# STAT 5232 – Generalized Linear Model Final Project

Authors: Yiding Xie (yx2443), Zhibo Zhou (zz2520); Gye Hyun Baek (gb2508)
Columbia University, M.A. Statistics
E-mail: yx2443@columbia.edu; zz2520@columbia.edu; gb2508@columbia.edu

*Abstract*—**In this project, we built machine learning algorithms for the prediction of loan defaulters based on Lending Club data. Our main goal is to correctly identifying defaulter's accuracy metric so that lending club can decide whether a person is fit for sanctioning a loan or not in the future. We dealt with several issues, such as imbalanced data, dirty data, combined with multiple categorical variable columns. Using cross validation, we fit the four models on the test datasets. The calculation metrics include accuracy score, recall score, area under curve (AUC) and model fitting time. XGBoost algorithm gave the best results in terms of training/test accuracy and training/test recall scores.**

## I. Purpose of Analysis

Since all three of us are into predictive analytics, we want to utilize what we have learned from class, such as modeling logistic regressions and dealing with categorical variables. Loan financing and lending payments have become such a big component in our daily life. In the industry, more and more companies are utilizing machine learning algorithms to predict loan defaults. Therefore, we decided to analyze the data from Lending Club and build predictive modeling using classification methods.

## II. Introduction

We will go step by step for building machine learning algorithms for the prediction of loan defaulters based on certain variables from Lending Club. Our main goal is to correctly identifying defaulter's accuracy metric (True positives + True negative) so that lending club can decide whether a person is fit for sanctioning a loan or not in the future.

## III. Company Overview

Lending Club (https://www.lendingclub.com/) is a peer to peer lending company based in the United States, in which investors provide funds for potential borrowers and investors earn a profit depending on the risk they take (the borrowers credit score). Lending Club provides the "bridge" between investors and borrowers.

## IV. Data Overview

The dataset contains complete loan data for all loans issued through the 2017-2018, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information. The file containing loan data through the "present" contains complete loan data for all loans issued through the previous completed calendar quarter. Additional features include credit scores, number of finance inquiries, address including zip codes, and state, and collections among others. The file is a matrix of about 500 thousand observations and 140+ variables.

The processed data is stored on Google Drive (https://drive.google.com/drive/u/1/folders/1MSkrU1IaPpamwulqDBbJ1RR_dYuUYuNc ). And access is available to all users with Columbia Lionmail.

| | Unnamed: 0 | id | member_id | loan_amnt | funded_amnt | funded_amnt_inv | term | int_rate | installment | grade | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | NaN | 30000.0 | 30000.0 | 30000.0 | 36 months | 7.34% | 930.99 | A | ... |
| 1 | 2 | NaN | NaN | 34825.0 | 34825.0 | 34825.0 | 60 months | 12.61% | 785.45 | C | ... |
| 2 | 3 | NaN | NaN | 2600.0 | 2600.0 | 2600.0 | 36 months | 7.96% | 81.43 | A | ... |
| 3 | 4 | NaN | NaN | 20000.0 | 20000.0 | 20000.0 | 60 months | 9.92% | 424.16 | B | ... |
| 4 | 5 | NaN | NaN | 17000.0 | 17000.0 | 17000.0 | 60 months | 20.39% | 454.10 | D | ... |

*Figure 1: Data Overview*

## V. Software Description

Both R and Python were utilized in our project. Detailed codes are provided in the Appendix. We used R to combine the datasets downloaded from Lending Club, and used Python for the rest of the analysis. Also, we utilized Google Cloud Platform Virtual Machine to perform our analysis.

## VI. Exploratory Data Analysis

### A. *Preprocessing*

After taking an initial look at the dataset, the percentage of missing data in many columns are far more than we can work with. So, we decide to remove columns having more than 49% of missing values.

| | Missing Values | % of Total Values |
|---|---|---|
| desc | 495250 | 100.0 |
| url | 495250 | 100.0 |
| member_id | 495250 | 100.0 |
| id | 495242 | 100.0 |
| orig_projected_additional_accrued_interest | 495017 | 100.0 |
| payment_plan_start_date | 494980 | 99.9 |
| hardship_reason | 494980 | 99.9 |
| hardship_status | 494980 | 99.9 |
| deferral_term | 494980 | 99.9 |
| hardship_amount | 494980 | 99.9 |
| hardship_start_date | 494980 | 99.9 |
| hardship_end_date | 494980 | 99.9 |
| hardship_dpd | 494980 | 99.9 |
| hardship_length | 494980 | 99.9 |
| hardship_loan_status | 494980 | 99.9 |

