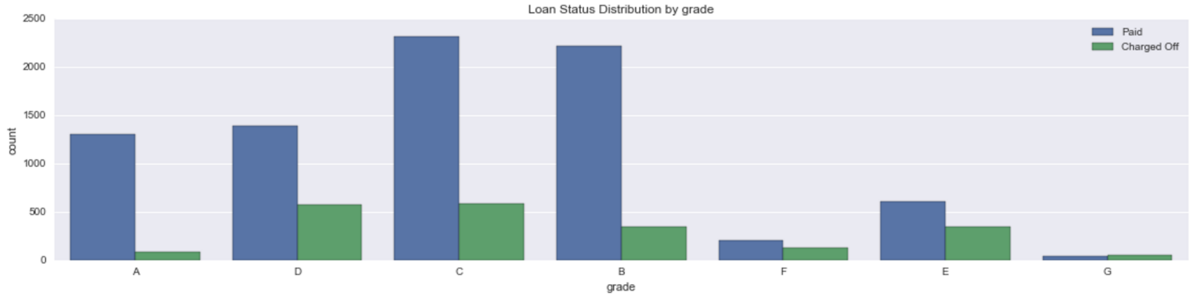


Fig. 3 A plotting showing the loan "sub_grade" distribution between paid and charged off loans. It shows the similar increased trend of charged off as grade from A to G.

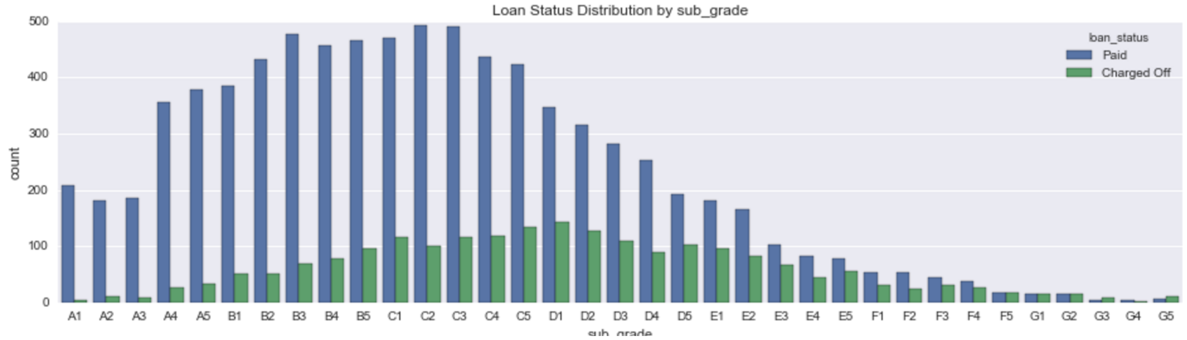


Fig. 4 A plotting showing how the delinquency times of recent 2 years distribute between paid and charged off loans. The charged off proportion increases as delinquency goes up.

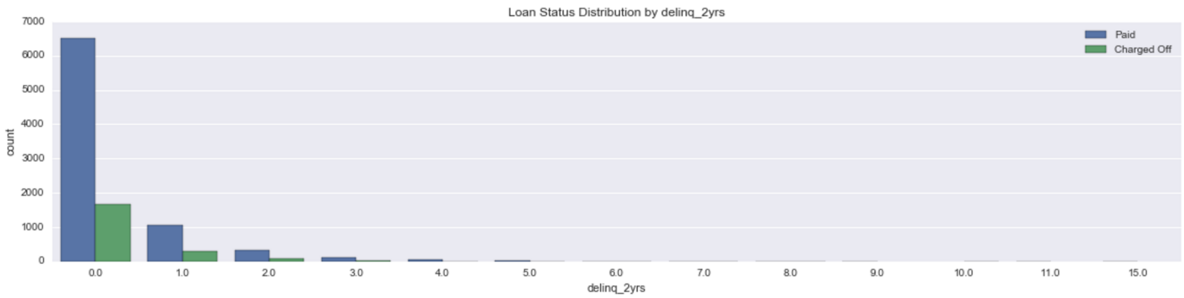


Fig. 5 A plotting showing how employment length distributes along loan status. There are no apparent trends of the impact of employment length on loan status.

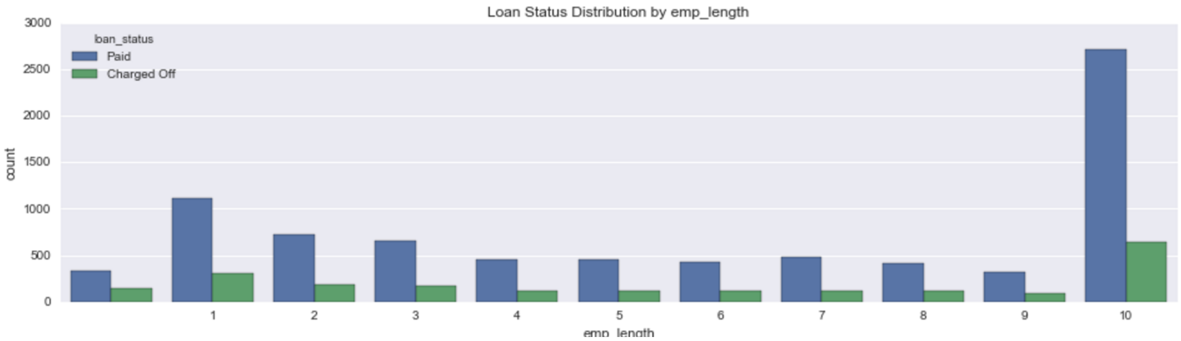


Fig. 6 A plotting showing loan status distribution along the address state. The proportion of charged off loans seems to be fair in each state expect the volume of loans.

