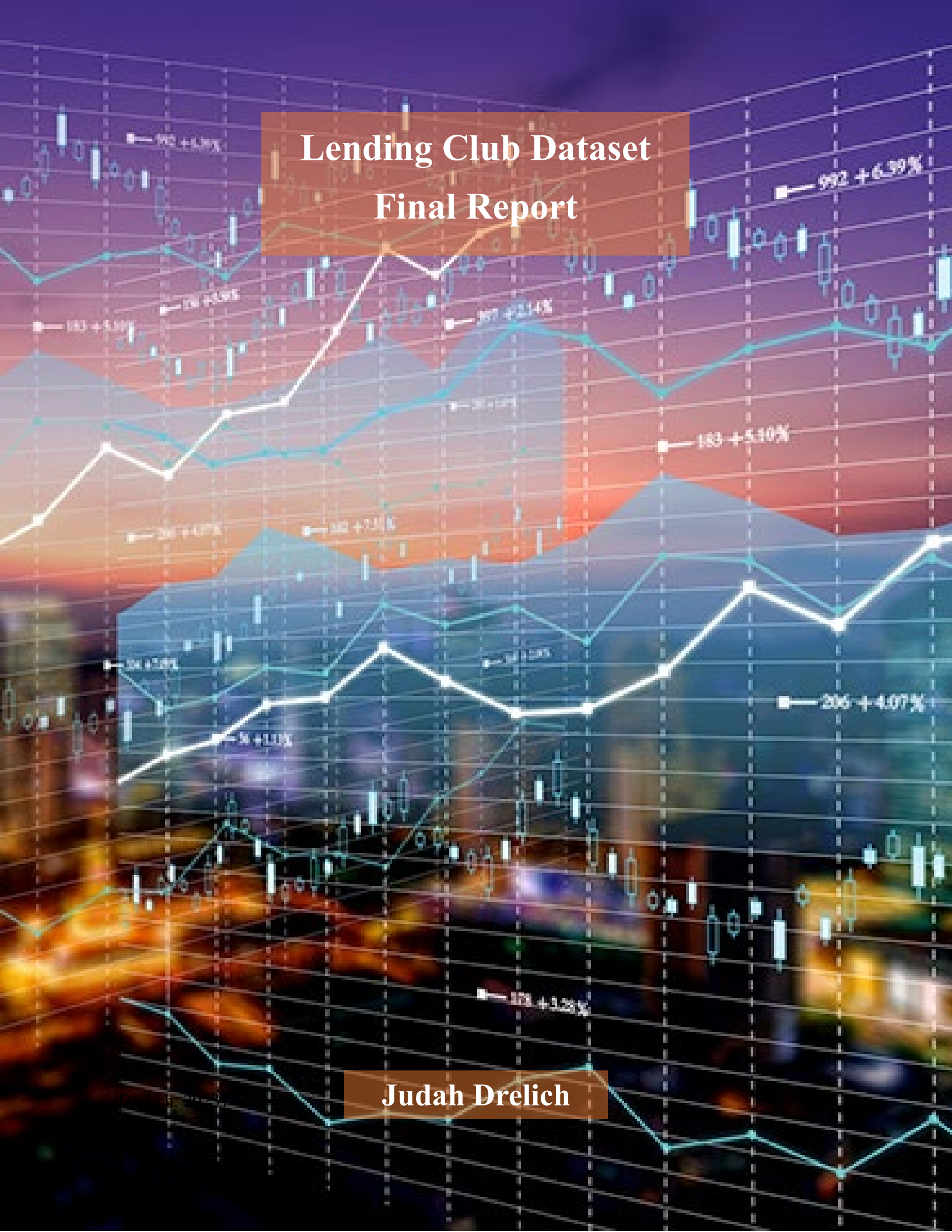
Lending Club Dataset
Final Report



Judah Drelich

Abstract

Since 2000 B.C.E. loans have been an integral part of society and economics. Lending Club is an online platform that facilitates lending between individual or institutional investors and borrowers who need short term liquidity. Lending Club publishes data on the borrowers and their loans (both current and finished) on Kaggle.com. While the dataset did not have enough features before the time of origination to allow me to conduct an audit of how good each loan is, I was able to create a derivative market for active loans based on the results of the finished loans. While I do not have a definitive answer on the precision and accuracy of the market, I used LIME, a machine learning explainer, to interpret the valuation of each loan. The best model that I used, Random Forest, averaged a 13% error rate on average, not accounting for macroeconomic factors or a potential skew in the data. With additional datasets and updated models these results could be vastly improved.