*Figure 2: Missing Values %*

### B. *Define Loan Status*

Based on Janio Bachmann's report named "Lending Club Loan Analysis", we decide to define the bad loans including "Charged Off", "Default", "Does not meet the credit policy. Status: Charged Off", "In Grace Period", "Late (16-30 days)", "Late (31-120 days)". And the rest of the loan categories belong to good loans.

```
Current                437318
Fully Paid              40240
Charged Off              6942
Late (31-120 days)       6509
In Grace Period          2901
Late (16-30 days)        1323
Default                     9
Name: loan_status, dtype: int64
```

*Figure 3: Loan Type Counts*

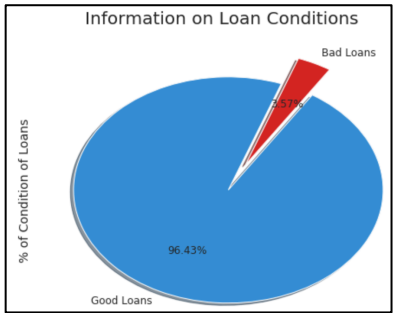


*Figure 4: Loan Condition Percentage*

### C. *EDA*

We first examined the relationship between loan condition and loan amount, payment plan (Figure 5). Clearly, none of the good loans tends to have payment plan while some of the bad loans do.

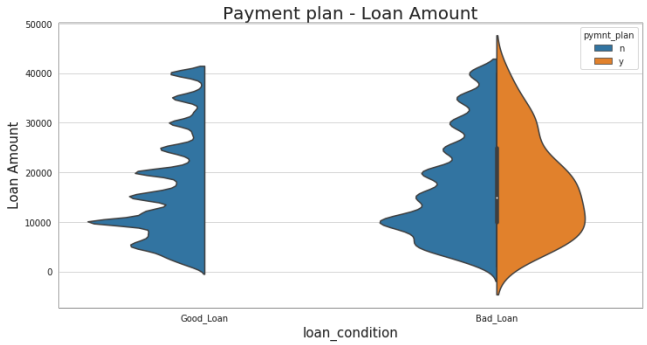


*Figure 5: Payment plan vs. Loan Condition*

Then we looked at the relationship between loan condition and interest rate, loan grade (Figure 6). It appears that for the same grade level, the interest rate level for bad loans is higher than its corresponding rate for good loans. This makes intuitive sense because people who tend to default will be charged for a higher spread (higher interest rate).

Figure 7 also confirmed this hypothesis. Compared with the distribution of good loans, bad loans tend to be more positively skewed.

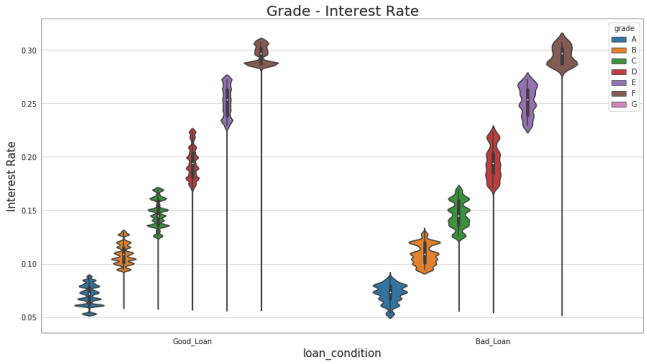


*Figure 6: Loan Condition vs Grade*

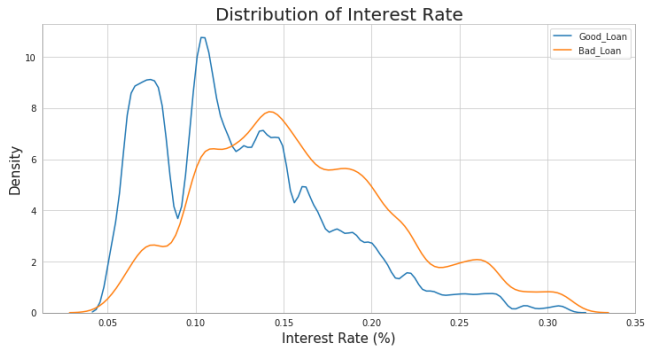


*Figure 7: Loan Condition vs. Interest Rate*

Then we try to examine if there's any geographical effect by looking at the state column. According to Figure 8 and 9, clearly bigger states like California, New York and Texas tend to have more loan amounts, for both good and bad loans. We do want to include state parameter in our final

