# Loan Approval and Quality Prediction in the Lending Club Marketplace

Milestone Write-up

Yondon Fu, Shuo Zheng and Matt Marcus

## Recap

Lending Club is a peer-to-peer lending marketplace where individual investors can provide arms-length loans to individual or small institutional borrowers. Lending Club performs the loan evaluation and underwriting, and investors such as you or I would fund the loans (in a way similar to KickStarter).

As a creditor, Lending Club performs loan underwriting in a much different method from traditional consumer loan creditors such as a consumer bank. Lending Club receives applications from individuals looking to borrow money, and evaluates the loan decision exclusively based on the information provided by the applicant; in-person evaluations are not involved. The company then assigns a rating of the loan, similar to how a rating agency such as Moody's assigns a rating to a publicly traded security, which significantly determines the interest rate on the loan. Lending Club then makes the loan available on the marketplace, where individual investors are able to evaluate the loan before making a decision to invest.

We are interested in encapsulating Lending Club's loan approval and rating assignment process using machine learning algorithms. Lending Club makes publicly available data about the loan applications they receive and the loans that are subsequently financed. We plan to apply machine learning techniques to this data to predict which loans they will approve, what grades they will assign them, and whether the loan will be a good investment.

For loan approval and loan quality prediction, given that our data is labeled and that we will be placing loans in various unordered categories, we will clearly be tackling a classification problem. For loan grade prediction, given that the grades assigned to a loan are an ordered set ranging from A to G, we will clearly be tackling a regression problem. Furthermore, our data set is fairly large. Consequently, suitable methods for both problems need to be able to perform well with large training sets.

# Progress

## *Tech Stack*

Our application is written in Python using a PostgreSQL database and the Flask web framework. We're using software packages from the Flask ecosystem to interact with our database, including the Flask-SQLAlchemy package which allows us to manipulate our records in our code. To handle our numerical calculations, we are using the Python libraries numpy and scipy. To benchmark our results against robust machine learning code, we have also been using the scikit-learn library, which offers out of the box functionality for the algorithms we've been using. After we get results using the code that we've written, we use the scikit implementations to verify that these results are accurate. We've written our code to be modular so that we can easily plug in different implementations to test our code.

## *Algorithms Used*

We ultimately narrowed down our method choices to random forests and ordered logistic regression. Thus far, we have implemented the random forest algorithm that grows multiple classification decision trees at runtime and outputs the mode of the classes predicted by the individual trees. Randomization is injected into the algorithm by randomly sampling with replacement bootstrap sample subsets of the original data set for the growing of each tree and also by taking a random subset of features of size sqrt(m) (where m = the number of features of the original set of features) at each node of the decision tree when looking for the best split. Our implementation involves separate Python modules for DecisionTreeClassifier and RandomForestClassifier, allowing us to use scikit-learn's implementation of DecisionTreeClassifier within our own RandomForestClassifier as a benchmark.

We wrote our decision tree growing algorithm using the CART (Classification and Regression Tree) methodology. CART trees are unique in that they can be tweaked to be used for regression which may be useful when we consider applying our random forest algorithm to the regression problem of loan grade prediction. Furthermore, they use the feature and threshold that minimizes the gini impurity in the resulting subregions when splitting the data set at a node. We chose the gini impurity measure as suggested by Leo Breiman in his original paper on random forests.

| Num trees | Num samples | Num trials | sklearn impl | our impl |
|---|---|---|---|---|
| 100 | 100 | 2 | 71.7% | 71.7% |
| 100 | 500 | 3 | 76.4% | 69.1% |
| 50 | 1,000 | 2 | 77.2% | 76.2% |
| 100 | 1,000 | 2 | 75.2% | DNF |
| 1,000 | 1,000 | 1 | 95.0% | DNF |
| 500 | 10,000 | 1 | 92.2% | DNF |
| 1,000 | 20,000 | 1 | 93.8% | DNF |

**Table 1. Comparison of performance on approval**

| Feature Importances | |
|---|---|
| Field | Value |
| Loan amount | 0.369 |
| Debt-to-Income Ratio | 0.208 |
| Employment Length | 0.082 |
| Zip Code and State | 0.342 |

**Table 2. Feature importances on approval**

# Results

## *Loan Approval v. Rejection*

Our implementation of the Random Forest Classifier (RFC) to the loan approval problem gave us between 69-76% accuracy. For all of our tests, we used an equal number of approved and denied loans in our sample set to ensure the algorithm didn't simply err toward the side of the majority of the samples. The best results were 76% accuracy with 50 trees and 1000 samples, which was the largest forest that we tested. When we halved the number of samples to 500 and increased number of trees to 100, we saw a drop-off in accuracy to 69%. When we tried to run our code with 100 trees and 1000 samples, our code did not finish running.