Introduction

Debt is the backbone of all economies. It allows for non-simultaneous transactions to occur by binding the debtor to the creditor for the life of the loan. Unfortunately, many Americans have entered into debt without having a straightforward way out. By Q2 2023, total credit card debt was $1.03 trillion and student loan debts were $1.57 trillion.[1] One way people overcome debt is to take on loans that give an infusion of cash so that they can stave off their creditors. This has become increasingly common in the U.S. as "22.7 million Americans owe a collective $232 billion in personal loans, more than double the $117 billion owed in 2017."[2]

[1] (Federal Reserve Bank of New York, 2023)
[2] (Schulz, 2023)

Lending Club is an online brokerage that matches borrowers to investors or institutions to facilitate loans at non-predatory interest rates. For the borrowers, Lending Club offers a chance to either relieve debt or pay for an expense that they could not otherwise afford. For investors, Lending Club provides an opportunity to make a profit while helping someone else. Lending Club offers some of their most stable loans to financial institutions; however, private investors hold most of the loans within Lending Club's portfolio. This dataset, curated and finalized by Lending Club in early February 2016, encompasses comprehensive data on more than 880,000 loans and borrowers, both active and finished.

Target Variable

My first step was to wrangle the data and separate the features (X) from the target variable (y). When I initially started this project, I had believed that `*loan_status*` was the target variable. My goal was to find the best deals on potential loans and then audit the grade that Lending Club had given to these loans. Since `*loan_status*` gave a clear description of the state of the loan and the dataset's description on Kaggle agreed, it was the best indicator of how well the loan would perform.

As I looked closer at the data, however, I realized that it was hard to predict `*loan_status*` just of the features in the dataset. Too many of the variables contained information that came after the loans originated, which muddied whether the variable is a part of the features or the target variable. An example of this is `*next_pymnt_d*` which is the variable that represents the date that the next loan payment is due. If I wanted to create a model that evaluated potential loans on behalf of investors before origination, it would be impossible to know if two years into the loan, the payments were still occurring or if the borrower had completely defaulted.

To use all the features of the dataset, I need to shift the timeframe of the questions that I had been asking to make it concurrent with the entire dataset. Instead of assessing the loans before origination, I looked to find the value of the remaining loan as of the time of the dataset. That way `*next_pymnt_d*` becomes a variable that will have happened and remains viable in future models. With the current values of the loans, I can create a market for active loans that will be helpful for the loan investors or institutions who are thinking of selling off their loans for immediate liquidity.

The new target variable needs to apply to all loans, regardless of size, so that the machine learning models can train and predict the entire dataset. The simplest way to do this is to use a ratio of the amount paid back compared to the full amount owed. Therefore, I separated out every column of the dataset that had information on the amount that the borrower has already paid and the amount that they owe. With those columns I created the ratio; the remaining columns became the features.

Data Wrangling

A fully wrangled dataset must have only numerical values. In this dataset, some columns contain numbers such as `*term*` (length of the loan) which had values such as " 36 months". The numerical information is there, but I needed to delete text and convert the value from a string into a number. Other variables are purely categorical such as `*emp_title*`, which gave the profession of the borrower. This needed to either be one-hot encoded or dropped if there were too many unique values.

One of the best tools at the start of any data science project is `*ProfileReport*` from `*Y-data Profiling*`. While this dataset was too large for my computer to use `*ProfileReport*`, the structure of the report is a good guideline on how to wrangle the data. Some of the things that `*ProfileReport*` checks for are:

- **Constant features**

Constant values are easy to fix because they add no relational information to the dataset, and I can consolidate them into a single cell.

- **Missing values:**

The imputation of values for this dataset is one of the most challenging aspects of this project and data science as a whole. There are many columns with large areas of missing values. These reasonable fill in values allow the algorithms to capture the existing patterns in the data instead of artificial ones coming from synthetic data.

My assumption was that most missing values were zeroes those borrowers never recorded. For instance, `open_il_12m` refers to the number of opened installment accounts in the

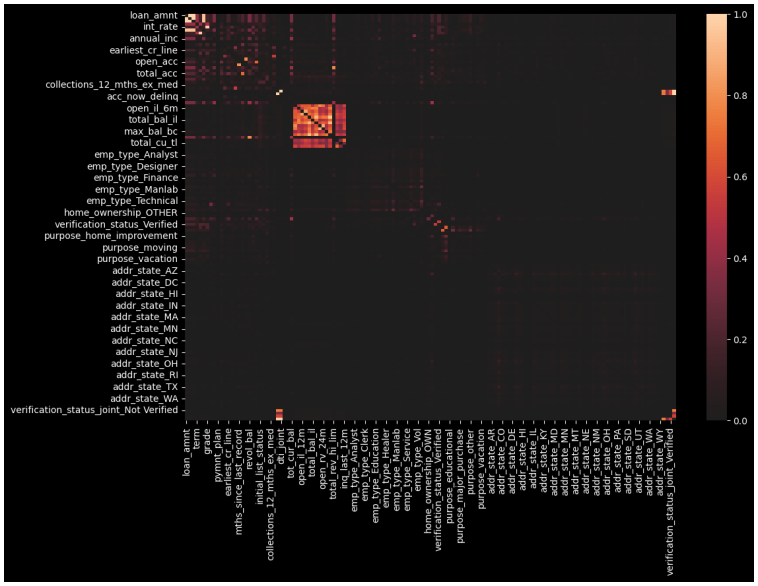
last 12 months. If there isn't any information on the number of installment accounts, I assumed there likely aren't any installment accounts and the person left that part blank on the application.