model because we found states like Illinois seems to have higher good-bad loans, compared to some other states like New York.

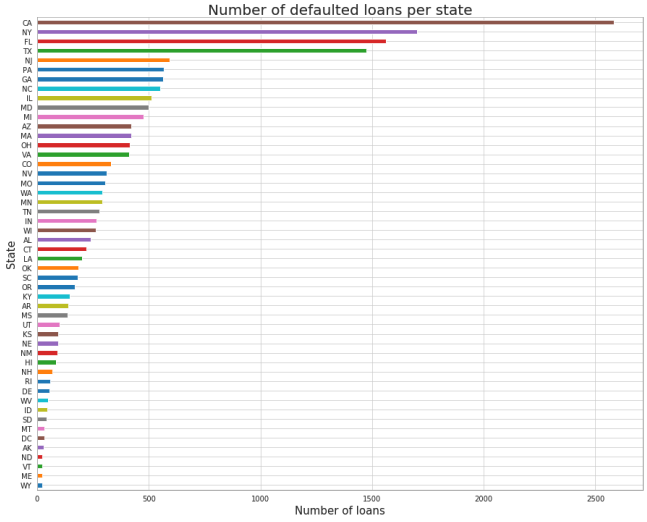


*Figure 8: Number of Bad Loans by States*

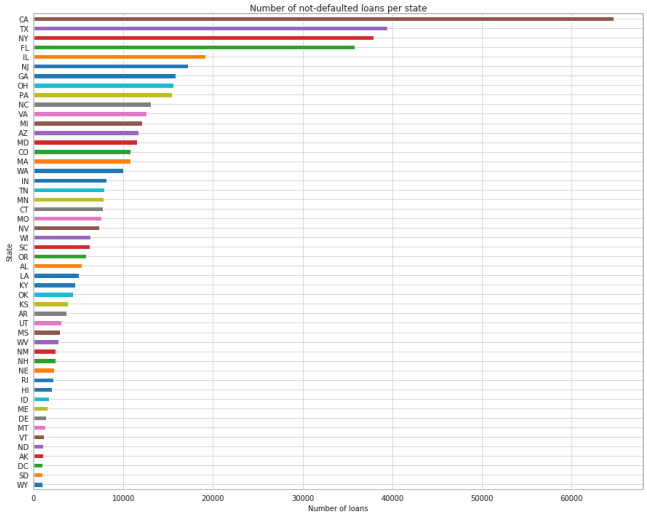


*Figure 9: Number of Good Loans by State*

## VII. Feature Engineering

After adopting the research theory from Janio and combining the results from boxplot analysis, we finally determined the following 23 predictors to be included in our prediction model: loan amount, funded amount, funded amount committed by investors, term, interest rate, installment, grade, sub-grade, homeownership, annual income, verification status, issue date, payment plan, purpose, address state, debt payment to obligation ratio, delinquency for the past two years, inquiry in the past 6 month, times of opening account, Number of derogatory public records, revolving balance, utilization rate, number of total accounts.

