Finding Bias in Human-Filtered Loan Applications

Motivation

On the Lending Club website [1], individuals or institutions can lend money to borrowers. The Lending Club is responsible for reading loan applications, and filtering out ones that they think might not be able to pay back, and setting an interest rate based on how safe the investment seems.

The Lending Club makes money off of a small cut that it takes when investors recoup each portion of their investment plus interest monthly. It's apparent from this model that both the Lending Club and the actual investors participating on the website have similar goals, since the financial incentives for those two parties are linked. It is in the best interest of both of those parties that the investors do not put money into borrowers who won't be able to pay back. That is why there are two stages at which borrowers are filtered out - the first is the initial screen of the application by the Lending Club, and the second is the fact that investors get to choose who they want to invest in.

The typical loan application problem is centered around taking a mix of accepted and rejected loan applications and trying to be able to predict if a different loan application should be accepted and rejected. My project is similar, but does have a notable difference. First, I'm only considering applications that have already been accepted by a human reader. Instead of using whether or not an application was accepted, I'm actually using the status of whether the loan was paid off and whether it was paid back in a timely manner. This approach has the advantage that is actually based on what happens with the investment and whether the money lent is recouped in a timely manner, rather than just trying to predict whether a human would accept or reject the application. The results of this project may point to biases in the way that humans read loan applications.

This project is of the utmost importance to investors on the Lending Club platform, because investors would likely be more wary of investing in someone if they had a suspicion that the loan won't be paid back on time. People defaulting on loans doesn't actively make the Lending Club lose money, but it does prevent the Lending Club from collecting a cut of the recouped investment from the investor. Further, if this happens at a large scale, investor confidence could decrease, and the Lending Club's business could dry out because investors would no longer believe they can reliably make back their money

Data

The Lending Club had data from several years, but I chose the earliest year that they had complete records for. I needed complete records so I had a lot of room to play with features and there was an adequate amount of data. I needed the earliest year possible with that criterion because I wanted to give as much time as possible so the loan status can be known. This was an important decision given that my data-set takes a while to mature, because loans are paid back over a period of time. I ended up finding the optimal data-set in the records The Lending Club had for loan applications from the year 2015.

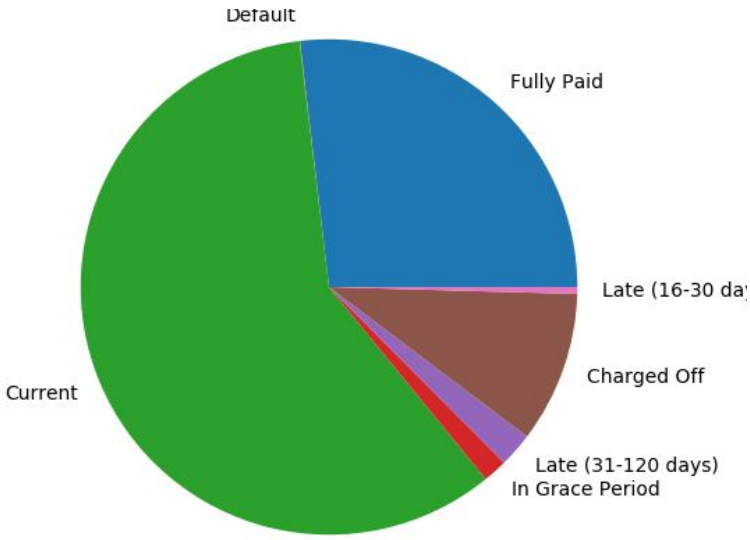

Figure 1: Pie chart with the breakdown of loan statuses in the 2015 Lending Club accepted loans dataset

The variable I'm trying to predict is the loan status. From the entire dataset I have, here is the breakdown of loan status: 143111 for Fully Paid, 64 for Default, 211830 for Current, 5708 for In Grace Period, 7668 for Late (31-120 days), 51062 for Late (16-30 days), and 1652 for Late (16-30 days). It's pretty clear that Fully Paid and Current loans make up the lion's share of this dataset, which is even more clearly seen in the pie chart below. This was an interesting bit of exploratory analysis, because lopsidedness over the loan status variable required slight modifications to my approach, which I'll expand on later.