Values such as `mths_since_last_delinq` or months since last delinquent are different because, if someone were never delinquent, then “zero” is the exact wrong answer because it would imply that the borrower is currently delinquent. The higher the value here, the better the likely outcome on their loan. I use in this case the value of 1000 for missing `mths_since_last_delinq` values because I believe 83 years’ worth of non-delinquency is the equivalent of never being delinquent in the model’s eyes.

- **Multicollinearity**

Multicollinearity

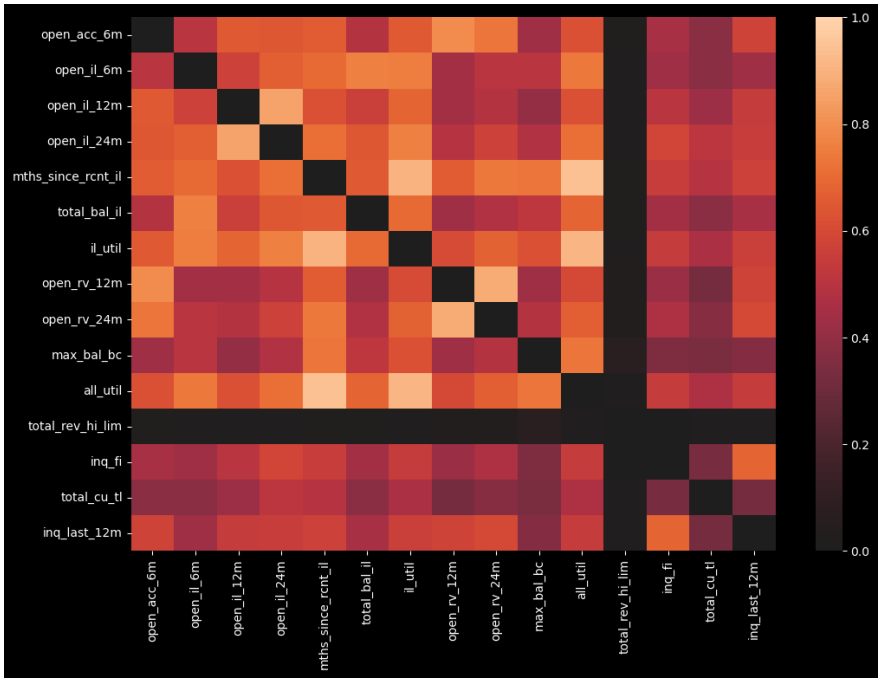
Multicollinearity is more than two features varying together. This should be avoided because it would become hard to assign correlation or causation to any given feature without a clear mathematical distinction between either of the features. Below is the starting point of the multicollinearity for the entire dataset:



While it may look like most of the data does not have a multicollinearity problem, the dark part of the graph holds all the dummy variables from the categorical columns. Because I dropped the first value from each of the categorical columns, it is mathematically impossible for those dummy columns to be collinear.

The heatmap above shows a concerning square from features 27 to 42. Cleaning up the collinearity in that square is not easy. While it may seem tempting to simply drop all the problematic columns, there is too much information in those columns to discard them.

My first step was to zoom in on bright spot of the heatmap to see if there were any groupings of features that had clear real-world explanations for their high correlations. This produced the following graph:[3]



Many of the features in this graph have tremendous overlap. For instance, `*open_il_6m*`, `*open_il_12m*`, `*open_il_24m*` are measuring the number of installment accounts in the last 6, 12, and 24 months that the borrower has opened. Instead of having to pick just one of these

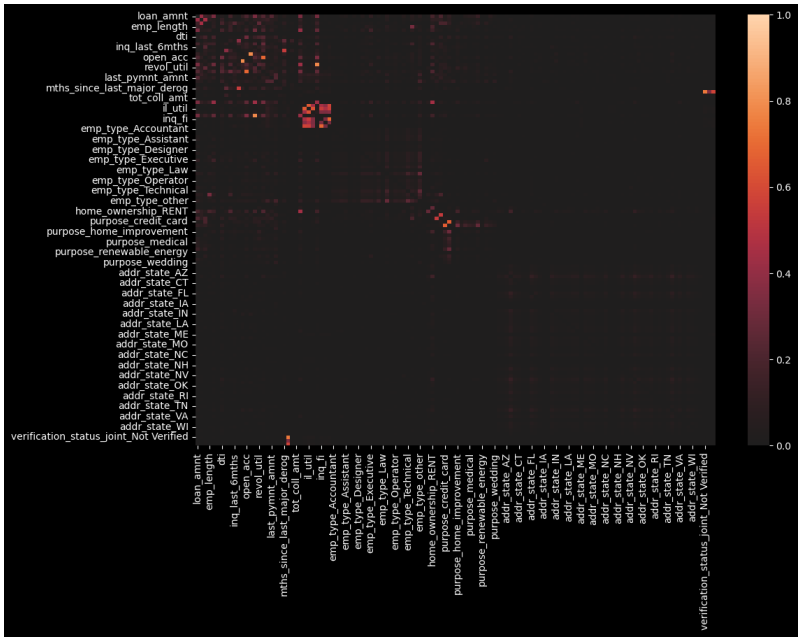
[3] (Drelich, 2023)