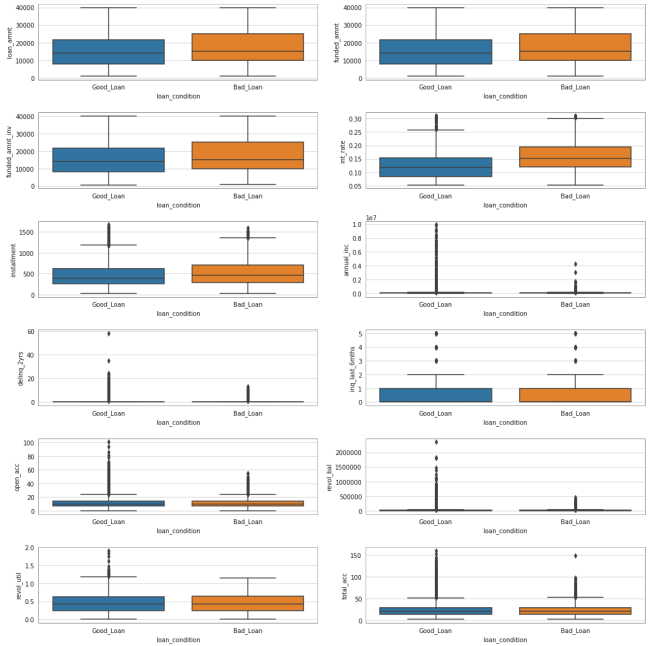


*Figure 10: Boxplots on the selected predictors*

## VIII. Dealing with Categorical Variables

Among the selected variables, five of them are categorical variables. In Python, we can use techniques such as label encoding and one hot encoding to convert them into dummy variables for further analysis. Then the datasets look like the following (Figure 11).

| term | int_rate | installment | annual_inc | issue_d | pymnt_plan | dti | ... | addr_state_SD | addr_state_TN | addr_state_TX |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0734 | 930.99 | 95000.0 | 2018 | 0 | 16.18 | ... | 0 | 0 | 1 |
| 1 | 0.1261 | 785.45 | 125000.0 | 2018 | 0 | 21.31 | ... | 0 | 0 | 0 |
| 0 | 0.0796 | 81.43 | 62000.0 | 2018 | 0 | 19.61 | ... | 0 | 0 | 0 |
| 1 | 0.0992 | 424.16 | 110000.0 | 2018 | 0 | 10.56 | ... | 0 | 0 | 0 |
| 1 | 0.2039 | 454.10 | 52000.0 | 2018 | 0 | 15.65 | ... | 0 | 0 | 0 |

*Figure 11: Data Overview after Converting to Dummy Variables*

## IX. Other Issues to Consider Before Modeling

### A. *Normalization*

Normalization is a technique often applied as part of data preparation for machine learning. Urvashi Jaitley in her blog "Why Data Normalization is necessary for Machine Learning models" made a good summary point on data normalization. She believed the goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

It makes sense for us to normalize our datasets, because some of the features have different ranges.

### B. *Train-Validation-Test Split*

Tarang Shah had a great discussion on this topic in his blog "About Train, Validation and Test Sets in Machine

Learning". Training set is the sample of data used to fit the model, validation set is the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters, while the test set is the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

Here, we decided to hold about 50% data for training set, 20% for validation set, and the rest 30% for test set.

Later in the sections, we will also talk about the use of cross validation to tune our prediction models.

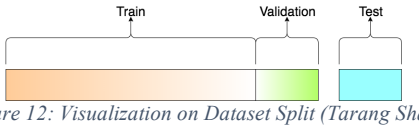


*Figure 12: Visualization on Dataset Split (Tarang Shah)*

### C. *Imbalanced Data*

The other issue we are facing when we deal with this project is that we are having a really imbalanced dataset. It's known that some of the machine learning algorithms are very sensitive to imbalanced data.
When we are dealing with imbalanced datasets, the most common methods are oversampling and downsampling. Some of the supervised learning problems will suffer from imbalanced data, therefore, it's very crucial to apply some of these methods before we build up the models. The most famous ones are SMOTE and NearMiss.

According to "SMOTE AND NEAR MISS IN PYTHON: MACHINE LEARNING IN IMBALANCED DATASETS", written by Saeed Abdul Rahim, SMOTE is an oversampling method. What it does is, it creates synthetic of the minority class. Hence making the minority class equal to the majority class. SMOTE does this by selecting similar records and altering that record one column at a time by a random amount within the difference to the neighboring records. On the other hand, NearMiss is an under-sampling technique. Instead of resampling the Minority class, using a distance, this will make the majority class equal to minority class.

