Classifying Potential Loans from Lending Club

Problem Statement and Data

Lending Club receives many loan applications. The goal is to try to use data on existing loans to build a model to predict whether a loan applicant is likely to be a customer that pays the loan off and makes timely payments or one that gets behind on the loan and/or defaults on the loan. Creating a model to do so would streamline the process for loan approval.

The data comes from https://www.kaggle.com/husainsb/lendingclub-issued-loans and contains two separate sets of data. The first, used as training data contains information on loans from 2007 until 2015. The second, used as test data, contains information from loans from 2016-2017.

Each set of data includes a number of features about the loan and loan applicant. In addition it has the current status of the loan, which is what was used to great the target for classification in this problem

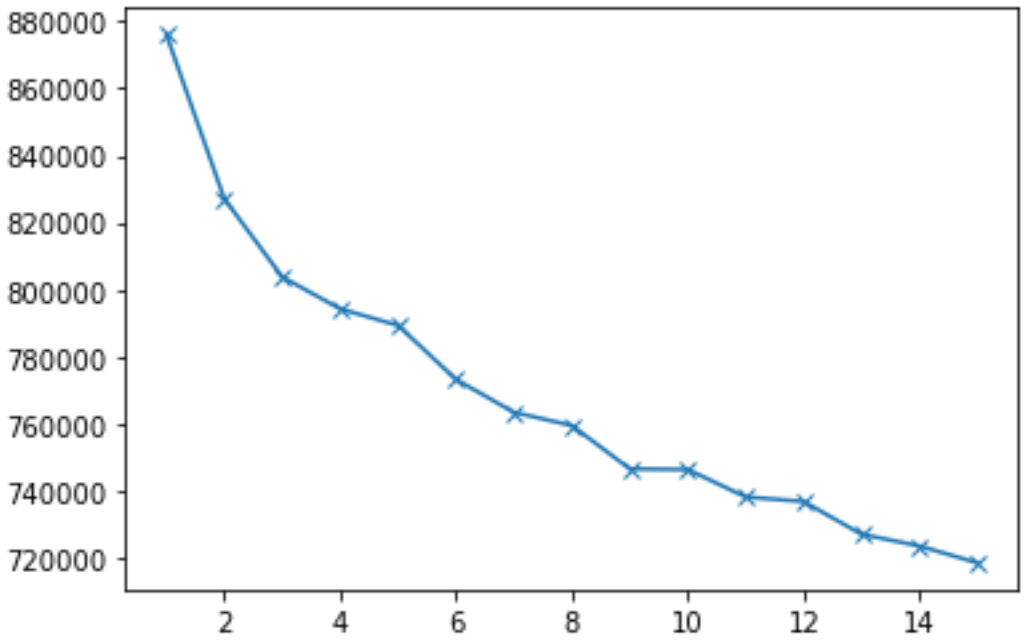
Data Wrangling

The data itself had several challenges just to get into a usable form. The first thing that was done was drop identifying features that were not predictive. (These included ID and member ID). The next issue was many features had large numbers of missing values. While it's hard to use a feature that has mostly missing values, in many cases there was a perfectly valid possible reason for values being missing. There were features such as "inq_last_12m" which was the number of inquiries in the last 12 months. We assumed that missing values there were simply no inquiries. There were also ones about "joint_dti" and a couple of other fields that only applied to joint loans.

One of the fields that was missing for most of the records was one called "desc" which were descriptions of loans. If there were more values present, NLP analysis could be done to try to get useful information from these. However, they were so often missing that it wasn't worth really going forward with using them. We also had fields like "mths_since_last_delinq" which was months since the applicant's last delinquency. This was often missing as well, since if an applicant had no delinquencies there would be no value to put for this. A possible useful replacement feature that could apply to everyone would be fields like "Number of Delinquencies in the last X months" for various values of X. This would apply to everyone, and assuming that older delinquencies would be less relevant, hopefully the machine learning model could apply that. In our case, given this feature, we simply imputed a very large number for missing values. By being far above the values that actually showed up, if this had a noticeable effect on the outcome models

could split along that feature easily. We did have one feature, “open_il_6m”, that only appeared in our training data, but not in the test data, so we did have to drop this feature.

In addition to the fact that the “desc” feature was missing almost all the values, there were two other features related to that. One was called “purpose” and the other was called “title”. The “title” field had many different values and if we wanted to work with it, we would have had to cluster different titles as similar. Luckily the “purpose” field had only a few fixed possible values, so we could use that field to represent the information about why the loan was being requested. Another similar situation was the feature of “emp_title” which was the employment title. There were many different values for these, some extremely similar (for example “Registered Nurse” and “LPN Nurse”, or “Consultant Engineering, Inc” and “IT Consultant”). There were too many unique values to try to use these as is, so we wanted to do a clustering analysis on this feature. The approach we took do this was as follows. We first put “other” for any missing employment titles. We then converted to lower case to avoid any case issues. We used tfidf, that we trained on the training data, to convert the titles into vectors. We then wanted to use a K-Means clustering to group these titles together. When we looked for an “elbow” to figure out the best K, the graph did not show an obvious elbow (see below)



Based on this, our best guess to use was to use k=10 for our K-means clustering. This gave us a reasonable looking way to cluster the employment titles as seen when we look at the first few titles in each category as we see below:

Label 0

1 Ryder
3 AIR RESOURCES BOARD
4 University Medical Group
5 Veolia Transportaton
6 Southern Star Photography
Label 1
17918 Consultant Engineering, Inc.
27724 Expert Consultant Inc
34806 Self Employed Consultant
35446 IT Consultant
36048 Consultant Engineering, Inc.
Label 2
20538 cellular sales
26016 C & B Sales and Service
35324 Advantage Sales & Marketing
37710 F&C Truck Sales and Service
42546 Area Sales Manager
Label 3
42543 Registered Nurse
42564 LPN nurse
42598 Registered Nurse
42654 Registered Nurse
42655 Registered Nurse
Label 4
42537 MANAGER INFORMATION DELIVERY
42550 PARTS MANAGER
42553 Project Manager
42555 Operations Manager
42570 Manager
Label 5
42548 Teacher
42577 Teacher
42578 Teacher Coach
42582 Teacher
42607 Teacher
Label 6
7764 engineer profiles
9566 U.S. Army Engineer R&D Center
17688 Jefferson County Engineer's Office
21705 Engineer Solutions & products
42539 aircraft maintenance engineer
Label 7
0 other
2 other
8 other
30 other
42 other

```
Label  8
5356           Driver
42549          driver
42647     Truck Driver
42660          Driver
42662    Forklift driver
Name: emp_title, dtype: object
Label  9
42866          owner
45918          Owner
46638   Owner / Operator
47036          Owner
48034          Owner
```