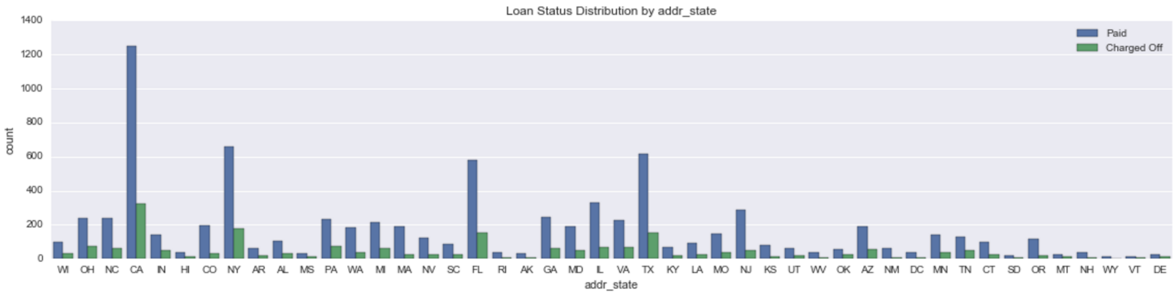


Fig. 7 A plotting showing how the number of credit line distribute between paid and charged off loans. Smaller number of credit line tend to have more loans from 4.0 to 21.0.

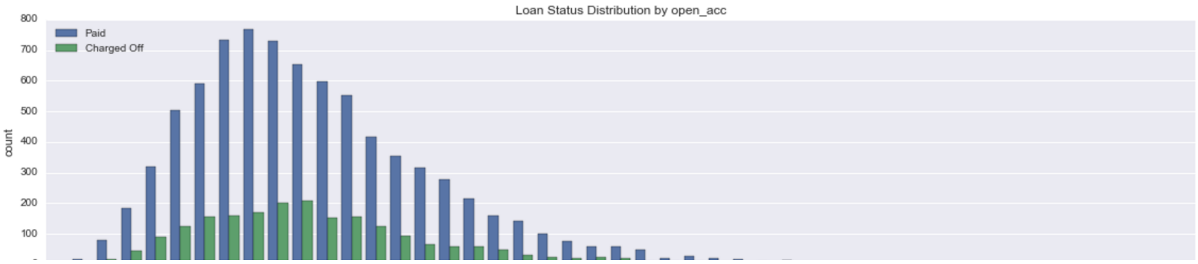


Fig. 8 A plotting showing how interest rates correlate with the loan outcomes. Charged off debtors have a slightly higher interest rate.

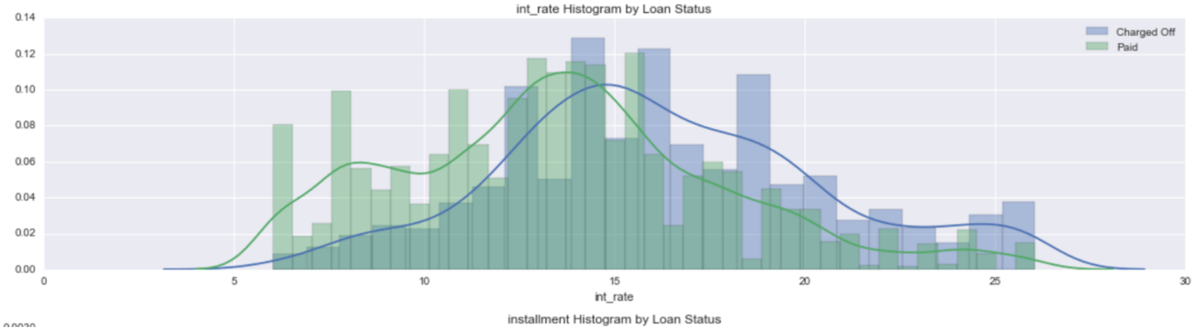
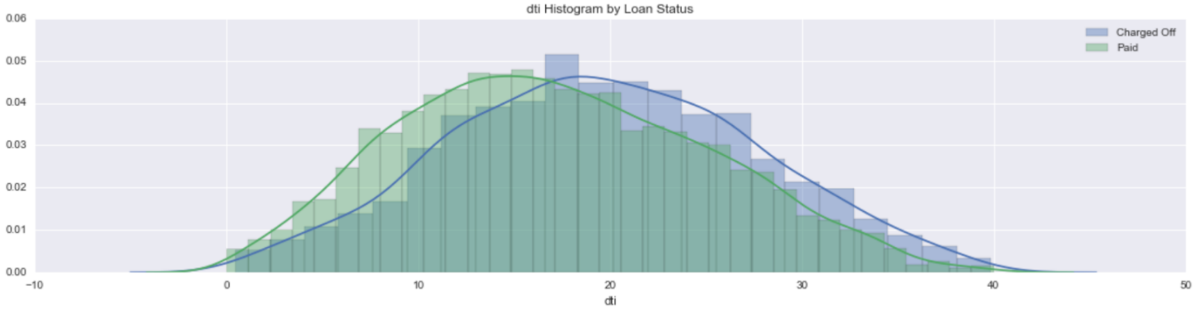


Fig. 9 A plotting showing the association between debt-to-income and loan status. Paid loans debtors tend to have lower dti compared with charged off debtors.