After utilizing SMOTE method, the total number of datasets is oversampled to 233224 counts and the processing time is about 25.71 seconds. Meanwhile, NearMiss methods down-sample the number of data counts to only 8600 and the processing takes 407.18 seconds. Based on the results (Figure 13 and 14), we decided to use SMOTE method for the rest of analysis because the results are more appealing. Here we apply logistic regression with the same parameters, and the only difference is the input dataset. Figure 13 shows the result for SMOTE method, and Figure 14 is for NearMiss. Both the accuracy and recall scores for SMOTE methods are much higher. This is not surprising because we lost many useful data during downsampling process.

```
print_score(log_reg_r, 0, 0, X_val, y_val, train=False)
 Test Result:

 accuracy score: 0.647260

 recall score: 0.644764

 Classification Report:
              precision    recall  f1-score   support

           0       0.07      0.71      0.13      3757
           1       0.98      0.64      0.78     99883

 avg / total       0.95      0.65      0.76    103640


 Confusion Matrix:
  [[ 2681  1076]
  [35482 64401]]
```

*Figure 13: SMOTE Result*

```
print_score(log_reg_d, 0, 0, X_val, y_val, train=False)
 Test Result:

 accuracy score: 0.347192

 recall score: 0.330627

 Classification Report:
              precision    recall  f1-score   support

           0       0.04      0.79      0.08      3757
           1       0.98      0.33      0.49     99883

 avg / total       0.94      0.35      0.48    103640


 Confusion Matrix:
  [[ 2959   798]
  [66859 33024]]
```

*Figure 14: NearMiss Result*

## X. MODELING

### A. *Logistic Regression*

Logistic regression models the probabilities for classification problems with two possible outcomes. For classification, we prefer probabilities between 0 and 1, so we wrap the right side of the equation into the logistic function. This forces the output to assume only values between 0 and 1.

$$P(y^{(i)}=1)=\frac{1}{1+exp(-(\beta_0+\beta_1 x_1^{(i)}+\ldots+\beta_p x_p^{(i)}))}$$

*Figure 15: Logistic Regression Formula*

Performing cross validation, the algorithm suggests using L1 penalty and 0.5 as the penalty tuning parameter.

Figure 16 below shows the cross-validation result using logistic regression. The accuracy score is about 0.711 and the recall score is about 0.687. And the model fitting time is about 1298.6 seconds.

```
print_score(log_reg_best, x_train0_r, y_train0_r, 0, 0, train=True)

 Train Result:

 accuracy score: 0.711348

 recall score: 0.687161

 Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.74      0.72    333107
           1       0.72      0.69      0.70    333107

 avg / total       0.71      0.71      0.71    666214


 Confusion Matrix:
  [[245012  88095]
  [104209 228898]]

 Average Accuracy:        0.711216
 Accuracy SD:             0.001293

print("--- %s seconds ---" % (log_reg_time6 - log_reg_time5))

 --- 1298.6267139911652 seconds ---
```

*Figure 16: Logistic Regression CV Results*

### B. *Random Forest*

Random Forest is a supervised learning algorithm. Like you can already see from its name, it creates a forest and makes it somehow random. The forest it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Performing cross validation, the algorithm suggests using 250 as the number of estimators, 11 as the maximum depth, and 2 as the minimum number of splits.

Figure 17 below shows the cross-validation result using logistic regression. The accuracy score is about 0.917 and the recall score is about 0.938. And the model fitting time is about 157.9 seconds.

```
print_score(clf_rf_best, x_train0_r, y_train0_r, 0, 0, train=True)

 Train Result:

 accuracy score: 0.917265

 recall score: 0.938188

 Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.90      0.92    333107
           1       0.90      0.94      0.92    333107

 avg / total       0.92      0.92      0.92    666214


 Confusion Matrix:
  [[298578  34529]
  [ 20590 312517]]

 Average Accuracy:        0.913166
 Accuracy SD:             0.025433

print("--- %s seconds ---" % (clf_rf_time4 - clf_rf_time3))

 --- 157.8775873184204 seconds ---
```

*Figure 17: Random Forest CV Results*

In Python, we can also look into the feature importance of this random forest model. Figure 18 tells us the most important predictor is number of inquiries in the last 6 months, verification status and home ownership.