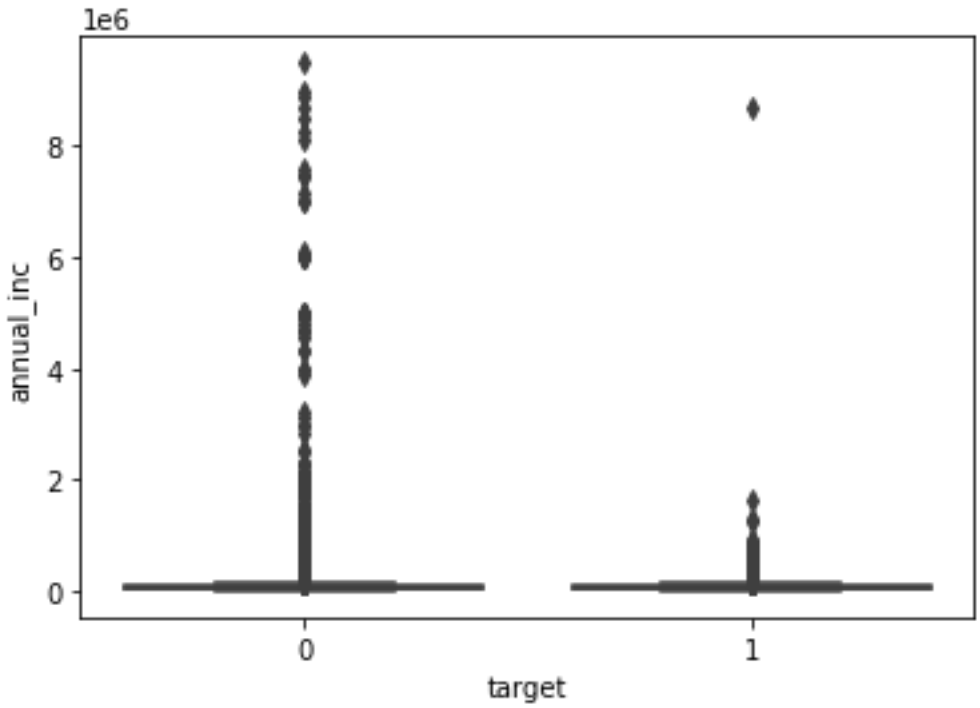
It wasn't necessarily going to be a perfect way to classify each employment title, but it reduced the number of groups of titles to a workable number.

Our other biggest challenge when looking at the data is that there were several fields that were dates. The problem we had there was that the training and test data were from different time periods, so absolute dates couldn't really be compared. The solution that was used was for each of the dates, compute a relative time difference from that date to the date the loan was issued. This allowed these to not be dependent on the date of the loan themselves and gave us data that could be compared across both the training and test data. After these dates were converted to time relative to the loan issue date, we just dropped the loan issue date, as again it couldn't really be used predictively when the set of possible dates in the training data would not even intersect the set of dates from the test data.

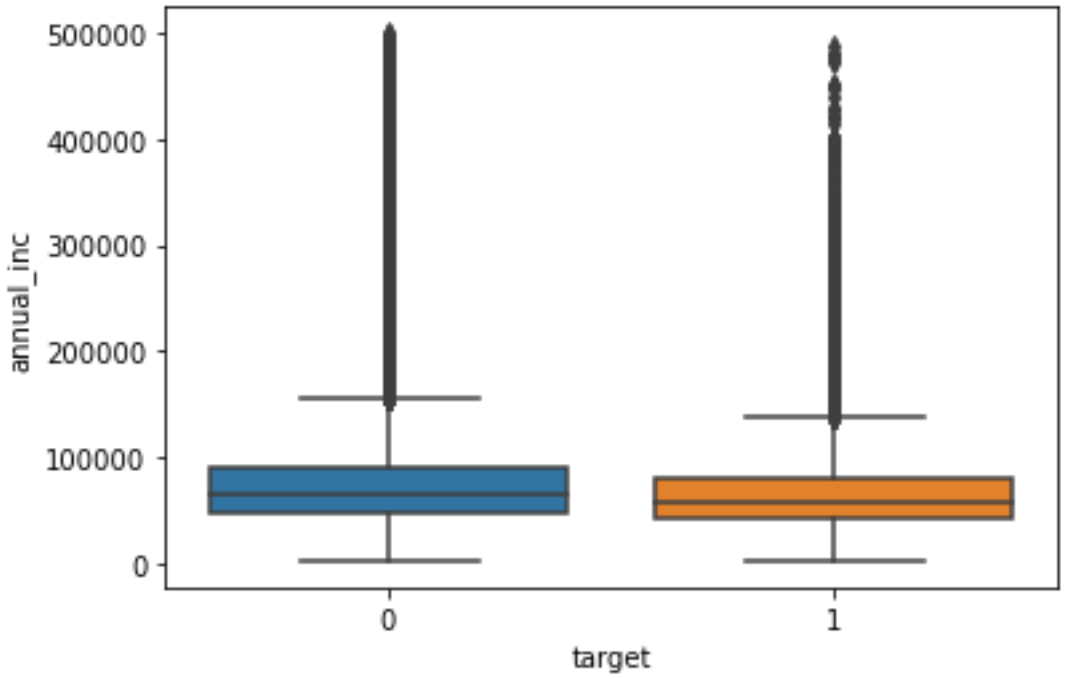
Exploratory Data Analysis and some Preprocessing

Once we had a complete set of data that we could try using for prediction, we wanted to look to see what trends or relationships could be seen, and if we could see differences between the borrowers that were "good" and those that were "bad". The actual loan status had several different values. Our definition of "bad" was any loan that was charged off, in default, or late 31-120 days. On time, paid off, or in grace period were considered "good". This definition is somewhat arbitrary, but really the only gray areas were "grace period" (16-30 days late) and the "31-120 days late" categories, and it seemed reasonable to split them up this way.