## Algorithms and Techniques

In this project, I started with a Logistic Regression and then improved it in a XGBoost algorithm. For linear regression, it assumes that there is one smooth linear decision boundary. Then it finds the weights by a maximum likelihood approach. It is a light-weigh algorithm and easy to implement. The overfitting could be easily controlled under L1 or L2 regularization. Moreover, as shown in the EDA, some features are sparse in values, which is not feasible for tree models. The important tuning parameters for logistic regression are as follows:

- “Penalty”: str, ‘l1’ or ‘l2’, used to specify the norm used in the penalization.
- “C : float, optional (default=1.0), Inverse of regularization strength L2; must be a positive float. Smaller values specify stronger regularization.

Furthermore, I improved the modeling performance using XGBoost after Logistic Regression, which has a higher prediction power. As any other boosting methods, XGBoost is trying to avoid overfitting of tree model by sampling without replacement of data points. Unlike random forest that bagging a bunch of trees, boosting uses a sequence of tree models to focus on correcting classifications where previous trees make mistakes. Moreover, XGBoost is built on Gradient Boosting by add some extra functions like regularization, paralleling processing, tree tuning (goes till the max depth even if there in no positive gain and prunes back), and built-in cross validation. However, XGBoosting is very difficult to tune since it has a lot of parameters. Hence, I combined manual and Bayesian Optimization to tune XGBoost. Here are some important parameters tuned in this project:

**General parameters:**

- "n_estimators": the number of trees
- "nthread": This is used for parallel processing and number of cores in the system should be entered

**Booster parameters:**

- "eta": learning rate, typically used in 0.01 – 0.2
- "**min_child_weight**": minimum sum of weights of all observations required in a child
- "max_depth, default 6": the maximum depth of a tree
- "max_leaf_nodes": the maximum number of leaf nodes
- "gamma": a node is split only when resulting split gives a positive gain above the gamma.
- "subsample": the fractions of observations for each tree
- "colsample_bytree": the fractions of features for each tree
- "lambda": L2 regularization on weights, analogous to ridge regression
- "alpha": L1 regularization on weight

**Learning Task Parameters:**

- "eval_metric": rmse, mae, logloss, error, merror, mlogloss, auc
- "seed": random number seed

In the training stage, the cleaned data (after preprocessing and data engineering) were fed into the algorithm. Then a cross validation was applied in grid search to find the best solution. The evaluation metrics was AUC scores. For XGBoosting, the tuning approach was a little complex, manual tuning around the default values were first performed to define the boundary and then a Bayesian Optimization (github repo by fernando) was implemented for a detailed tuning.

## Benchmark

The Logistic Regression was used to benchmark the prediction model. It gave us ROC score of 0.54 for training data. As mentioned in the previous section, we could penalize the parameters by tuning the "L1" or "L2" to control over-fitting. I used ElasticNet algorithm which is built on Logistic Regression with the combination of "L1" and "L2" as the regularization parameters. Besides, the ElasticNet was fed to grid search (Sklearn package) to search optimal parameters. Here I chose alpha as the tuning parameters. The final score of ElasticNet was 0.76, improved by 0.22. (see modeling benchmark)

