Loan Approval and Quality Prediction in the Lending Club Marketplace

Final Write-up

Yondon Fu, Matt Marcus and Shuo Zheng

Introduction

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 riskiness 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 apply machine learning techniques to this data to predict which loans they will approve, what grades they will assign to those loans, and which loans will ultimately 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 face a classification problem. For loan grade prediction, given that the grades assigned to a loan are an ordered set ranging from A1 to G5 (each grade of A - G has subgrades 1 - 5), we face 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.

Implementation

Tech Stack

Our application is written in Python using a PostgreSQL database and the Flask web framework. We use 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 numerical calculations, we use the Python libraries NumPy and SciPy. To benchmark our results against robust machine learning code, we take advantage of the scikit-learn library, which offers out of the box functionality for the random forest algorithms we implement. 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 logistic ordinal regression.

For our classification problems, we implemented the random forest classification algorithm which grows multiple classification decision trees at runtime. 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.

We wrote our decision tree growing algorithm using the CART (Classification and Regression Tree) methodology. Specifically, our CARTs for classification choose 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.

Initially, prediction of a new example was a voting strategy, outputting the mode of the classes predicted by the individual trees. At the suggestion of Professor Torresani, we revisited this problem after the milestone. Instead of simply outputting the mode of all estimator trees, a more robust method is to take a generative approach and output the class that maximizes the mean posterior probability of all estimator trees. This approach ultimately improved our classification results.

A unique feature of the random forest algorithm that we also implemented is the out-of-bag error estimate. Since we select the bootstrap sample subsets for each decision tree by randomly sampling with replacement from the original training set, approximately 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 a pseudo validation set. Passing each left out sample down the tree it was left out from will result in an out-of-bag classification that can be compared to the actual label for that sample. The percentage of mistakes made in our out-of-bag classifications will be our out-of-bag error.

CART trees are unique in that they can be tweaked to be used for regression, which we naturally extended to apply to the regression problem of loan grade prediction. Instead of minimizing the gini impurity of the resulting subregions at each split, the regression trees optimize by choosing the feature and threshold that minimizes the mean squared error of the resulting subregions. Prediction involves taking the mean output of all regression trees, rounding it to the nearest integer, and outputting the corresponding discrete grade. For example, if the mean output of the estimators is 3.4, we round to 3, which corresponds to an output of grade A3.

Our random forest implementation involves separate Python modules for DecisionTreeClassifier/Regressor and RandomForestClassifier/Regressor, allowing us to use scikit-learn's implementation of DecisionTreeClassifier/Regressor within our own RandomForestClassifier/Regressor as a benchmark.

To determine the efficacy of using random forest regression as the solution to our grade prediction problem, we also implemented logistic ordinal regression as a benchmark. Logistic ordinal regression is an extension of the logistic regression method designed to handle ordinal data. It uses a threshold parameter θ as a separator between each class, thus we have K - 1 threshold parameters for K classes, such that $\theta_1 \leq \theta_2 \leq \cdots \leq \theta_{K-1}$. Like logistic regression, which outputs $p(\hat{y}^{(i)} = j \mid x^{(i)})$, logistic ordinal regression outputs $p(\hat{y}^{(i)} \leq j \mid x^{(i)}) = \phi(\theta_j - w^T x^{(i)})$.

We implement the methods necessary for logistic ordinal regression in a separate Python module and as a benchmark, we use an open-source implementation in Python, minirank by Fabian Pedregosa, former lead developer at scikit-learn.

Other Notable Changes

The time required to train our random forest classification was a major issue at the time of our milestone submission. Training our algorithm using a large number of trees and/or a large training set either resulted in long runtime or the algorithm not finishing at all. After the milestone we identified the bottleneck in our code by profiling our code. We had previously built all our decision trees in parallel during training time using joblib, a Python module with tools for parallel processing. After profiling our code, we realized that our parallel processes were frequently trapped in a deadlock as they tried to access the same resources, i.e. the same samples in the training set at the same time. As a result, each process had to wait a lengthy amount of time before finally being able to

access the resources needed to build a decision tree. We fixed this issue by instead building our decision trees one at a time, thereby eliminating the possibility of a deadlock.