One of the first comparisons we looked at was income data for those that were negative class (“good”) or positive class (“bad”). Looking at the whole data wasn’t very easy to do because there were a lot of outliers



We can see a lot more people in high incomes in the negative class but the distribution is hard to read. Just to potentially see more detail we looked at the people whose income was under 500000



We can see a slight difference here with the negative class having a slightly higher median income. Overall we could see that as well

Positive class income:

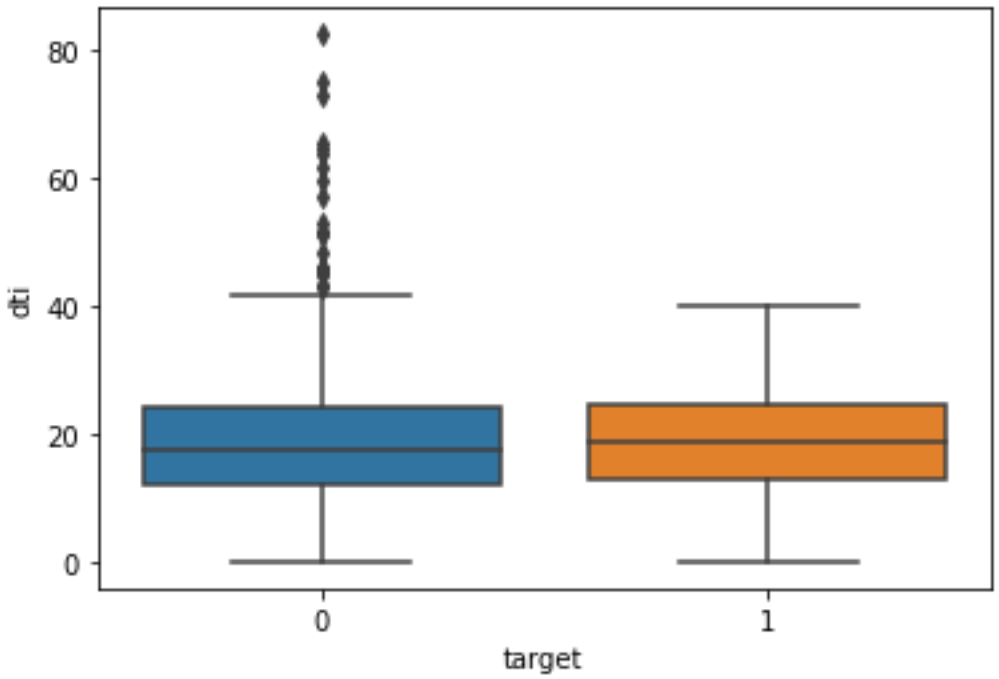
count 5.881900e+04
mean 6.615066e+04
std 5.520877e+04
min 2.000000e+03
25% 4.100000e+04
50% 5.700000e+04
75% 8.000000e+04
max 8.706582e+06

Negative class income:

count 8.285550e+05
mean 7.567356e+04
std 6.527481e+04
min 1.896000e+03
25% 4.600000e+04
50% 6.500000e+04
75% 9.000000e+04
max 9.500000e+06

There was a difference here, where the median income for those in the positive class was $57000, and the median income for the negative class was $65000.

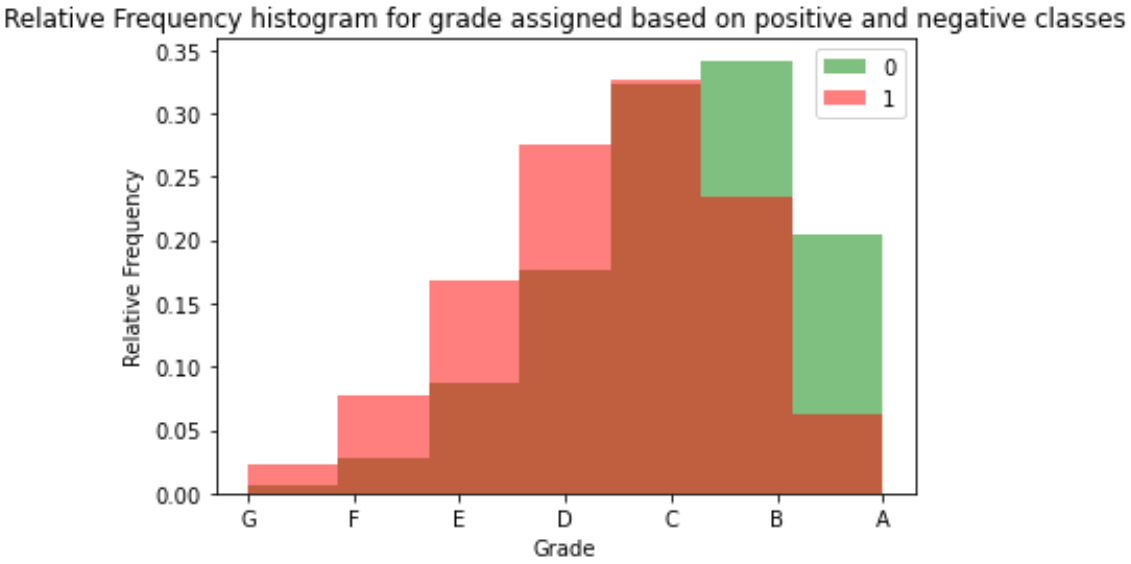
Another feature we wanted to compare was DTI (Debt-To-Income ratio). Having a high DTI could impact the ability to pay back another loan. We wanted to see how the DTI compared between the two groups. One of the first things we found when looking were some very strange looking values for the DTI. We found a few with a DTI well over 300, which doesn't seem very realistic (As an example someone with an annual income of $50000 would have a debt of $15000000 with that kind of DTI). We chose to drop those records as having such a strange value made them unreliable. After we dropped those we could look at the two groups distribution of DTI



We see here that while the negative class contains a fair number of outliers the median DTI for the negative class is a bit lower (17.59 compared to 18.55), but it's not clear that we have a statistically meaningful difference between the distributions.

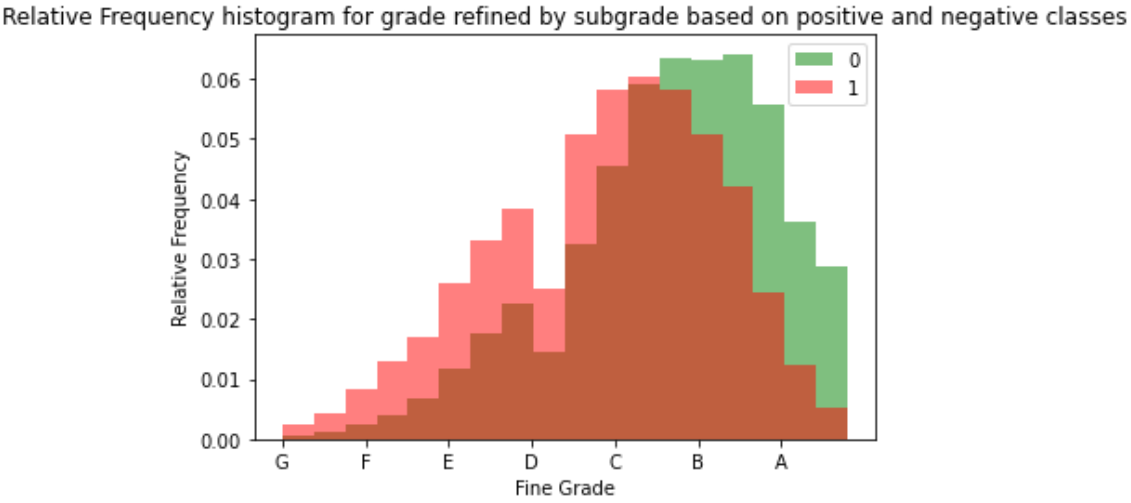
One feature we saw that clearly had a large correlation to rates of bad loans was the credit grade and credit subgrade. To each loan applicant, Lending club assigned a letter grade of A-G. Each of these was further divided into 5 subgrades (e.g. A1-A5). We wanted to see how this related to how likely some loan was to be bad.