Fig. 10 A plotting showing ROC curve and score of Logistic Regression

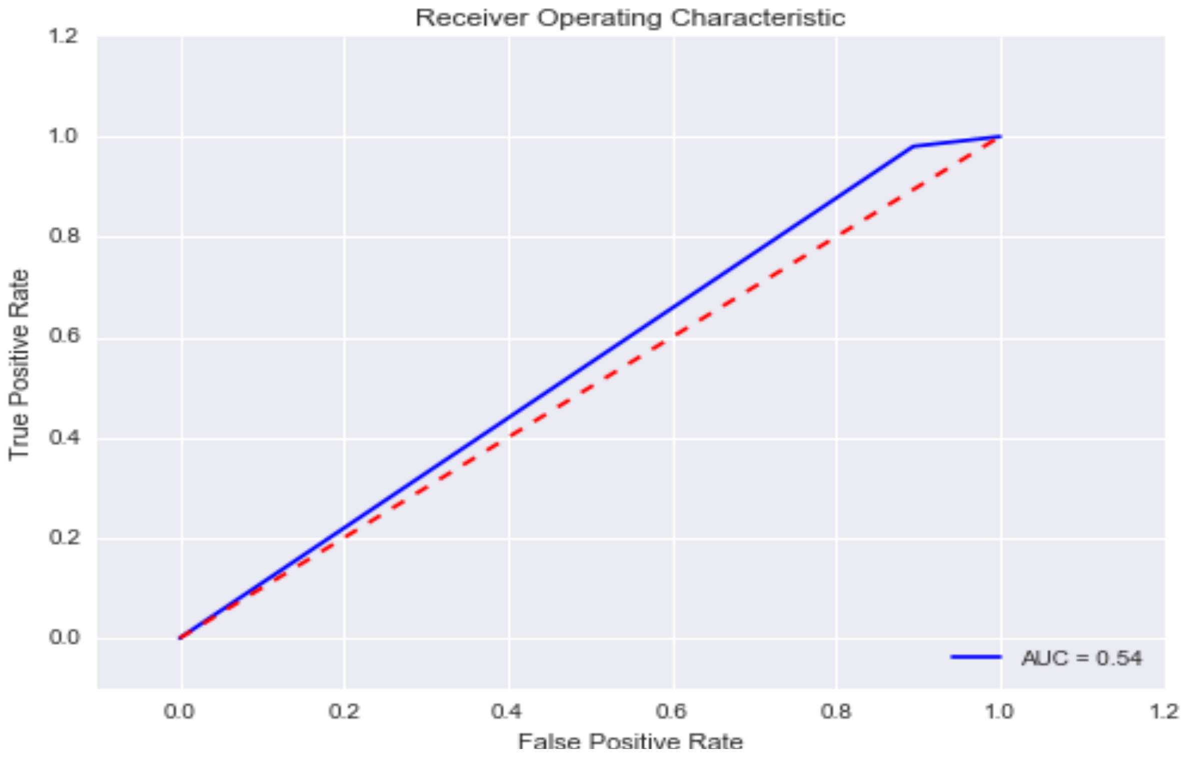
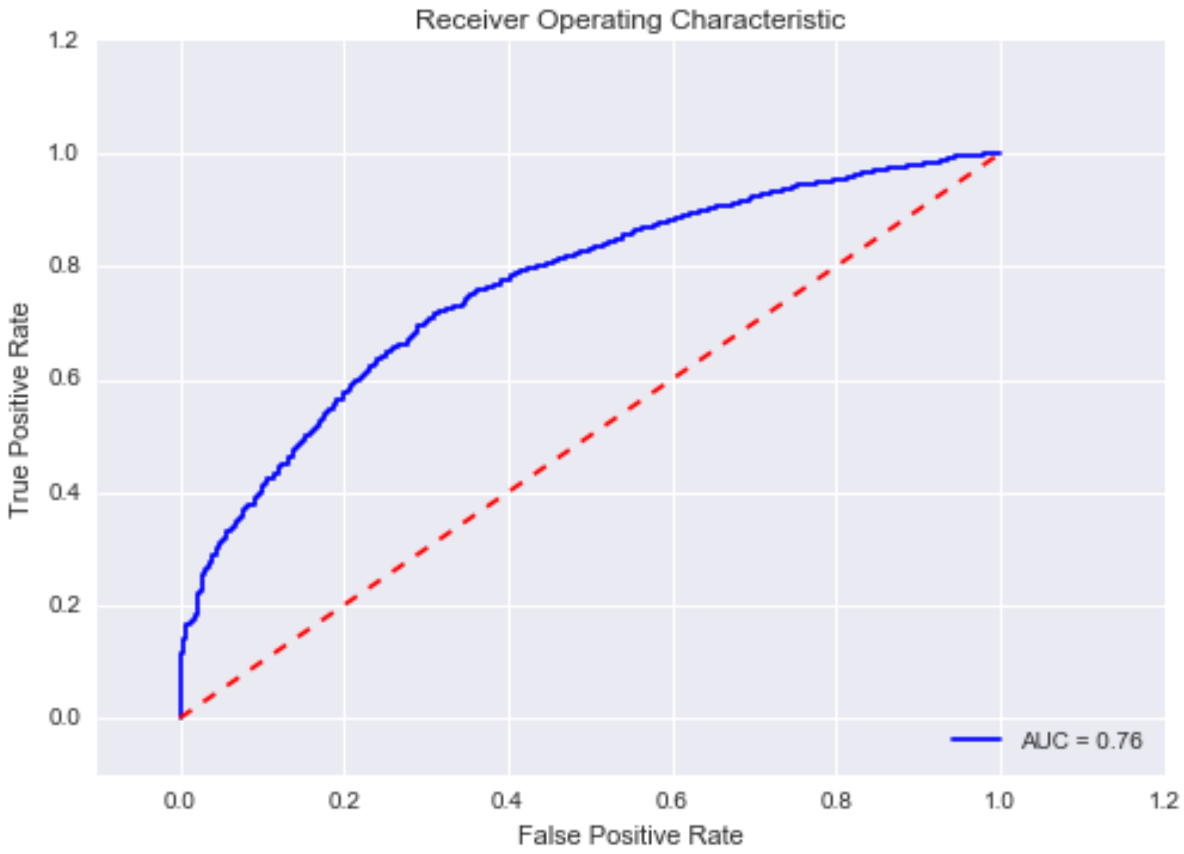


Fig. 11 A plotting showing ROC curve and score after grid search of Elastic Net



# Methodology

## Data Preprocessing

The data preprocessing consists of the following steps:

- Removed post-decision features like 'funded_amnt', 'funded_amnt_inv', etc.
- Stripped categorical value into numeric, like “60 months” to “60”.
- Imputed the “NaN” with “0” as one specific type.
- One-hot encoding low cardinality features like 'emp_title', 'title', 'zip_code', 'desc'
- Tried frequency encoding, leave-one-out encoding, and feature encoding on high

- Normalized numeric features by mapping original value onto a normal distribution.
- Transformed the label into binary “Charged off” and “Paid”

Besides, the sample data for exploratory visualization was dropped to control overfitting.