variables, I combined them all into an `*open_il*` variable by scaling the shorter and longer versions to match with the yearlong iteration and then averaged the three.

After combining as many variables as I could, I wrote code to find all the features that had correlations with each other above a given threshold. The criteria that I had for dropping features were the following:

- Features that have default values of 0 are better than features with undefined values.
- Dollar amounts are better than number of accounts. They give more information and with proper scaling are more useful.
- The more information the variable contains, the better.
- Synthesized features from multiple variables are better than original features.

After dropping enough collinear features the heatmap looks like this



Variance Inflation Factor

To make sure that I had fully ended the multicollinearity I calculated every remaining feature's Variance Inflation Factor (VIF). The formula for VIF is:

$$VIF = \frac{1}{1-R^2} \quad (1)$$

Where R^2 is the R-squared value that represents correlations.[4] When R^2 approaches zero, the VIF approaches one, whereas when R^2 approaches 1, the VIF approaches infinity. The guidelines for an acceptable VIF score can range from 2.5 to 10 although anything over five is suspicious.[5] With the remaining features that I had, 119 out of the 120 had a VIF of under 2.5 and one had a VIF ≈ 2.73.

Lasso Regularization

"Lasso Regularization is a technique for variable selection that uses regression to evaluate the effect that features have on a target variable. The idea is to add a penalty term to the regression loss function that has the coefficient or slope of the independent variable with respect to the target variable, multiplied by a parameter α.

$$Loss(\beta) = SSD + \alpha|\beta| \quad (2)$$

"Where β is the coefficient of the feature, *SSD* is the sum of squared distances of the observations to the regression line and α is the penalty's parameter. If there are many features, then we generalize equation (2) by summing the coefficients:

$$Loss(\beta_1, \dots \beta_n) = SSD + \alpha \sum_{i=1}^{n} |\beta_i| \quad (3)$$

[4] (Menor, 2022)
[5] (Frost, 2020)

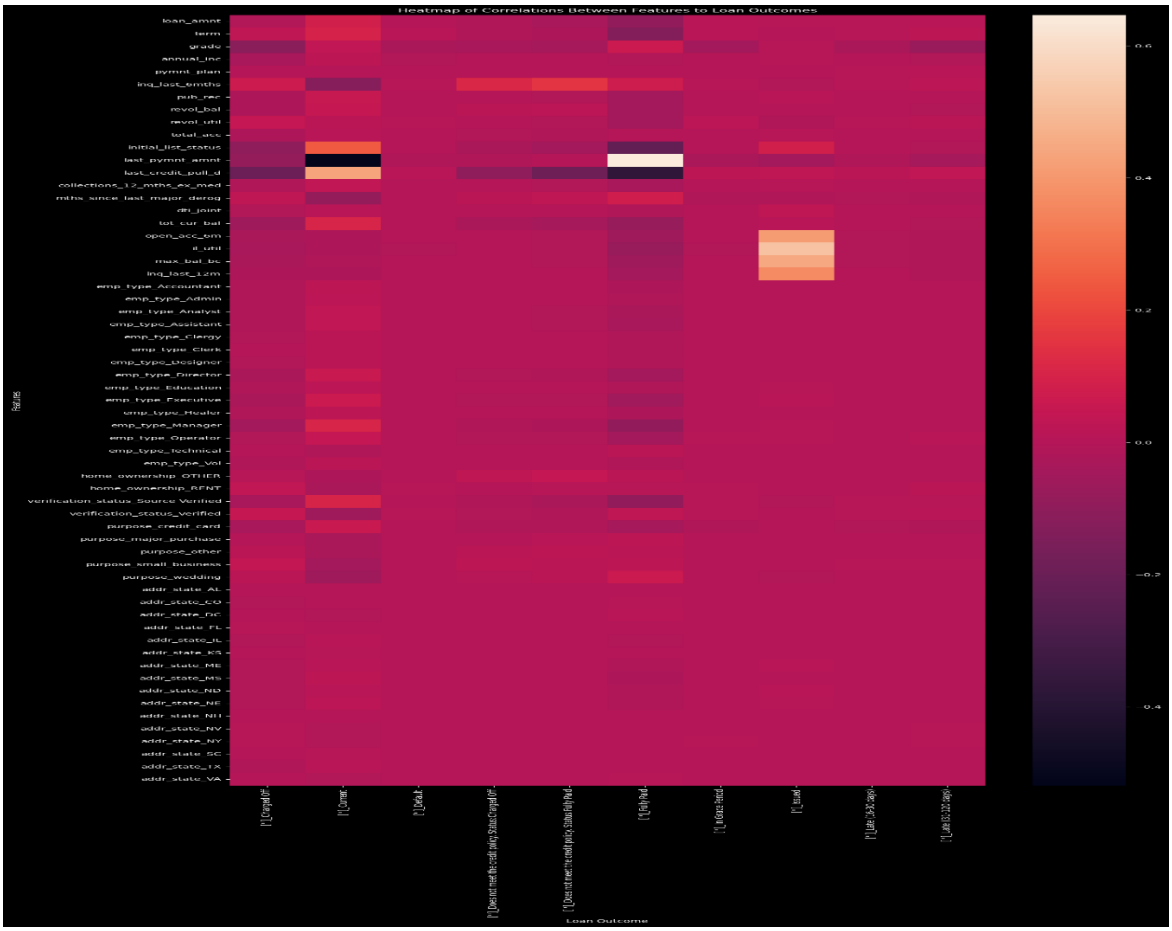
"Lasso aims to reduce the loss in equation (3). If a feature is not important, then changing its slope will not move the regression line close enough to the data points to decrease the loss function with a non-zero value. If a feature is important, the regression line will move towards the data and minimize the SSD faster than it increases the penalty term. Since only features with non-zero coefficients are meaningful, I can discard all the features with a coefficient of zero."[6]