A good amount of time was spent understanding the dataset. First of all, the column headings weren't entirely clear, so I consulted the extra document they had that showed the mapping between the column names and a lengthy description of what exactly the name means. I selected features based on a combination of identifying features that were likely to give an indication as to whether loans would be paid back in time and columns with enough entries filled in. It's easy to justify the annual income of the borrower, because a borrower with a job that makes a good living is more likely to be able to pay back the money. This features is obviously important enough that I'd expect that the people at the Lending Club who read the applications cared mainly about this features. This would imply that I might see fairly similar incomes across the board, but nonetheless, I thought it was important enough of a feature to include anyways. It's expected that the interest rate should be inversely related with the likelihood that somebody pays back the investment on time. A less obvious feature that still might provide some signal is the installment, the monthly amount that the borrower has to pay to keep up with the sum of the principal and interest. Thinking through the relationships between the features and what I understand of human behavior related to those features was how I ended up at the feature list I used.

In terms of feature engineering, there were a couple cases where I had to use some stuff we had learned in the class. For example, one feature was a flag corresponding to whether the person looking for the loan was on a hardship plan, and that was a "Y" for yes and "N" for no. I went through each entry and changed "Y"s to 1s and "N"s to 0s.

I noticed some columns had all entries missing and I of course had to leave those off. However, there were some columns where a very small percentage of the entries were missing. Initially, I decided to take only rows that only had no entries missing for the columns corresponding to the features that I was using in my model. When I started off with a far smaller number of features, this was fine, and I still had plenty of data to work with, around 75% of the dataset, which was around 300k examples. When I added

more features, I realized that this had dropped off to only around 20k examples. This drop off in the number of examples, along with the fact that missing entries could mean something about the data, led me to using matrix completion to fill in the missing entries. The matrix completion task was solved via Principal Component Analysis to get a low rank approximation of the matrix, using the LowRankModels package for Julia. Since I was running everything on my laptop, this data imputation step took a while, a few hours, but I figured it was well worth the time to drastically increase the size of my data set (with the larger list of features) and also be able to learn the trends that might be indicated by missing data.

When I did imputation, I considered all rows where I had the loan status, so I could actually use them for my learning problem. This ended up at around 300k rows, out of the initial 420k or so rows.

Feature Set

Table 1: Data Types of Features and Feature Engineering

Feature	Input/Output	Example entry	Data Type Before Feature Engineering	Feature Engineering	Example entry after feature engineering
Loan Amount	input	20000	Numerical		20000
Annual Income	input	55000.0	Numerical		55000.0
Debt-to-Income Ratio	input	29.15	Numerical		29.15
Earliest Credit Line	input	Sep-94	String	“YYY-XX” -> XX	94
Loan Term	input	36 months	String	“XX months” -> XX	36
Interest Rate	input	6.49%	String	“XX.XX%” -> XX.XX	6.49
Months Since Most Recent Installment Accounts Opened	input	12	Numerical		12
Loan Grade	input	C	Ordinal	A -> 0, B -> 1, ... , E -> 4	2
Hardship Flag	input	Y	Boolean	Y -> 1, N -> 0	1
Installment	input	612.89	Numerical		
Loan Status	*output*	*Late (31-120 days)*	*Nominal*		*Late (31-120 days)*

With the Earliest Credit Line feature, I had to be careful to deal with the format. There were entries corresponding to months in years 2000 and later, like “Aug-09”, and entries corresponding to months in years before 2000, like “Oct-82”. I stripped off the text and converted to integer to get 9 and 82, respectively, for those two examples. Then I added 100 to the integer if the integer was less than 17. The idea there is that I don’t want years 2000 and later to be counted as earlier than years before 2000, simply because only 2 digits were used. The cut-off of 17 was used because the current year is 2017, so having the 2 digits be an integer larger than 17 means that the datapoint is meant be a year of the form 19XX. Not catching this result could have compromised the integrity of the models I obtained.