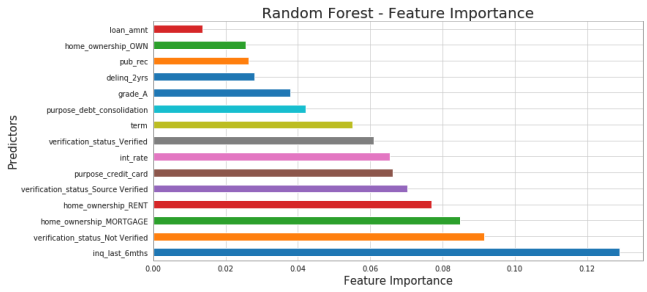


*Figure 18: Random Forest Feature Importance*

### C. *XGBoost*

Before getting into XGBoost, it makes more sense to start with boosting algorithms. In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library, which proves to be faster and better than traditional boosting methods.

Performing cross validation, the algorithm suggests using 200 as the number of estimators, 7 as the maximum depth, and 0.3 as the learning rate.

Figure 19 below shows the cross-validation result using logistic regression. The accuracy score is about 0.982 and the recall score is about 0.999. And the model fitting time is about 218.4 seconds.

```
print_score(clf_xg_best, x_train0_r, y_train0_r, 0, 0, train=True)

 Train Result:

 accuracy score: 0.982441

 recall score: 0.999955

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    333107
           1       0.97      1.00      0.98    333107

 avg / total       0.98      0.98      0.98    666214


 Confusion Matrix:
  [[321424  11683]
  [    15 333092]]

 Average Accuracy:        0.980807
 Accuracy SD:             0.036781

print("--- %s seconds ---" % (clf_xg_time4 - clf_xg_time3))

 --- 218.42041444778442 seconds ---
```

*Figure 19: XGBoost CV Results*

Similarly, we can try to understand the feature importance of this XGBoost model (Figure 20). The top three predictors are interest rate, debt payment to obligation ratio, and utilization rate. Comparing this feature chart with the

previous chart on random forest, we notice many of the top predictors are dramatically different.

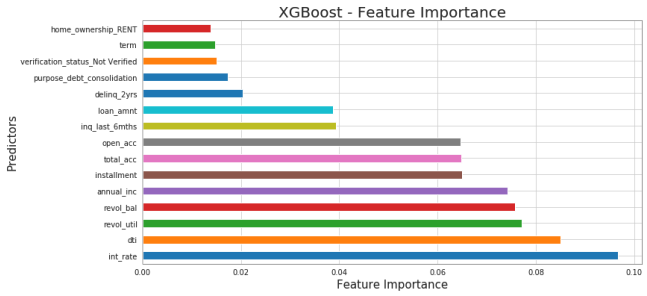


*Figure 20: XGBoost Feature Importance*

### D. *Averaging Ensemble*

Ensemble averaging is the process of creating multiple models and combining them to produce a desired output. Here we combined all logistic regression, random forest, and XGBoost into the ensemble model.

Figure 21 below shows the cross-validation result using logistic regression. The accuracy score is about 0.944 and the recall score is about 0.950. And the model fitting time is about 1407.9 seconds.

```
print_score(ensemble_best, x_train0_r, y_train0_r, 0, 0, train=True)

 Train Result:

 accuracy score: 0.943511

 recall score: 0.950235

 Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.94    333107
           1       0.94      0.95      0.94    333107

 avg / total       0.94      0.94      0.94    666214


 Confusion Matrix:
  [[312050  21057]
  [ 16577 316530]]

 Average Accuracy:        0.939275
 Accuracy SD:             0.027609

print("--- %s seconds ---" % (ensemble_time2 - ensemble_time1))

 --- 1407.9327342510223 seconds ---
```

*Figure 21: Ensemble CV Results*

## XI. Model Evaluation

Finally, we fit the four models on the test datasets and we obtained Figure 22. The calculation metrics include accuracy score, recall score, area under curve (AUC) and model fitting time.