We first looked at just the letter grade and the relative frequency of each in the two classes

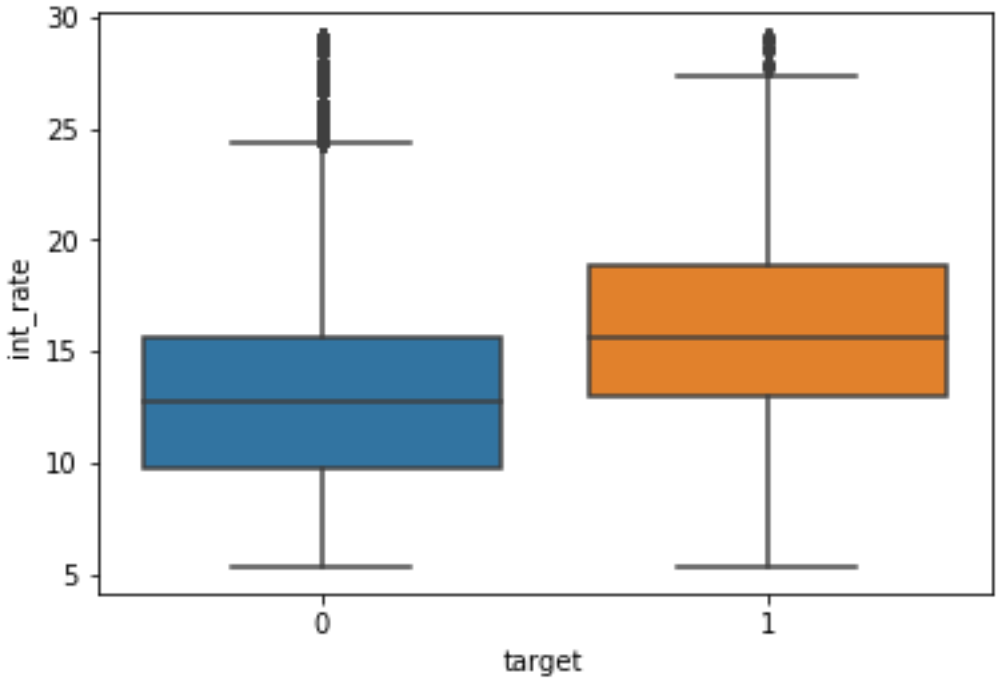


We can see here that higher grades were more frequent for the negative class than for the positive class, and lower grades were more frequent for the positive class than the negative class. This is to be expected, as it suggests that people who were late or defaulted skewed towards worse credit scores.

We then did this refining further using the subgrade and the story was very similar:

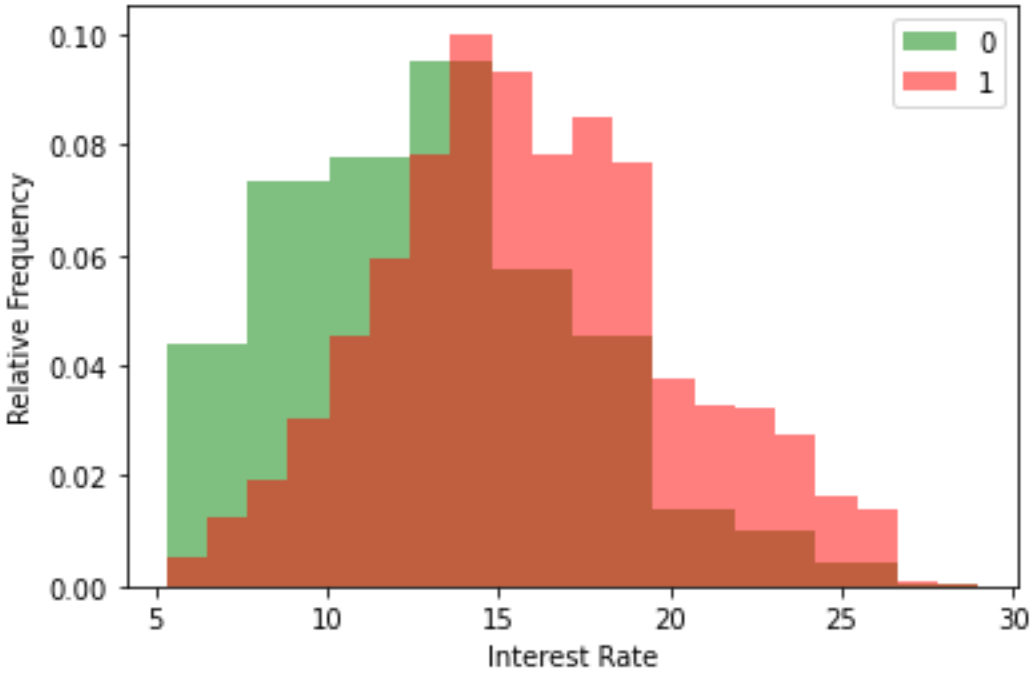


We then looked at a boxplot of the interest rates split by the good loans vs the bad loans.