Using the Flask lightweight web framework, we built a web application that allows users 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 easily build a web server that uses our machine learning algorithm. We only offer the option of loan approval vs. rejection prediction in our web application because our accuracy for loan grade prediction was suboptimal and loan quality prediction would require information that would not be available to an average user before the maturity of a loan.

We wanted to provide quick predictions for users, so retraining a random forest classifier for every query was not an option. For model persistence, we used the Python module pickle to serialize both our DictVectorizer object (a scikit-learn class that performs binary one-hot coding on features that have string values such that each string value of a feature is represented by a boolean valued feature) and our random forest classifier which was locally trained using 25 trees and 500 samples. We saved a copy of the DictVectorizer object used to transform our training set because it is needed to also transform any data inputs from user queries according to its learned vocabulary. We ultimately decided not to deploy our web application into production because the Python module pickle is not suitable for loading very large recursive objects (all the decision trees in our random forest are constructed recursively) in production environments, an issue we did not get a chance to address due to time constraints. See the README of our project for instructions to run our web application locally.

Results

Loan Approval vs. Rejection

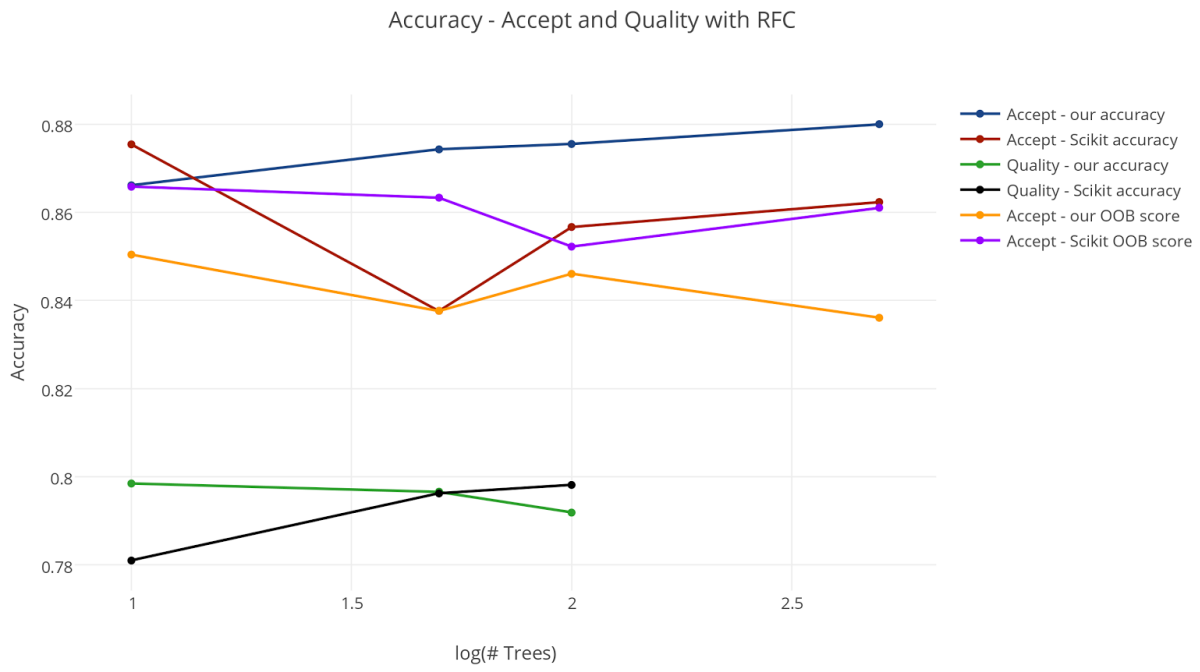
Our initial implementation of the random forest classification (RFC) to the loan approval problem gave us between 69-76% accuracy. For all of our tests, we used an equal number of approved and denied loans in our sample set to ensure the algorithm didn't simply err toward the side of the majority of 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 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 had the time to implement our RFC in Cython to be more efficient in memory and processor time usage, we should be able to see similar results for the approval classification.

Out of curiosity, we also looked at which fields were the most important to approval prediction, using some functionality that scikit provided during its run with our data. 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.