When I use Lasso on the 120 features, only 61 have an impact on the outcome of a loan. I also used VIF one more time since before there was a variable that was above 2.5. While Lasso was not able to bring it under 2.5 it did get it to less than 2.504 which is close enough not to make a difference.

[6] (Drelich, 2023)

Exploratory Data Analysis (EDA)

With the dataset fully cleaned and prepped, the simplest thing to explore is the correlation between the features and the target variable. While it is true that `*loan_status*` is no longer the official target variable, it is a good proxy for the outcome of the loan and is easy to analyze.

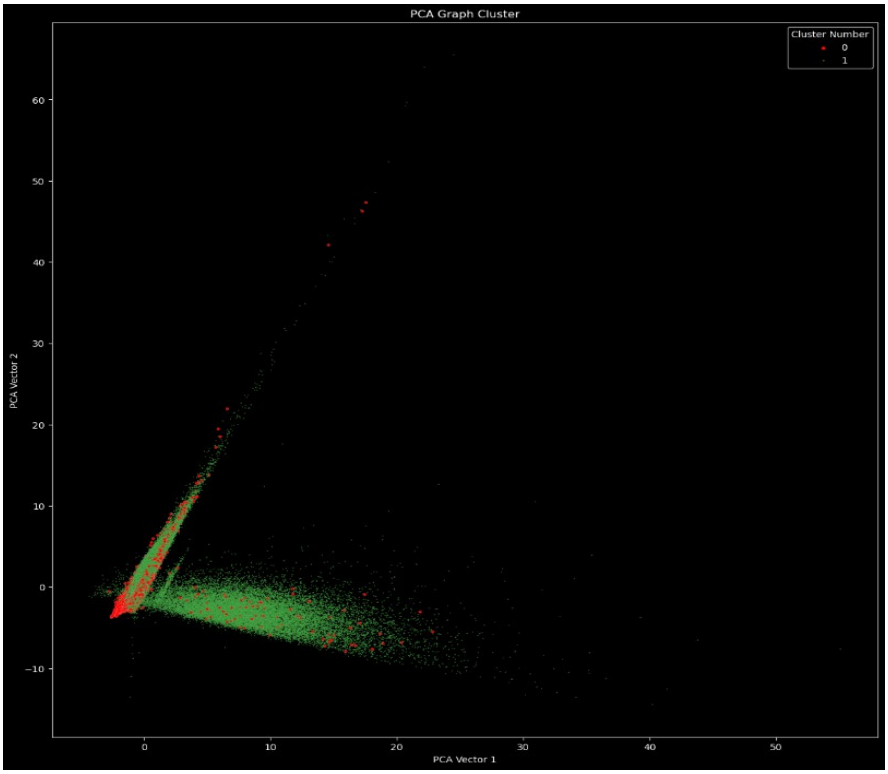


The graph above is a heatmap of the correlation between the features and each potential outcome of the loan. Most of the columns have correlations near zero because they represent infrequent loan outcomes. Overall, there are only a few features that have an impact on `*loan_status*`. The feature with the highest correlation is `*last_pyment_amnt*` to *Fully Paid.* This makes sense, as fully paid loans will have consistently higher payment amounts than non-paid loans.