A problem that featured heavily in my project was the problem of balance. I was using loan applications that had already been accepted by a human, and I was trying to find cases where human bias might have led to the incorrect decision (to accept when it shouldn’t have been, a false positive on the part of the human), which could affect the bottom line of the lender. Since these applications are pre-filtered, the vast majority of them are going to be paid off in a timely way.

Training Process

Before actually training, I made sure to split the dataset into training and validation sets, so as to ensure I didn’t make the error of training on what I was going to be validating. I used a pretty standard 70%/30% split of training and validation, respectively.

Results

Table 2: Results for when training set had 14566 examples, and 3 output classes were used (very late, late, and on time).

Model	Test Set Accuracy	Train Set Accuracy	% Very Late Predicted Correctly	% Late Predicted Correctly	% On Time Predicted Correctly
Linear Discriminant Analysis	0.86	0.87	0.03	0.05	1.00
Quadratic Discriminant Analysis	0.86	0.86	0.07	0.09	0.98
Random Forest (class weight balanced)	*	*	*	*	*
Random Forest (not using class weight)	0.87	0.86	0.00	0.00	1.00
Naive Bayes	0.83	84	0.13	0.00	0.95
Support Vector Classification	0.86	0.99	0.00	0.00	1.00

Table 3: Results for when training set had 84351 examples, and data was imputed for missing entries using rank-1 approximation via PCA using LowRankModels package. 7 output classes, sampling was done to choose rows so that the proportion of Current and Fully Paid examples didn't heavily outnumber the examples from the other classes.

Model	Test Set Accuracy	Train Set Accuracy	% Default correct	% Current correct	% In Grace Period	Late 31-120 days	Charged off	Late 16-30 days	Fully Paid
Linear Discriminant Analysis	0.39	0.39	0.00	0.01	0.00	0.00	0.99	0.00	0.00
Quadratic Discriminant Analysis	0.29	0.30	0.00	0.13	0.07	0.00	0.61	0.15	0.05
Random Forest (class weight balanced)	0.21	0.21	0.11	0.01	0.02	0.13	0.37	0.08	0.35
Random Forest (not using class weight)	0.40	0.41	0.00	0.23	0.00	0.00	0.84	0.00	0.00
Naive Bayes	0.10	0.11	0.00	0.05	0.02	0.83	0.03	0.00	0.11
Support Vector Classification	0.20	0.42	0.07	0.25	0.16	0.15	0.19	0.09	0.18

Table 4: Results for when training set had 84351 examples, and data was imputed for missing entries using rank-1 approximation via PCA using LowRankModels package. 3 output classes, sampling was done to choose rows so that the proportion of On Time examples didn't heavily outnumber the examples from the other classes.

Model	Test Set	Train Set	Very Late	% Late	% On Time

	Accuracy	Accuracy	Predicted Correctly	Predicted Correctly	Predicted Correctly
Linear Discriminant Analysis	0.48	0.48	0.03	0.00	0.98
Quadratic Discriminant Analysis	0.42	0.42	0.80	0.03	0.23
Random Forest (class weight balanced)	0.36	0.36	0.30	0.24	0.52
Random Forest (not using class weight)	0.50	0.51	0.51	0.00	0.58
Naive Bayes	0.46	0.47	0.85	0.01	0.18
Support Vector Classification	0.39	0.57	0.35	0.26	0.45

I used Scikitlearn.jl [2] to leverage their implementation of these models. Linear Discriminant Analysis (LDA) is commonly used as a dimensionality reduction technique, but unlike Principle Component Analysis, LDA is concerned with finding the basis vectors that maximize the separation between the output classes [3]. Quadratic Discriminant Analysis (QDA) is similar, except it doesn't have the requirement that LDA has that the Gaussians for the each class have the same covariance matrix. QDA not having that requirement allows it to have quadratic decision surfaces [4]. A Random Forest classifier creates multiple Decision Tree classifiers on different bootstrap samples from the dataset and averages among these to come to a decision. Naive Bayes assumes that every feature is independent and uses Maximum A Posteriori estimation to estimate the probability that a feature vector belongs to a particular class [5]. Lastly, Support Vector Machines for Classification, as discussed in class, use hinge loss and are designed to make the yield the highest possible safety margin, at the expense of not preventing severe mistakes. The only non-standard parameters I used were trees of max depth 5 and 10 decision trees for the Random Forest classifier and gamma=0.1 and C = 1 for the SVM classifier.