Following our second implementation of RFC using a posterior probability strategy rather than a voting strategy, we saw an increase in our accuracy of ~10%, bringing our best accuracy up to 87% with 50 trees and 1000 samples. After eliminating our previous multithreading bottleneck, we were able to increase the number of trees and/or the number of samples. We yielded 89% accuracy with 500 trees and 1000 samples, and 87% with 50 trees and 5000 samples.



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.

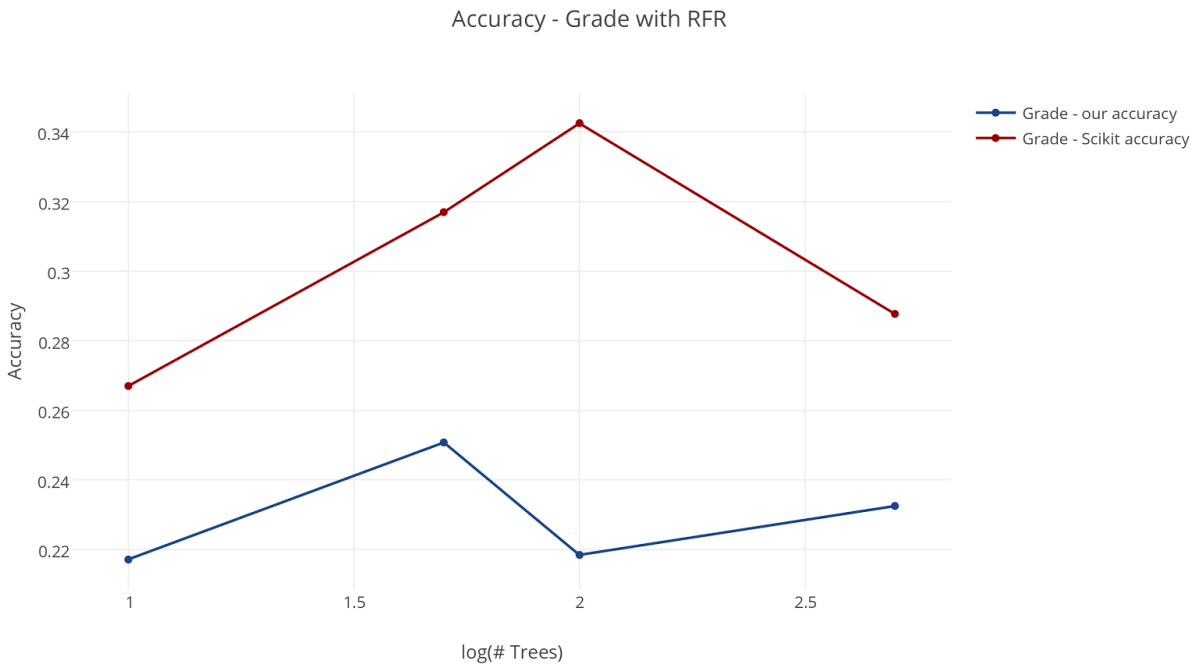
Similar to our approach in the approval problem, we used an equal number of fully paid and charged off loans, and randomly assigning into training and test sets. Unfortunately, our initial results for loan quality prediction using RFC (voting strategy) weren't as strong as the results of loan approval prediction. Accuracy hovered around 50-60%, which is only slightly better than randomly 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.