(see data engineering part)

## Implementation

The modeling process was implemented in Jupyter Notebook. It can be further dived into the following steps: (see XGBoosting and Bayesian Optimization part)

1. Loaded raw data and preprocess them as the previous section.
2. Before modeling, the data set was split along the time series to generalize the prediction. The data of previous 10 months was used as the training data, and the last two months as the test data.
3. In Logistic Regression, the grid search function was used to tune the parameters
4. In XGBBoost, a helper function “xgb_fit” was used to output accuracy, AUC scores of Train and Test data and feature importance.
5. Manual tuning of XGBoost was performed to decide the initial boundaries of parameters.
6. Based on manual tuning, the Bayesian Optimization was applied to reach the final optimal parameters.

## Results

To improve the prediction performance, we moved onto the XGBoosting algorithm. First manual tuning of XGBoosting was performed. Then Bayesian Optimization was applied for comparison. (see XGBoosting and Bayesian Optimization part)

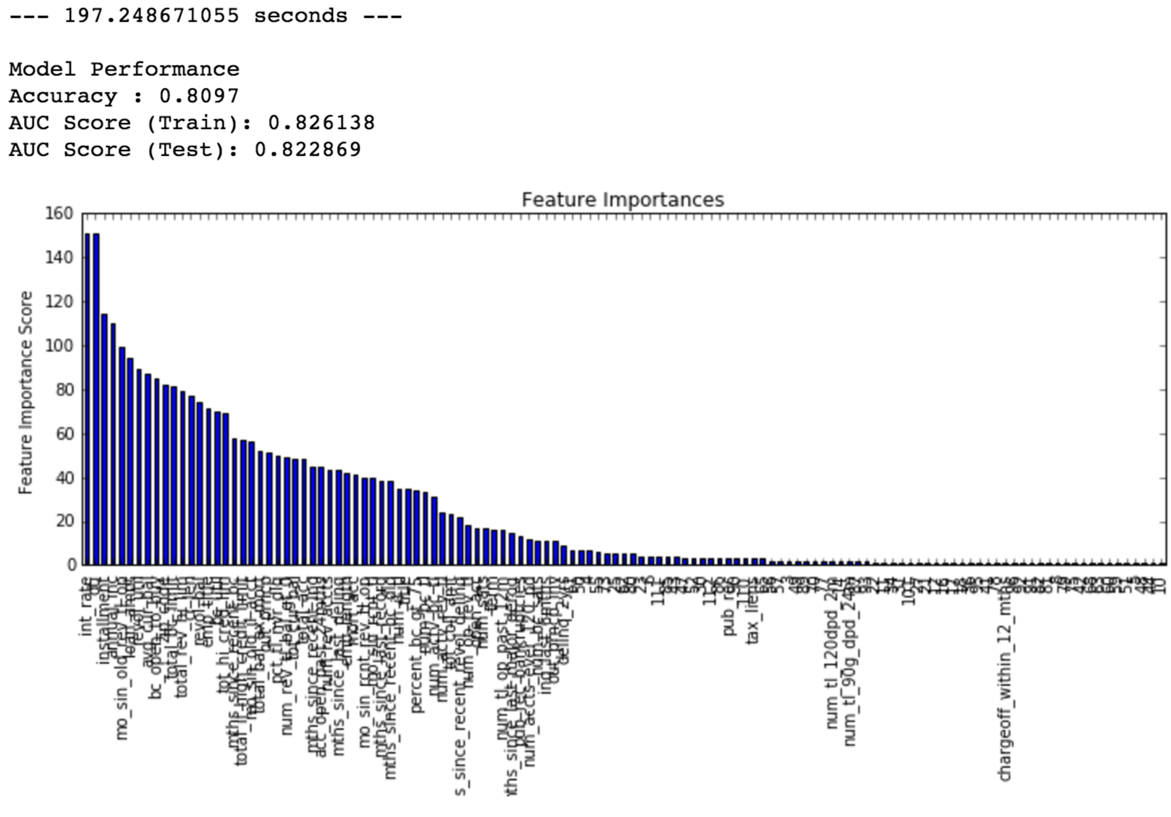
**Manual Tuning:**

1. The model started with the number of estimators of 145, outputting the test score of 0.822869.

2. Given the number of estimeors of 145, grid search of “max_depth” and “min_child_weight” gave us a cross validation score of 0.787256 and parameters of 5 and 4 respectively.
3. With previous parameters fixed, grid search of “min_child_weight” provided with cross validation score of 0.787389 and “min_child_weight” of 8.
4. Grid search of “gamma” kept the optimal value at 0.0 with score of 0.788591
5. Grid search of “subsample” and “colsample_bytree” output the cross validation score of 0.77808 with the parameters of 0.6 and 0.9.
6. Grid search of “reg_alpha” output the cross validation score of 0.786870 with “reg_alpha” of 0.1.
7. After feeding all data to the final model, we got a test score of 0.821945.