You'll see that Table 3 has 7 output classes considered, and Tables 2 and 4 only have 3 output classes considered. I started off with using the seven output classes that were seen as loan statuses in the dataset. It was clear that some of the classes were very similar to each other (like Full Paid to Current, and Late 31-120 days and Late 16-30 days), and I figured this might make the learning task more difficult than necessary. Three categories (On Time, Late, and Very Late) would still give enough information to be helpful to lenders, and would be easier to learn and encapsulate in a model. On Time had all examples with output classes of Fully Paid or Current, Late had all examples with output classes of Late 16 - 30 or Late 31 - 120, and Very Late had all examples with output classes of Default, Charged Off, or Grace Period. As you can surmise from the output classes corresponding to Very Late, the Very Late moniker is really meant to describe loans that are most likely never going to be paid back.

Because of the balance problem, testing set accuracy was not a good way of seeing how a model performed, especially since in the small percentage of cases where loans are defaulted on or paid back late, the lender can lose a lot of money. Stated another way, the balance meant that the system could get around 90% accuracy by just predicting the loan will be paid back in a timely manner, but that is not a very enlightening or useful result. A better result would be a model that can correctly identify 90% of the loans that were not paid back in a timely manner, because that would be a result that would save lenders money. You can see this in the results from Tables 2, 3, and 4. The models that have lower test set and training set accuracies tend to be better at predicting the less common classes like Very Late, which is what we want. The models that have higher test set and training set accuracies tend to nearly always predict that a loan will be paid back in time, which is not optimal given the negative financial impact of people defaulting on loans.

So, I used a confusion matrix for the test set to understand how the models were doing, rather than relying on just the accuracy on the test set. That way, I can understand at a deeper level how the actual loan status matches the predicted loan status, at the granularity of the classes. The last three columns in the tables below come from the confusion matrix. For the row of the confusion matrix that corresponds to loans that are actually Very Late, I divide the number that were predicted as Very Late by the total number of loans that were actually Very Late. I calculate similar things for the classes of late and on time.

One suspicion I have is that the low rank approximation was not quite as good as it could have been. Using the LowRankModels package, I tried to fit a rank-1 approximation to fill in missing entries in the table and it took around 5 hours, since it was operating on a very large array. After 100 iterations, the objective function reduced from 2.8e10 to 2.4e9. The reason the objective values are so high is because the array is so large that once you add on the loss for all of the examples, you get a large number. However, if it weren't a prohibitively long process, I would have run the algorithm for more iterations because the loss function was not yet to the point where it was finished decreasing and found the minima. I think running this for more iterations would have led to better results for the models in Tables 3 and 4.

Another trend that is pretty clear from Tables 2, 3, and 4 is that when the training and testing accuracies were quite high (close to 80%), the percentage for picking Current or Fully Paid (for 7 output classes version) or picking On Time (for the 3 output classes version) is very high (close to 100%), and the other classes have abysmal accuracies. There are far more Current and Fully Paid loans than any of the other classes combined. So, the cases with absymal accuracies with regards to other classes come when the model decides to always return Current or Fully Paid or On Time. This speaks to the balance of the dataset. There were a couple of things I tried to resolve this problem. I discovered from the Scikitlearn.jl API that some of the algorithms, like Random Forest and Support Vector Machine for Classification have a parameter called `class_weight` that can be set to "balanced" so that the proportion of examples in each output class is drawn from the training data and used to better inform the model. So, the first thing I tried was to use that parameter. For Tables 3 and 4, if you take a look at the rows for Random Forest with class weight and compare it to the rows for Random Forest without class weight, the model that uses class weight has more level performance across all of the output classes, rather than just being excellent for Current/Fully Paid/On Time. Another thing I tried to help further sovle the problem was introducing sampling of examples from the dataset. You'll notice that the problem is more apparent in Table 2 than it is in Tables 3 and 4 because for 3 and 4, I tried sampling the rows so as to get a more balanced distribution of examples for the output classes. This idea of the accuracy being balanced for the output classes as opposed to predicting On Time for everything is an important thing, because lenders care more about a loan application being incorrectly labeled as one that will be paid back on time. As such, it's important that the model doesn't always recommend that a loan application will be paid back on time or early, because that model would lose lenders lots of money. Denying someone a loan or bumping