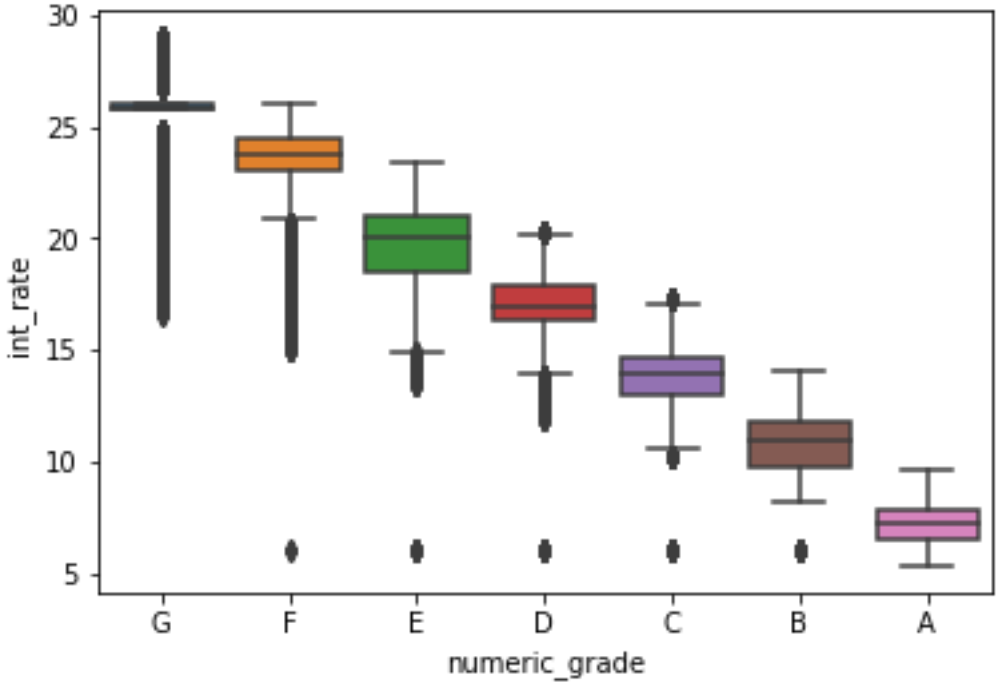


Interest rates definitely appear to be higher for people with bad loans as well. The most likely interpretation is that they were considered higher risk and thus given higher rates. We will look at this more carefully below.

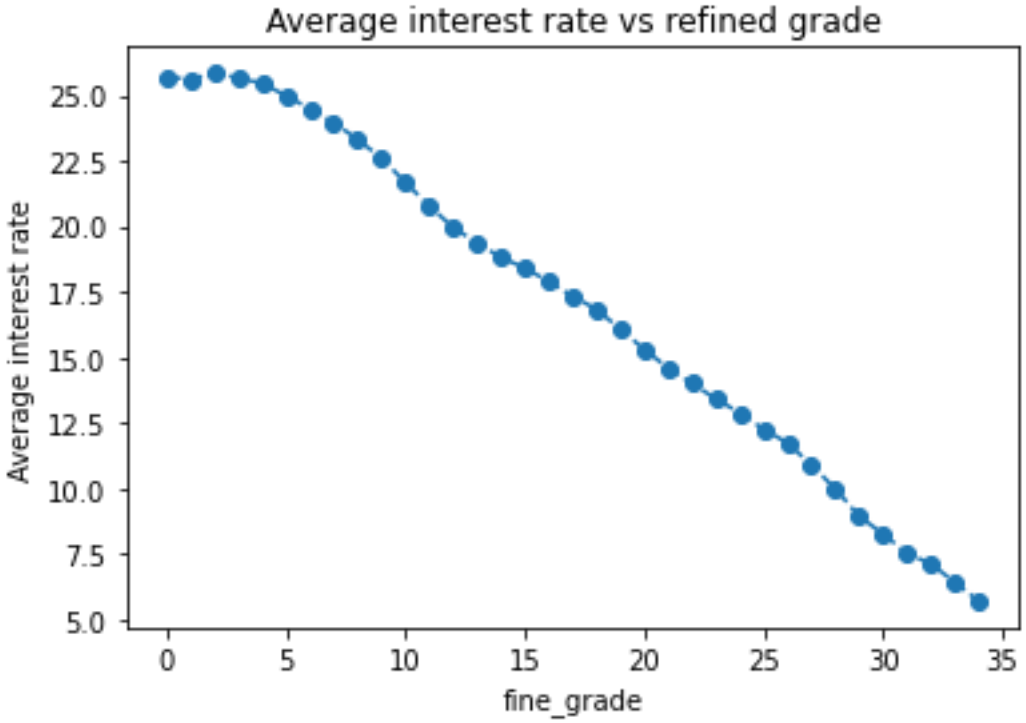
If we look at a histogram of interest rates split by classes we see the same sort of behavior we saw for the credit scores.

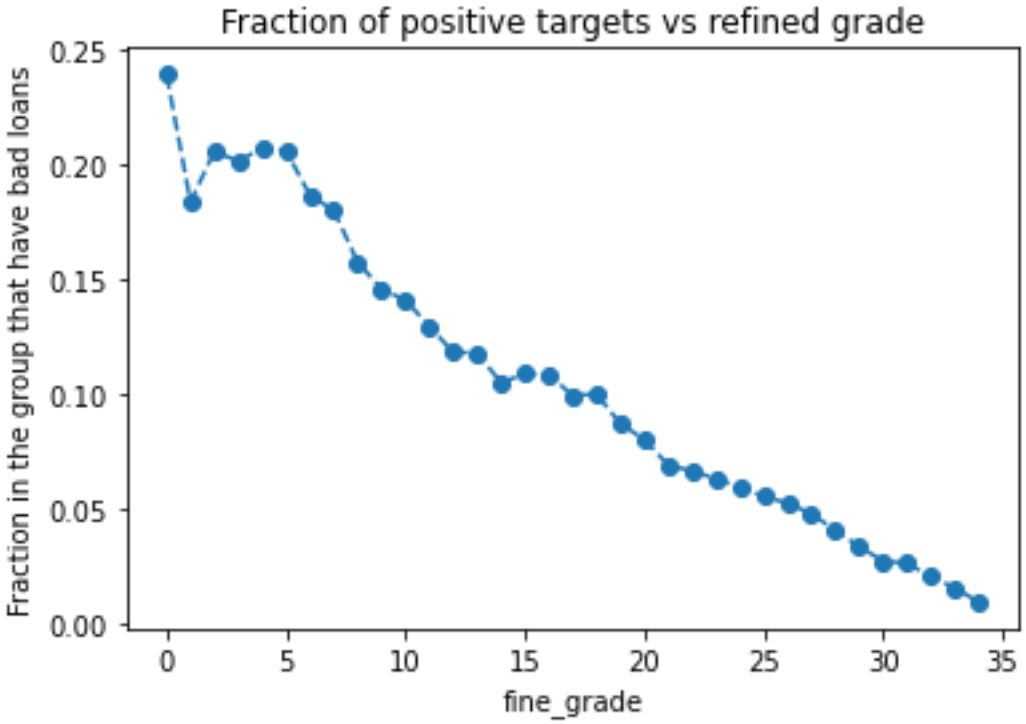


Both interest rates and credit grade/subgrade seemed to have a similar effect. In fact these were closely related as the following shows. First, we look at a box-plot of interest rates by credit grade:



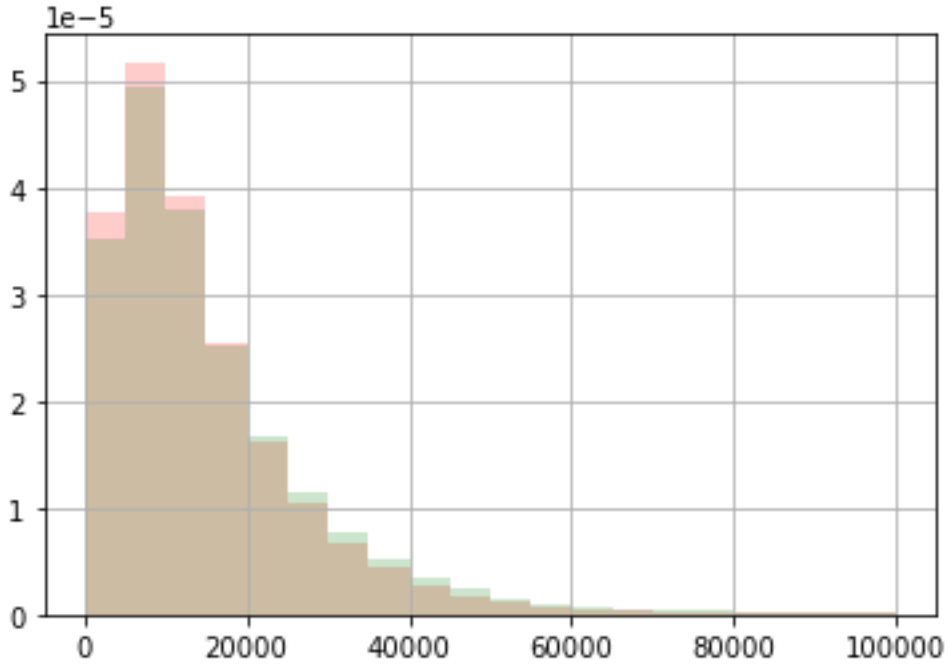
We can see as we move up in grade, the grouping of interest rates moves down. If we plot the refined grade vs average interest rate, we get a very obvious relationship.





We get a similar relationship between credit grade and risk of bad loan. This tells us that the credit score assigned by Lending Club does seem to correlate well to risk of being a bad loan with nearly a quarter of the ones with the worst grade being bad, going down to about 1% for the top grade.

We did look at some other possible features that might be related. One that seemed like it could be related was revolving debt balance, but when we looked at that, there didn't seem to be a significant difference in the distributions between the classes.



The red again corresponds to positive class, green to negative class. There is a slight difference, but it doesn't appear to be significant.

Some final preprocessing and modeling

Before we were ready to start running models, we still had one other feature that we had to deal with. The zip codes were given as three digits followed by "XX" , so "902XX" would be any zip code starting with 902. We had a few options of how to deal with it. At first, we didn't want to simply get rid of it, as zip code is more refined location information than just the state, but after looking more carefully we saw it was simply unreliable. There were zip codes listed with very different states. We chose to drop the zip code because of this and use the state as a categorical feature. Before we started to model, we took a few other categorical features and converted them to numeric ones. There were a few methods. One of the categories was length of loan, which we simply converted to an integer. Some other were binary categories so we just converted them to zero and one. Finally, any other category with multiple values was dummy encoded.

For actual modeling we had a couple of challenges that we might have to address. The first was that our data was imbalanced. In both the training and test data, only approximately 6.5% of the loans were bad. This had challenges in two ways. The first was that when building models, we needed a way to make sure the model was addressing this. The second was that we needed a good way to measure how well our model performed. Accuracy would not work well at all, as the "model" of simply declaring everyone to be good would have over 93% accuracy but would be a terrible model. When picking what model to use, we looked at a number of metrics, including the confusion matrix, precision, recall, and F-1 score. The other potential challenge was that our training and test data came from different time frames. We addressed this issue as far as what features to use and how to feature engineer so that the date information the model received wouldn't depend on when the loan was given, but it could well affect how well the model performed if there was data drift.

First Models

Our baseline model was a Random Forest Classifier. We used a grid search to look for the best hyper parameters, and we addressed the imbalanced data in a couple of ways here. The first was the scoring we used was "balanced accuracy". This is the average of the true positive rate and the true negative rate, which avoids the problem that accuracy has with imbalanced data. The other was that we used balanced class weight, which meant that when decision trees would measure how they did, the positive classes had a much higher weight than the negative class. Both measures would help alleviate the problems from the unbalanced data.

On the test data, we got saw that this had an F1-score of 0.77. The biggest problem was that the recall was relatively low at 0.73. Out of 52587 members of the positive class, 15743 of them were classified as in the negative class. We looked at difference scoring methods for the grid search including ROC-AUC and average precision. The results varied slightly , but were all in this ballpark, and we didn't see any one method really stand out above the others. We then looked at using LightGBM as a model, but overall as it was, it performed worse than the Random Forest classifier.

The next step was to try a neural network. The main downside to this was a matter of interpretability. With the other methods, we could see what features were important, and this could be useful in refining the loan application process. The neural network did do a pretty good job of not giving false positives with only 867 false positives out of 706751 members of the negative class, but still had a problem with false negatives, classifying 15462 of the positive class as negative. Changing the threshold of probabilities didn't change this much as setting the threshold as low as 0.05 still didn't improve things much.

Attempts to improve the models

The next step we took was to see if we could deal with the class imbalance by oversampling. We use a random oversample to create a training set where 50% of the entries were of the positive class. This had a worse performance on a random forest classifier than we had before with even more false positives and false negatives. This indicated that this route was not going to work out best.

The next thing was to try to introduce some new features created out of the old one. Generally, we looked for features that might be relevant. So, we did a few things. One was to compare the difference between the amount of money requested and the amount of money received in the loan. We also looked at some ratios. These included the ratio of loan received to loan funded, ratio of existing debt to the loan itself, proportion of debt still outstanding, ratio of payments to principal, and a ratio of how many of the delinquencies in the past two years were still current. These all seemed like they might have some predictive value and were based on existing features.

On the random forest model, the performance was actually worse on the test data, and on the lightGBM model we had similar performance to before. To potentially improve the result we wanted to see what features were actually important, so we looked at feature importances from both the random forest model and from the light GBM model. The results are shown as follows.