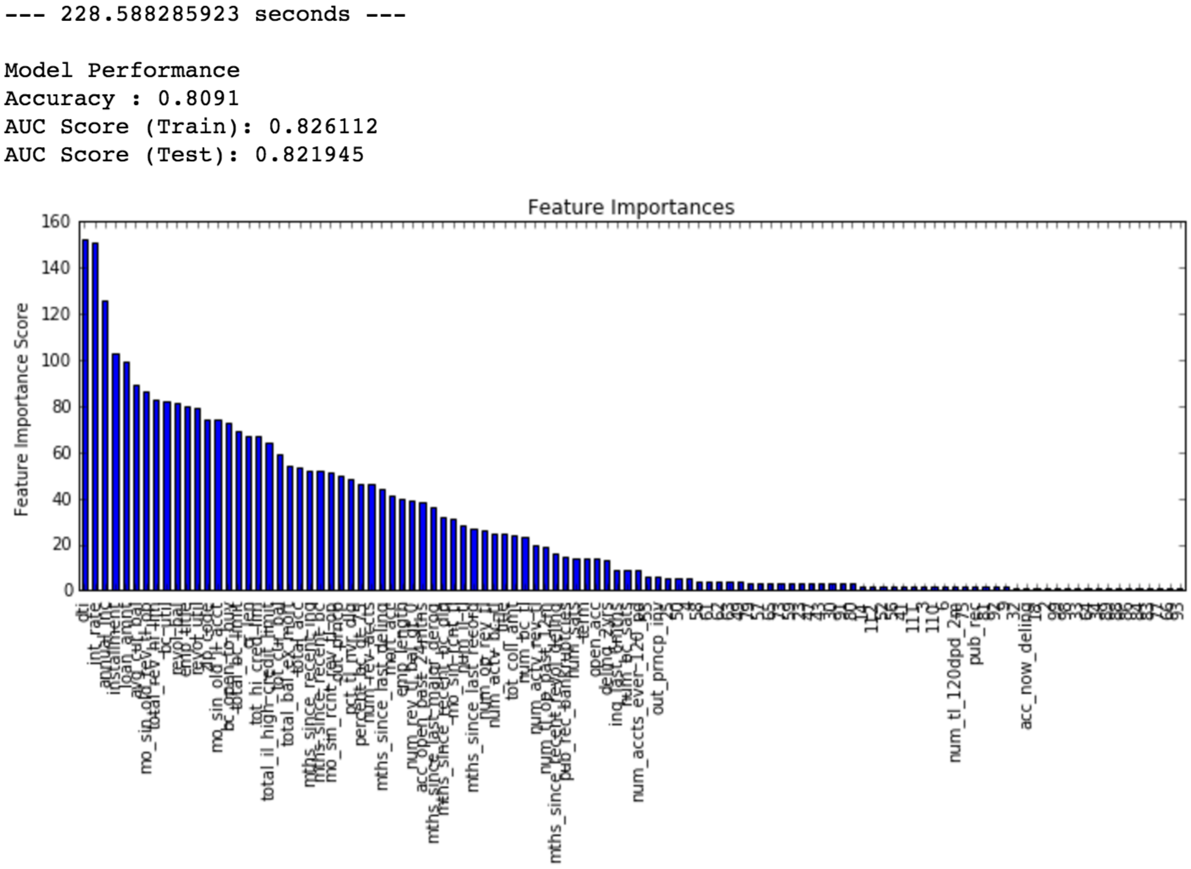
Compared with Logistic Regression, XGBoosting improved the performance from 0.76 to 0.822869. But manual tuning didn’t improve further with the final score of 0.821945.

Fig. 12 A plotting showing the score and feature importance of XGBoosting under the default settings.



```
--- 197.248671055 seconds ---

Model Performance
Accuracy : 0.8097
AUC Score (Train): 0.826138
AUC Score (Test): 0.822869
```

Fig. 13 A plotting showing the score and feature importance of XGBoosting after manual tuning.