Clustering

One way to recognize deeper patterns is to look at spatial clusters created by the structure of the data. Instead of looking at singular features that would pop in correlation metrics, multiple features can be combined to sort the data. Given the different outcomes in the `*loan_status*` column, it would seem evident that there are three groups of borrowers: Those that fully pay on time, those that are late but pay in full (including late fees), and those that do not pay in full.

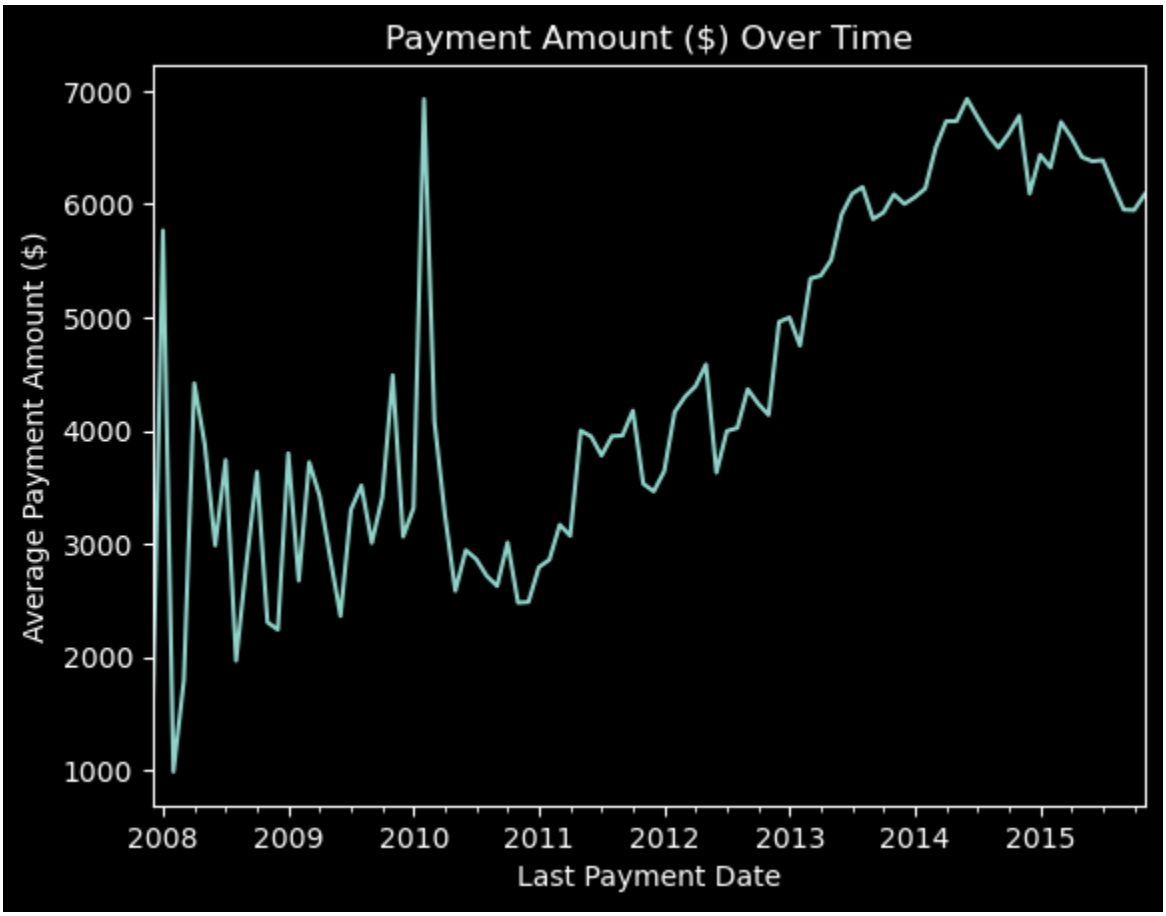
Before creating these clusters, I visualized the dataset with Principal Component Analysis (PCA). PCA reduces the number of variables by selecting the features with the highest variances and combining them into a few vectors. This allows a person to visualize high dimensional data to see if there are visual patterns that are not obvious from the numbers themselves. While it is true that PCA can discard a low variance variable that carries vital information, PCA is still a good starting point.