Not surprising, XGBoost algorithm gave the best results in terms of training/test accuracy and training/test recall scores, while its AUC is just slightly worse than logistic regression and its model fitting time takes just slightly longer than Random Forest.

| | ML Name | Train Accuracy | Test Accuracy | Train Recall | Test Recall | Test AUC | Model Fit Time (sec) |
|---|---|---|---|---|---|---|---|
| 2 | XGBoost | 0.982441 | 0.964385 | 0.999955 | 0.999559 | 0.738 | 218.420 |
| 3 | Averaging Ensemble | 0.943511 | 0.951019 | 0.950235 | 0.982066 | 0.754 | 1407.933 |
| 1 | Random Forest | 0.917265 | 0.909683 | 0.938188 | 0.936639 | 0.687 | 157.878 |
| 0 | Logistic Regression | 0.711348 | 0.688881 | 0.687161 | 0.688368 | 0.758 | 1298.627 |

*Figure 22: Comparison between Models*

## XII. Conclusion

Determining the loan outcome, like many financial predictions, is clearly not an easy task. Throughout the project, we have confronted several issues, such as imbalanced data and imperfect data. We also selected logistic regression as the baseline model, and extended to Random Forest, XGBoost, and Averaging Ensemble.

In summary, we have successfully built machine learning algorithms to predict the people who might default on their loans. This can be further used by Lending Club for their analysis. Also, we might want to look on other techniques or variables to improve the prediction power of the algorithm. One of the drawbacks from the dataset is the limited number of people who defaulted on their loan. The other drawback is that we first wanted to utilize the datasets starting from year 2007 to end of 2018. However, due to limited computational power, we have to select a subset of the data available, which is only available from beginning of 2017 to end of 2018.

## XIII. Contribution

This project is completed by the collaborative effort by Yiding Xie, Zhibo Zhou, and Gye Hyun Baek. Specifically, here is a detailed breakdown:

*Yiding Xie: EDA, Data Cleaning, Model Tuning and Prediction, Report, Poster*
*Zhibo Zhou: EDA, Data Cleaning, Report, Citation, Additional Analysis, Poster*
*Gye Hyun Baek: Topic Pick, Model Selection and Comparison, Report, Poster*

## XIV. Reference

[1] "3.2.4.3.1. Sklearn.ensemble.RandomForestClassifier." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[2] "A Hitchhiker's Guide to Lending Club Loan Data." *Kaggle*, www.kaggle.com/pragyanbo/a-hitchhiker-s-guide-to-lending-club-loan-data#Modeling.

[3] "About Train, Validation and Test Sets in Machine Learning." Towards Data Science, Towards Data Science, 6 Dec. 2017, towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[4] Beuckelmann, Markus. "Customizing Nbconvert's PDF Export." Weblog, 23 Sept. 2016, www.markus-beuckelmann.de/blog/customizing-nbconvert-pdf.html.

[5] Gunasekara, Pavithra. "Installing LaTeX on Ubuntu - DZone Open Source." Dzone.com, 8 Jan. 2019, dzone.com/articles/installing-latex-ubuntu.

[6] Jaitley, Urvashi, and Urvashi Jaitley. "Why Data Normalization Is Necessary for Machine Learning Models." Medium, Medium, 7 Oct. 2018, medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

[7] "Lending Club || Risk Analysis and Metrics." *Kaggle*, www.kaggle.com/janiobachmann/lending-club-risk-analysis-and-metrics.

[8] "Sklearn.model_selection.GridSearchCV." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[9] "Sklearn.linear_model.LogisticRegression." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[10] "Sklearn.ensemble.VotingClassifier¶." Scikit, scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html.

[11] "Personal Loans Borrow up to $40,000 and Get a Low, Fixed Rate." *LendingClub*, www.lendingclub.com/info/download-data.action.

[12] "Python API Reference." *Python API Reference - Xgboost 0.83.dev0 Documentation*, xgboost.readthedocs.io/en/latest/python/python_api.html.

[13] Rahim, Saeed Abdul, and Saeed Abdul Rahim. "SMOTE AND NEAR MISS IN PYTHON: MACHINE LEARNING IN IMBALANCED DATASETS." Medium, Medium, 4 June 2018, medium.com/@saeedAR/smote-and-near-miss-in-python-machine-learning-in-imbalanced-datasets-b7976d9a7a79.

[14] "Running Jupyter Notebook on Google Cloud Platform in 15 Min." Towards Data Science, Towards Data Science, 8 Sept. 2017, towardsdatascience.com/running-jupyter-notebook-in-google-cloud-platform-in-15-min-61e16da34d52.

[15] "Save and Load Machine Learning Models in Python with Scikit-Learn." Machine Learning Mastery, 10 Mar. 2018, machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/.