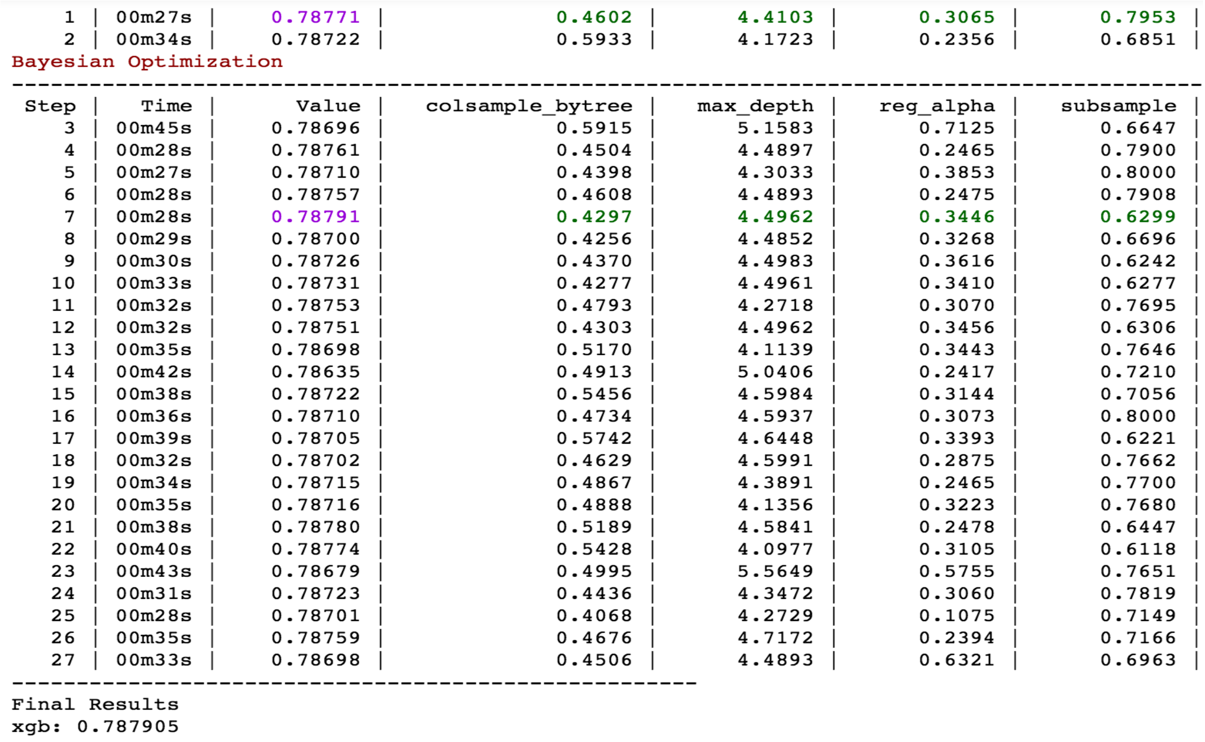
```
--- 228.588285923 seconds ---

Model Performance
Accuracy : 0.8091
AUC Score (Train): 0.826112
AUC Score (Test): 0.821945
```

## Bayesian Optimization

Based on manual tuning, we chose “max_depth” range from 4 to 6, “subsample” from 0.6 to 0.8, “colsample_bytree” from 0.4 to 0.6, and “reg_alpha” from 0.01 to 1 for a Bayesian Optimization to search the optimal parameters. The optimal cross validation score was 0.787905 with “colsample_bytree” of 0.4297, “max_length” of 4.4962, “reg_alpha” of 0.3446, and “subsample” of 0.6299. The final test score was 0.822673, slightly lower than the default settings and higher than the manual tuning.

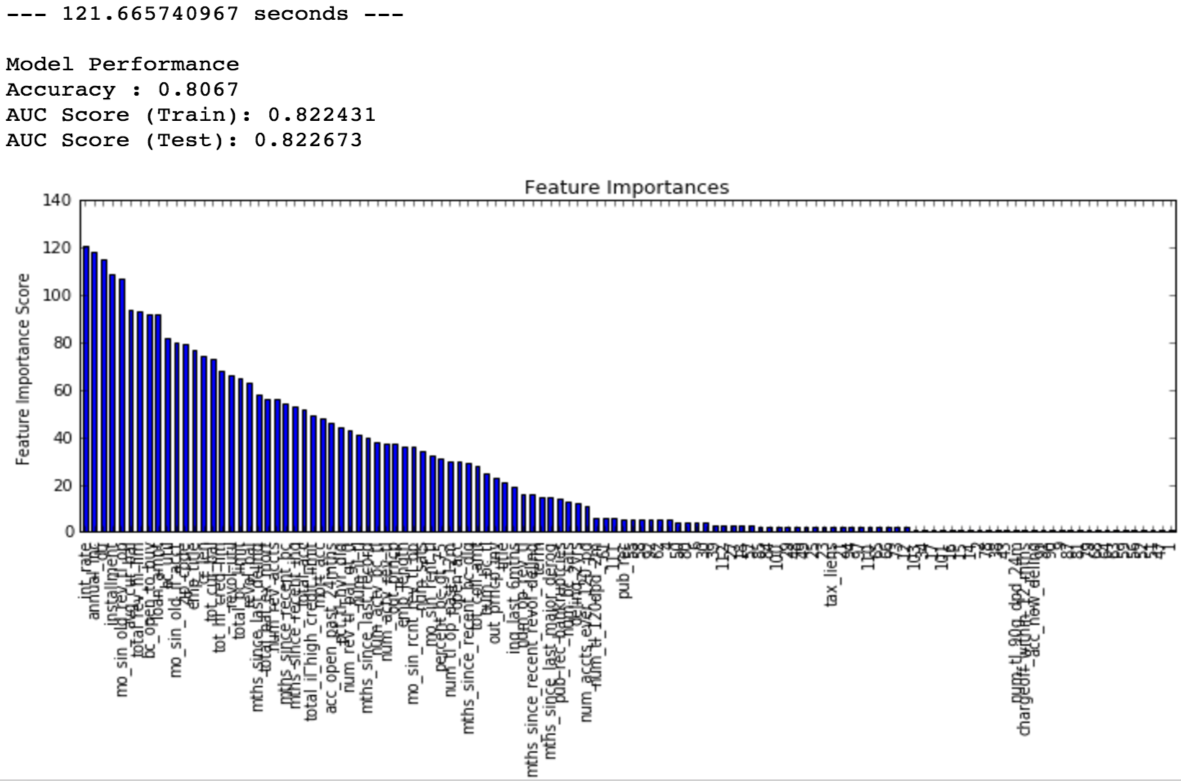
Still, Bayesian Optimization was very efficient compared with manual tuning. For this combination of parameters, the time complexity is 3 * 200 * 200 * 900. But Bayesian Optimization used only 27 trials to reach the similar performance as manual tuning. It saved a huge amount of time, considering the possible combinations.