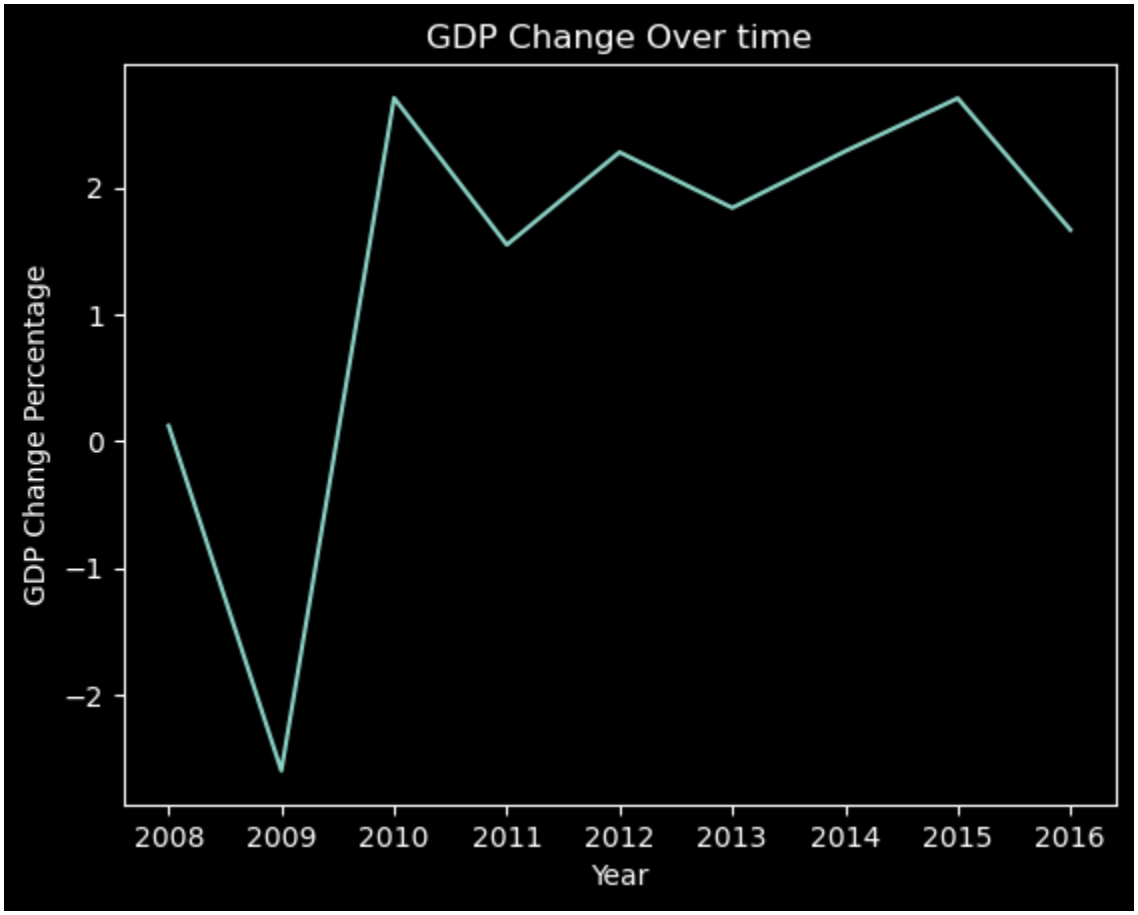
The graph above shows that while the bad loan outcomes (in red) congregate around the left perpendicular vertex, the points are too dispersed throughout the structure to justify the use of clustering.

External Factors

Any financial system in use over many years that involves people's critical financial resources is going to be subject to macro-economic forces. In particular, the state of the economy can impact the level of liquidity that borrowers have left to pay off their existing loans. To investigate this, I have graphed the borrowers' average last payment amounts per month.



I was curious how the graph above compares to macroeconomic data from the same period. So, I accessed the World Bank's data on U.S. GDP over time[7] and plotted it below.



This graph is eerily similar to the previous graph. Economic winds will influence not only borrowers' available capital to pay off their loans, but also their willingness to part with that capital. Since macroeconomic data is impossible to forecast, all the models will have errors that will never disappear.

[7] (Bank, 2022)

Preprocessing

Only a subset of the dataset contains completed loans. While my goal is to develop a market for active loans, without clear structures in the data I need to use supervised learning models to predict loan outcomes. To distinguish between completed and ongoing loans, I used the `*loan_status*` column which, in addition to providing the financial status of the loan, also indicates whether the loan has finished.

The first step in creating a market for the loans is to find the expected value of each loan. Lending Club does not use compound interest for its loans, so the formula is:

$$Expected\ Payment = Principal * (1 + Interest\ Rate) * Years \quad (4)$$

Where $Interest\ Rate$ is in decimal form. I then calculated the amount that the borrower has paid back to the lenders:

$$Actual\ Payment = Total\ Payment + Total\ Late\ Fees + Recoveries \quad (5)$$

Where Total Payment is the amount that the borrower has paid back to the lenders, Total Late Fees are the late fees that the borrower incurred if late on a payment, and Recoveries are monies that Lending Club was able to recover after charging off the loan.