up the interest rate on a loan is far preferable for a lender than extending a loan and having the loan default.

It deserves to be reiterated that the dataset I used contained loan applications that had already been accepted by a human. You can imagine that the humans reading the applications are looking at most of the features that I used in my model. Given that the reader of the applications approved these applications, the assumption is that a human would expect 0 incidences loans classified as Late or Very Late, because it would be foolish for them to accept those kinds of applications. Given that, I think the some of the results seen in Table 4 heavily outperform a human at learning trends from the applications to understand risk factors that distinguish good and bad applications. This speaks to the heart of this project, which is trying to identify if any human biases in the application can be eliminated by using a model like the one I used in this assignment.

Improvements

There are a few ways I think the data analysis project could have gone better. One way is if I had used computing resources that I had available to me, either the ORIE server we were given access to or AWS EC2 instances. This would have sped up iteration time, so as to allow more approaches to be attempted.

Another thing I considered was that using Grid Search might have led to better results because we could tune the parameters for the algorithms being used. This, however, leads back to computing resources, because Grid Search can be slow since you have to train a model for every grid point.

If I had more time on this project, I think it would have been really interesting to use word embeddings/vectors on the emp_title (Employment Title) column of the dataset. This would have been really interesting because this seems like the only field in the loan application that the applicant writes from scratch. This means the format is not very standard and would have to be standardized (there are entries like “TEAM LEAD, QUALITY ENGINEER” versus entries like “Educator”, and there are entries where either the applicant is not telling the truth or is omitting some information (somebody claiming to be “President”). On another note, the employment title might be able to tell a lot about whether a person who receives a loan is likely to be able to pay it back. I already have annual income as a feature, but I think having the Employment Title would add another dimension to that. I would think that the people who have higher up positions (leaders, managers, etc.) would have more stable jobs than lower-level positions (engineers, individual contributors) and therefore be more likely to pay back loans. This would have been a really interesting angle to look into.

I would trust some of the models (such as the class weight balanced Random Forest seen in Table 4) I worked with to function in a production system. A positive aspect of dealing with loan applications is that you can be conservative and avoid risk. As long as a production system defaults to rejecting applications, there is no threat of losing lots of money. Especially considering that my project is centered around taking pre-filtered human-accepted applications and applying another filter to sift out even more bad applications, I’d say that selecting the best model from my project could exclusively help the bottom line for the Lending Club and the investors that participate on that website. The one caveat there is that none of the models that always (or nearly always) predict that a loan will be paid back in full on time should be chosen. This is obvious, given the constraints of the problem I’m trying to solve. The performance metric to optimize on is the percentage of the Very Late or Late applications that are identified properly, so as long as a model that optimizes for those two metrics is chosen, I see this project as a boon to investors. One of the models I used identified 80% of the Very Late applications correctly and 3% of Late applications correctly and another model I used identified 30% of the Very Late applications correctly and 24% of the applications correctly. Given that originally, 0% of the Very Late and 0% of the Late applications would have been identified before an investor actually lends money to a borrower, I would say that my system could be used to great effect by investors.

References

[1] https://www.lendingclub.com/public/investing-faq.action
[2] http://scikitlearnjl.readthedocs.io/en/latest/
[3] http://sebastianraschka.com/Articles/2014_python_lda.html
[4] http://scikit-learn.org/stable/modules/lda_qda.html
[5] http://scikit-learn.org/stable/modules/naive_bayes.html