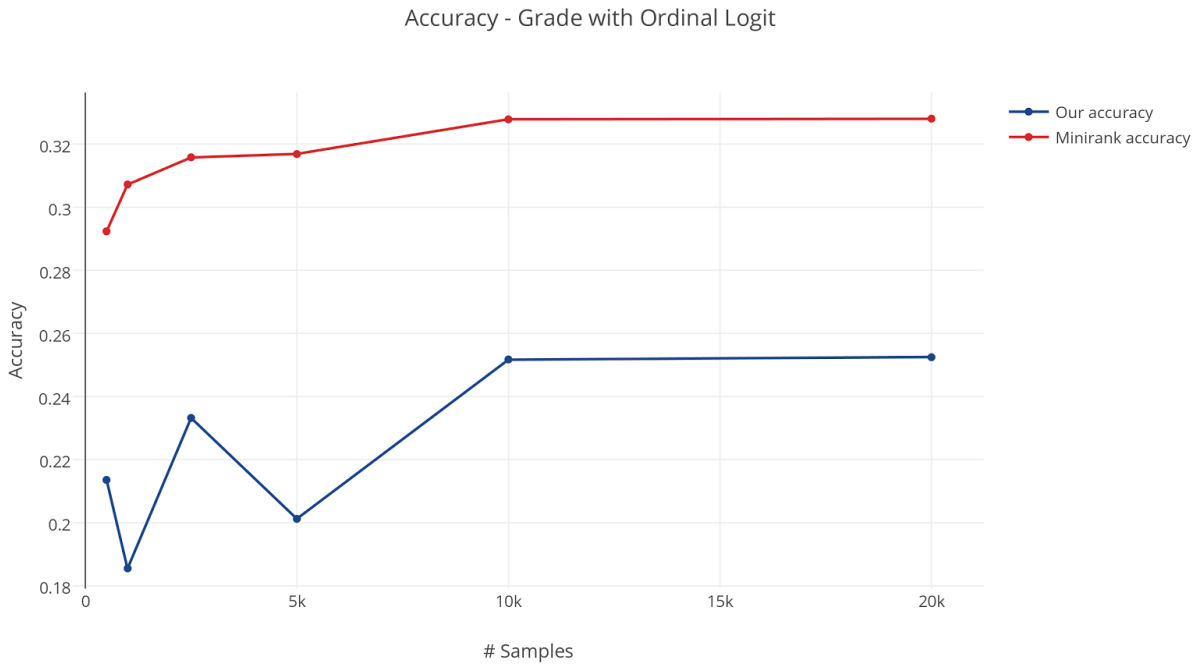
Following our revisit of RFC, we significantly improved the accuracy of our predictions on this problem. Using the posterior probability strategy rather than the voting strategy increases our accuracy to ~80%. Both with 100 trees and 500 samples, and with 50 trees and 1000 samples, we predicted with 79% accuracy.

Loan Grade

Unlike the previous two problems, we were not as successful with loan grade prediction. All accuracies reported with our regression problems are expressed as the R^2 value of the results (how much of the test data sets' deviance from its mean is explained by our model). Using the random forest regression (RFR) algorithm, we were only able to predict loan grade outcomes with ~20% accuracy with a high variance across tests. Increasing the number of trees or the number of samples did not consistently improve our accuracy, or even decrease the variance of our accuracy and give a consistent result.



Using the logistic ordinal regression algorithm, we were able to predict loan grade outcomes with a slightly higher accuracy of ~22% with a slightly lower variance across tests. Similar to RFR, we also reported the accuracy of our logistic ordinal regression algorithm using the R^2 value of the results. However, the improvements in our results compared to those of random forest regression are not very significant which demonstrates that the loan grade prediction may not be a problem well suited for machine learning techniques.



Given that loan grades are both discrete categories and an ordered set, it is not particularly clear whether classification or regression methods work the best. Our poor results may also reflect the possibility that the assumptions made by RFR or logistic ordinal regression do not fit the data well. One such incorrect assumption may be that logistic ordinal regression assumes that the hyperplanes separating all the classes are parallel.

Conclusions

Our surprisingly accurate results with Lending Club's approval prediction seems to indicate that a similar algorithmic or machine learning approach backs their decisions. As for the next step in the loan's life, our grade prediction results were subpar. This suggests that perhaps after an approval has been made by a machine, human judgment is brought into the process to critically evaluate a loan's risk and assign its grade. Naturally, since risk assessment is a very difficult and complex problem and may vary across different human judges, this judgment may be made in a way that is hard to model and learn with machine learning. While our methods may give a prospective borrower some precognition into his or her loan approval, we cannot provide much help after that. It'll be up to the borrower to best present his information to make him or her seem as riskless as possible to obtain a better grade.

As for the lenders, our quality prediction results also suggest good news. Unfortunately for us and for them, the necessary information for this prediction is not available until a loan's maturity to lenders not invested in that loan. If Lending Club is willing to disclose more information about loans on its marketplace, we can predict quite accurately how a loan ultimately performs. This information would allow investors to put most of their money into safe investments and sleep better at night.

References

[1] https://www.lendingclub.com/info/download-data.action
[2] http://fa.bianp.net/blog/2013/logistic-ordinal-regression/
[3] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
[4] http://scikit-learn.org/stable/modules/tree.html