Comparing our results to those produced by scikit-learn shows that our implementation of the RFC was effective for the approval problem. At 50 trees and 1000 samples, our accuracy was within 1% of the accuracy of scikit's implementation.

Since the scikit implementation runs faster than our code currently does, we were able to test the RFC with larger sample sizes and more trees. Using 1000 trees and 1000 samples yielded 95% accuracy. With 500 trees and 10,000 samples, we had 92% accuracy. And with 1000 trees and 20,000 samples, we had 94% accuracy. This leads us to believe that if we can make our implementation of the RFC quicker and more efficient, we should be able to see similar results for the approval classification.

We also looked at which fields were the most important the approval prediction. We used some functionality within scikit to determine this. We found that the loan amount was the most important predictor at 37%. This was interesting because it may mean that the amount that was requested is indicative of the individual's ability to pay off the loan. Another important feature was the person's debt-to-income ratio, which had an importance of 21%. This makes sense since someone with more debt relative to their income would be a poor choice for a new loan. An individual's employment length had 8% importance, and their zip code and state combined had 34% importance.

| Samples | Trees | Our RF+DT | Our RF+SKDT | SKRF+SKDT |
|---|---|---|---|---|
| 100 | 1 | 59% | 50% | 55% |
| 1000 | 1 | 55% | 56% | 55% |
| 100 | 10 | 52% | 52% | 55% |
| 1000 | 10 | 53% | 56% | 56% |
| 100 | 100 | 62% | 58% | 56% |
| 1000 | 100 | 59% | 61% | 62% |

**Table 3. Comparison of performance on quality**

| Feature Importances | |
|---|---|
| **Field** | **Value** |
| Interest Rate | 0.127 |
| Debt-to-Income Ratio | 0.092 |
| Annual Income | 0.090 |
| Revolving Utility | 0.080 |
| Revolving Balance | 0.079 |
| Months since last delinquency | 0.077 |
| Total Accounts | 0.701 |
| Installment | 0.065 |
| Open Credit Lines | 0.049 |
| Funded Amount from Investors | 0.043 |
| Months since most recent 90-day or worse rating | 0.032 |

**Table 4. Feature importances on quality**

### *Loan Quality*

We decided to tackle the loan quality problem by approaching the simple problem of whether a loan will be fully paid or charged off at completion, simplifying our initial approach to a binary classification problem.

Similar to our approach in the approval problem, we used an equal number of fully paid and charged off loans, and randomly assigning into training and test sets. Unfortunately, our results for loan quality prediction using the RFC weren't as strong as the results of loan approval prediction. Accuracy hovered around 50-60%, which is only slightly better than random guessing and not enough to become a significant advantage for an investor to predict which loans are safe to invest in. Increases in the number of trees did not significantly improve our results.

We also examined the most relevant fields to the quality problem. The interest rate is the most important predictor at 12%. This makes intuitive sense, as a higher interest rate not only indicates a riskier loan, but also is a larger payment per period, making it more difficult for the borrower to make payment on time. The other important relevant fields include debt-to-income, annual income, and statistics about the borrower's credit line.

## Future Steps

We intend continue tweaking our random forest algorithm to improve both its accuracy and efficiency. One possibility we will consider is analyzing the mean class probabilities outputted by all the trees, as this might be a more accurate way of predicting classes than simply taking the majority vote amongst the trees. Furthermore, we want to identify where the bottlenecks are in our code. This will hopefully allow us to run our tests using more samples and trees that we did up to the milestone.

We also plan on properly implementing the out-of-bag error estimate for our random forest algorithm. Since we select the bootstrap sample subsets for each decision tree by randomly sampling with replacement, a third of the samples ends up being left out of the construction of the trees. Consequently, these left out samples can be treated as the test set that we would have if we were performing cross validation. Passing each left out sample down the tree it was left out from will result in a test classification that can be compared to the actual label for that sample. The percentage of mistakes made in our test classifications will be our out-of-bag error. We have most of the code written for generating this error, but we just need to make some slight changes to make sure it is calculated properly.

We still need to implement the appropriate learning algorithm for the prediction of loan grade assignment. Since loan grades are an ordered set of labels, we clearly need a regression model. We are currently considering tweaking our random forest algorithm such that it can be used for regression, which would require us to also create a regressor version of our current classification decision tree growing algorithm, or using ordered logistic regression. Both methods have their merits and we intend to figure out which one will best serve our needs.

For the final submission, we plan to make a web application that will allow users to interact real-time with the application that we have built. This will, for example, allow people to plug in their information to see if they would be approved for a loan. Part of the reason we chose to implement our application in Python was to allow us to build a web server easily that uses our machine learning algorithm. The Flask web framework that we're using is a lightweight framework that will help us set up our website, hosted on Heroku.

# References

**[1]** http://link.springer.com/article/10.1023%2FA%3A1010933404324