For Random Forest:

	feature	importance	cumulative importance
28	collection_recovery_fee	0.126478	0.126478
136	amount_left_ratio	0.110099	0.236577
137	ratio_of_payments_to_principal	0.102296	0.338873
20	out_prncp	0.100921	0.439794
21	out_prncp_inv	0.092089	0.531883
40	next_pymnt_d	0.066905	0.598788
29	last_pymnt_amnt	0.063439	0.662228
37	last_credit_pull_d	0.049959	0.712186
27	recoveries	0.046212	0.758398
4	int_rate	0.044219	0.802618
24	total_rec_prncp	0.040548	0.843165
25	total_rec_int	0.027786	0.870951
142	ratio_of_int_pay	0.026242	0.897193
19	initial_list_status	0.025608	0.922801
23	total_pymnt_inv	0.016693	0.939494

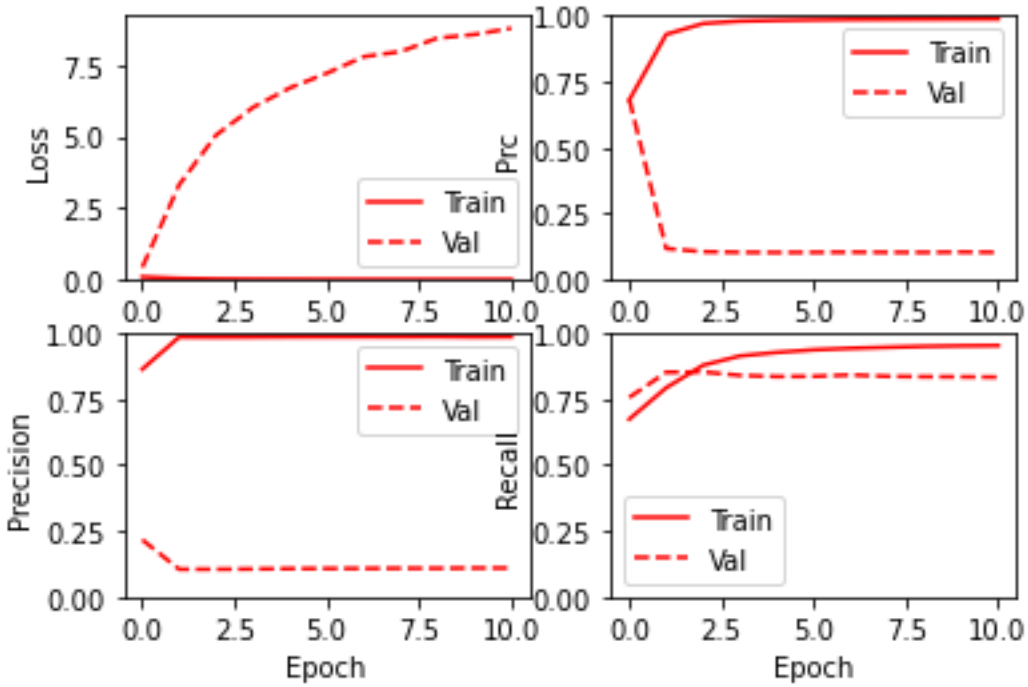
For LightGBM:

	features	importance	cumulative_importance
137	ratio_of_payments_to_principal	187	0.311667
20	out_prncp	123	0.516667
29	last_pymnt_amnt	78	0.646667
27	recoveries	77	0.775
40	next_pymnt_d	76	0.901667
26	total_rec_late_fee	44	0.975
4	int_rate	15	1.0

The list doesn't need to go past here because LightGBM had zero importance for the rest of the features. While the two models don't have all the same features as important, some notable common features were ratio of payments to principal, outstanding principal, recoveries, and interest rate. In order to have models that might avoid "noise" that were features of low importance, we looked to see what would happen if we took the features that accounted for about 90% of the importance for the random forest model and the one additional feature that had non-zero importance for LightGBM. This wouldn't chance the LightGBM model as it already reduced to features in that list, but we could look at how some other models did using that restricted set of features.

When we applied a random forest model to the reduced set of features, the performance degraded. We had a significantly larger number of false positives and the number of false negatives did not improve. This was not promising for this reduced data set but we looked to see how a neural network might perform on the reduced feature space.

We tried a few neural networks but especially on this reduced feature space, what we found was that we got very good performance on the training data, but extremely poor performance on the test data. This graph shows the loss function, area under precision-recall curve, precision and recall for training one of these neural networks on the reduced data set. The solid lines are for the training data, the dashed ones for the test data.

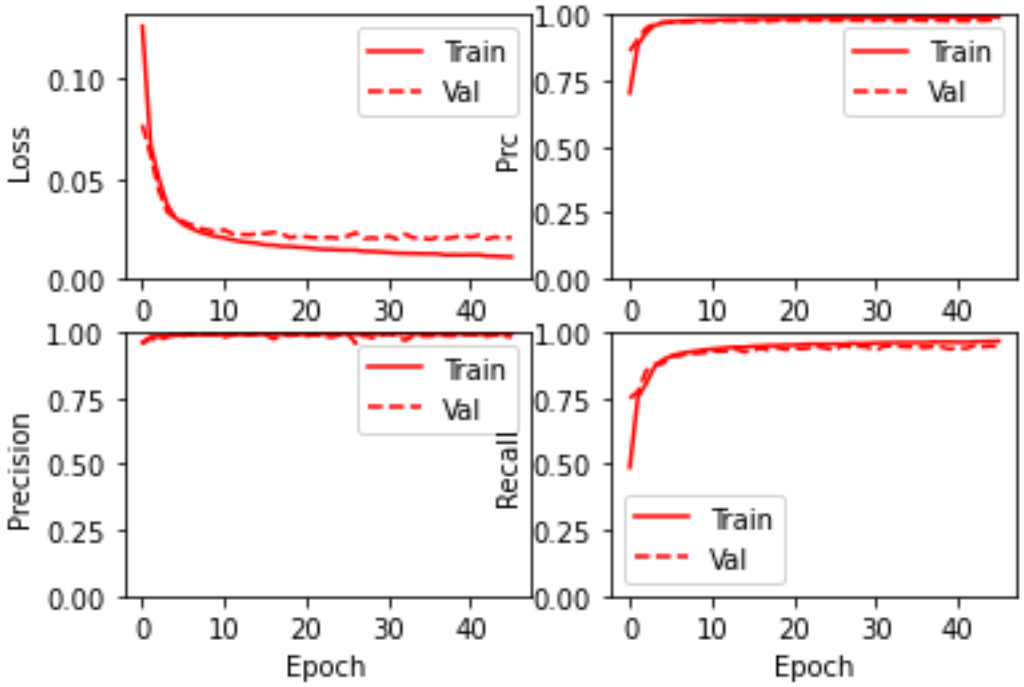


We can see that the neural network was able to get excellent performance on the training data here, but this did not translate to good performance on the test data.