Fig. 14 A plotting showing the process of Bayesian Optimization



```
     1 | 00m27s |    0.78771 |             0.4602 |     4.4103 |     0.3065 |     0.7953 |
     2 | 00m34s |    0.78722 |             0.5933 |     4.1723 |     0.2356 |     0.6851 |
Bayesian Optimization
-----------------------------------------------------------------------------------------------
  Step |   Time |      Value |   colsample_bytree |  max_depth |  reg_alpha |  subsample |
     3 | 00m45s |    0.78696 |             0.5915 |     5.1583 |     0.7125 |     0.6647 |
     4 | 00m28s |    0.78761 |             0.4504 |     4.4897 |     0.2465 |     0.7900 |
     5 | 00m27s |    0.78710 |             0.4398 |     4.3033 |     0.3853 |     0.8000 |
     6 | 00m28s |    0.78757 |             0.4608 |     4.4893 |     0.2475 |     0.7908 |
     7 | 00m28s |    0.78791 |             0.4297 |     4.4962 |     0.3446 |     0.6299 |
     8 | 00m29s |    0.78700 |             0.4256 |     4.4852 |     0.3268 |     0.6696 |
     9 | 00m30s |    0.78726 |             0.4370 |     4.4983 |     0.3616 |     0.6242 |
    10 | 00m33s |    0.78731 |             0.4277 |     4.4961 |     0.3410 |     0.6277 |
    11 | 00m32s |    0.78753 |             0.4793 |     4.2718 |     0.3070 |     0.7695 |
    12 | 00m32s |    0.78751 |             0.4303 |     4.4962 |     0.3456 |     0.6306 |
    13 | 00m35s |    0.78698 |             0.5170 |     4.1139 |     0.3443 |     0.7646 |
    14 | 00m42s |    0.78635 |             0.4913 |     5.0406 |     0.2417 |     0.7210 |
    15 | 00m38s |    0.78722 |             0.5456 |     4.5984 |     0.3144 |     0.7056 |
    16 | 00m36s |    0.78710 |             0.4734 |     4.5937 |     0.3073 |     0.8000 |
    17 | 00m39s |    0.78705 |             0.5742 |     4.6448 |     0.3393 |     0.6221 |
    18 | 00m32s |    0.78702 |             0.4629 |     4.5991 |     0.2875 |     0.7662 |
    19 | 00m34s |    0.78715 |             0.4867 |     4.3891 |     0.2465 |     0.7700 |
    20 | 00m35s |    0.78716 |             0.4888 |     4.1356 |     0.3223 |     0.7680 |
    21 | 00m38s |    0.78780 |             0.5189 |     4.5841 |     0.2478 |     0.6447 |
    22 | 00m40s |    0.78774 |             0.5428 |     4.0977 |     0.3105 |     0.6118 |
    23 | 00m43s |    0.78679 |             0.4995 |     5.5649 |     0.5755 |     0.7651 |
    24 | 00m31s |    0.78723 |             0.4436 |     4.3472 |     0.3060 |     0.7819 |
    25 | 00m28s |    0.78701 |             0.4068 |     4.2729 |     0.1075 |     0.7149 |
    26 | 00m35s |    0.78759 |             0.4676 |     4.7172 |     0.2394 |     0.7166 |
    27 | 00m33s |    0.78698 |             0.4506 |     4.4893 |     0.6321 |     0.6963 |
---------------------------------------------------------------------
Final Results
xgb: 0.787905
```

Fig. 15 A plotting showing the score and feature importance of XGBoosting after Bayesian Optimization



```
--- 121.665740967 seconds ---

Model Performance
Accuracy : 0.8067
AUC Score (Train): 0.822431
AUC Score (Test): 0.822673
```

# Conclusion

The modeling process for this project can be summarized as the following steps:

1. Described the problem and the objectives
2. Downloaded and preprocessed the data sets
3. Performed data engineering on the data sets
4. Built the baseline Logistic Regression classifier
5. Manually tuned XGBoosting classifier
6. Applied Bayesian Optimization to XGBoosting classifier

Understanding the data is the key to build and improve any model. I found step 2, 3 difficult and valuable, especially in identifying post decision features and high cardinality feature encoding like frequency encoding, leave-one-out encoding, and ECDF normalization method. Because data split based on time series was made after EDA. For modeling part, it became straight forward after the previous work. XGBoosting tuning was tedious in some sense, but with Bayesian Optimization, the tuning process became efficient.