Issues and challenges

As we noticed in particular for that neural network shown above, we could get models that had excellent performance on the training data, and poor performance on the test data. The first issue might be that we are experiencing overfitting, but that does not seem to be the case. We were able to see what happened if we only looked at the training data and held out part of it to be the test data. This could help spot overfitting, because if the models are still having problems on the data that was held out, then that would show that we have overfitting models. That did not occur, however. We saw that if we randomly held out a part of the training data to use as test data, and trained that way, the models had much better performance on the data that was held out. We can see this by looking at one example of a neural network. For this neural network we randomly held out 30% of the

training data set and trained on the remaining 70%. The graphs for loss, area under precision-recall curve, precision, and recall look like this:



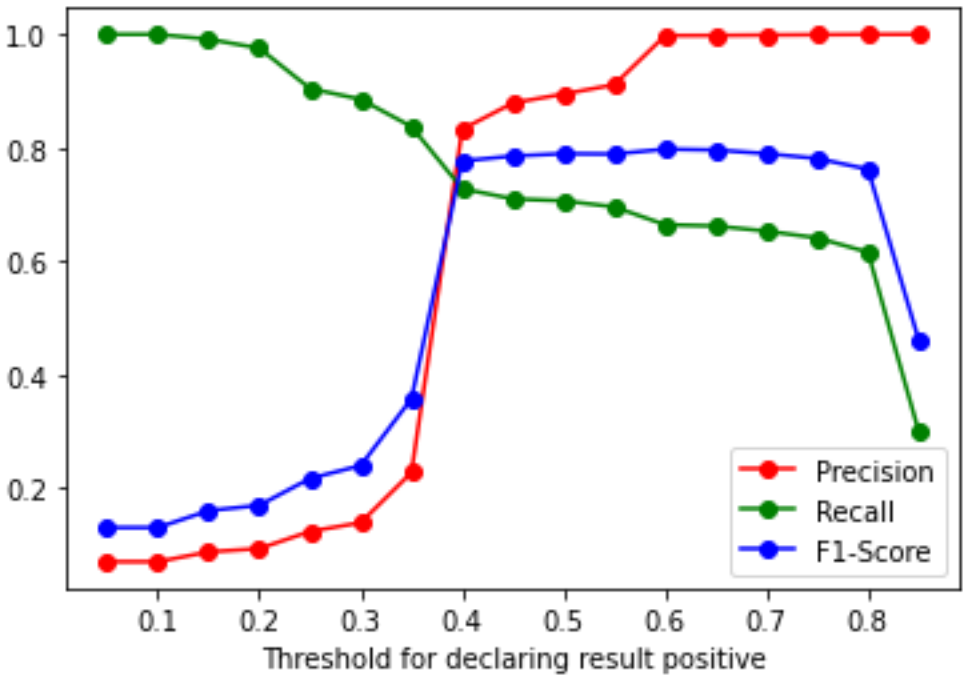
Unlike the previous case, we see excellent results on the training and validation set here. However, when this model was used with the actual test data, the performance was quite poor.

The fact that we could get very good results on held out portions of the training data set but never performance nearly as good on the separate test set suggests that there has been some drift in the data. The training and test sets cover loans from different periods of time, so it's possible the distribution of the features or the relationship between the feature values and target have changed.

For future work, this suggests we need to see how features and/or their relationship between whether the loan is being paid back are changing over time. For now, the big issue that every model ran into was a fairly large number (usually about 30%) of the positive class being identified as negative. This appears to be the result of drift in the data, and if that is the case, no model based on this set of data can really do better.

Final Result and Summary

In the end what looked like the best model we could get was the LightGBM model using our extended set of features including the ones we engineered. We actually can get a bit of an improvement on the standard "predict" method from this by looking at the probabilities generated. We tested various cutoff points from probability of 0.05 of being in the positive class up to a 0.90 probability. We could see that the precision, recall, and F1-score varied by this:



We wanted to try to optimize our result by F1-score here and we can see that 0.6 was the best cutoff to use:

	Threshhold	Confusion Matrix	Precision	Recall	F1-Score
11	0.6	[[706676, 75], [17672, 34915]]	0.997857	0.663947	0.797355
12	0.65	[[706684, 67], [17787, 34800]]	0.998078	0.661761	0.795847
13	0.7	[[706708, 43], [18265, 34322]]	0.998749	0.652671	0.789447
9	0.5	[[702376, 4375], [15447, 37140]]	0.894616	0.706258	0.789356
10	0.55	[[703182, 3569], [16018, 36569]]	0.911082	0.6954	0.788762
8	0.45	[[701610, 5141], [15272, 37315]]	0.87891	0.709586	0.785224
14	0.75	[[706729, 22], [18891, 33696]]	0.999348	0.640767	0.780859
7	0.4	[[699048, 7703], [14328, 38259]]	0.832405	0.727537	0.776446
15	0.8	[[706746, 5], [20223, 32364]]	0.999846	0.615437	0.7619
16	0.85	[[706751, 0], [36852, 15735]]	1.0	0.299218	0.460613

The main concern here is that optimize this we made even more false negatives than we had before, but overall this result worked the best for a single cutoff.

A slightly better approach might be to have two cutoffs. As we can see from the above situation, using a cutoff of 0.6 would be very good at preventing false positives. We can look to see what a good cutoff on the other end might be by looking at these sorted by recall.

	Threshhold	Confusion Matrix	Precision	Recall	F1-Score
0	0.05	[[0, 706751], [0, 52587]]	0.069254	1.0	0.129537
1	0.1	[[0, 706751], [0, 52587]]	0.069254	1.0	0.129537
2	0.15	[[157013, 549738], [452, 52135]]	0.086621	0.991405	0.159322
3	0.2	[[202621, 504130], [1285, 51302]]	0.092364	0.975564	0.168751
4	0.25	[[366809, 339942], [5027, 47560]]	0.122735	0.904406	0.216138
5	0.3	[[416285, 290466], [6030, 46557]]	0.138142	0.885333	0.238993
6	0.35	[[556610, 150141], [8559, 44028]]	0.226751	0.837241	0.356855
7	0.4	[[699048, 7703], [14328, 38259]]	0.832405	0.727537	0.776446
8	0.45	[[701610, 5141], [15272, 37315]]	0.87891	0.709586	0.785224
9	0.5	[[702376, 4375], [15447, 37140]]	0.894616	0.706258	0.789356

Too low a threshold here simply classifies everyone as positive, which is not useful, but if we look around the values of 0.15 and 0.2, we still have very low false positives. So one possible approach to using this sort of model for loans, would be to pick thresholds p_1 and p_2 with anyone having a predicted probability below p_1 would be approved immediately, and anyone having one above p_2 be rejected immediately, and anyone who is in between could be reviewed more carefully by a specialist.

One other thing that might be useful is to use that probability of being in the positive class to help set the interest rate. However, in order for that to work, we would need to remove interest rate from the predictive features. This looks problematic on the surface as interest rate was one of the features we kept when we narrowed down our feature space. However, as we saw that refined grade based on grade and subgrade strongly correlated to interest rate, so hopefully models trained that way would use the that refined grade instead of interest.

To sum up what we were able to do, a LightGBM model that used some additional features engineered from old ones turned out to be the best approach, especially if we modified the cutoff probability from 0.5 to 0.6. We can see certain features really seem to be strongly relevant. The drift of the data over time is something that should be considered, as well as the possibility of using multiple cutoffs to help avoid either giving a loan to someone that is likely to have problems paying it back or to refuse a loan to someone who is actually likely to be able to pay it back.