With Expected Payment and Actual Payment calculated, I can now create `*frac*` which is the ratio of the amount of money that the borrower actually paid back to the lenders compared to the amount that the borrower was expected to pay back to the lenders:

$$frac = \frac{Actual\ Payment}{Expected\ Payment} \quad (6)$$

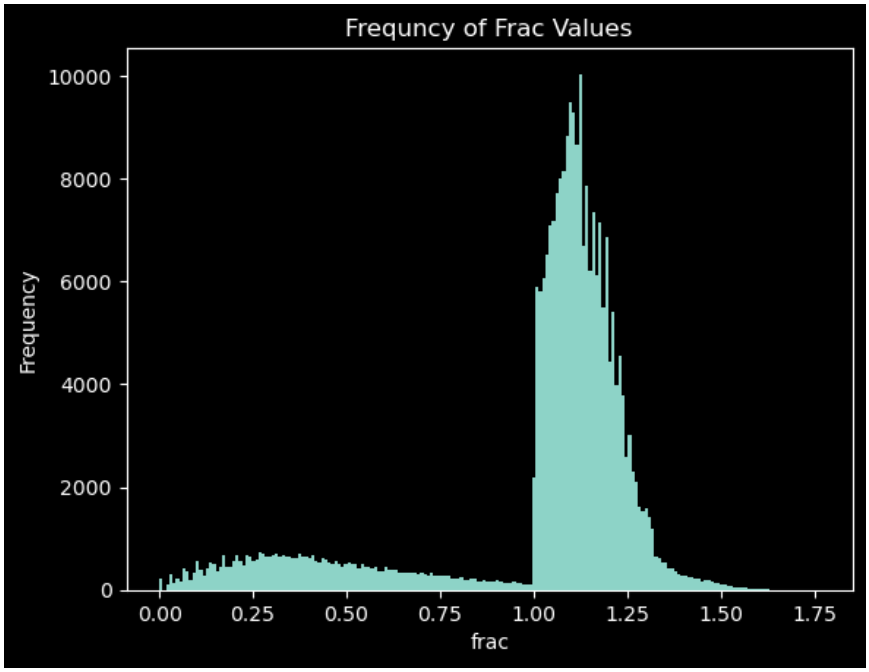
The active loans are slightly different because the expected payment applies only to the remaining portion of the loan. This is how I calculated it:

$$Expected\ Payment\ =\ Principal\ *\ (1 + Interest\ Rate)\ *\ Years\ -\ Actual\ Payment \quad (7)$$

Where `Actual Payment' is the same as the cell above. The idea is that the expected payment is the entire loan's expected value minus the money that the borrower has already paid.

EDA

Before I start the modelling sections, it is important to know a little bit more about the `*frac*` variable. I created this histogram of the distribution of `*frac*` to gain an idea of what the model is trying to predict:

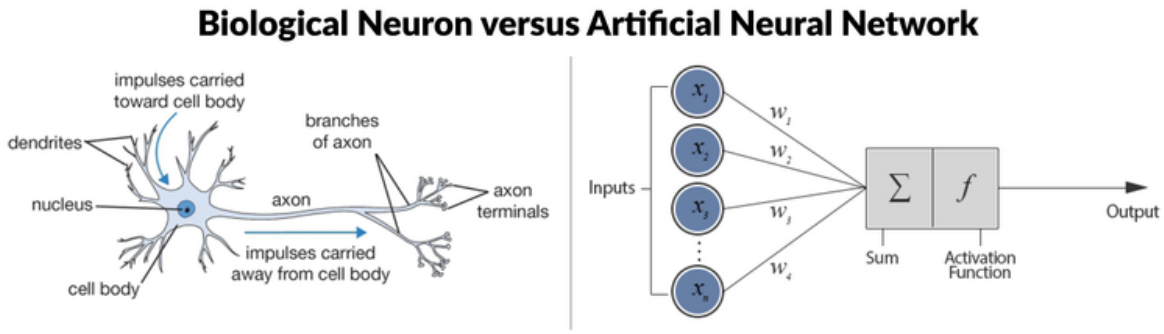


Below one, the values on the graph are skewed to the left. If the borrower does not pay off the loan, their credit rating will suffer. If a borrower is close enough to paying Lending Club back, the graph shows they will repay rather than damage their credit score.

Modelling

Neural Networks

"The term neural networks is inspired by the structure and function of biological neurons. The diagram below shows both a diagram of a neuron, and a node of a neural network.



In both, a node receives and processes many inputs, then produces an output. When many nodes are connected, they form a network that can emulate non-linear complex systems. The formula for a node in a neural network is:

$$Output = f(\sum_1^n w_i x_i) + b \tag{8}$$

Where f is the activation function of the node, w_i is the weight of a given input into the node, x_i is the value of a connected node, n is all the inputs into the node, and b is the bias of the node. The output of one node can become one of the inputs of another node or it can be the final node in the network that bears its prediction.

"When a neural network is training, the network updates its model weights to minimize the difference between its predictions and the target values in the training set. This is achieved by using the derivative of the function representation of the entire model. The network moves in the direction with the lowest derivative value to find the minima until there are no more directions with negative derivative values. When the model finishes, it can be used to predict a test set that it has not been trained on but has the same type of data as the model to make predictions."[8]

The smaller neural net that I created has an input layer of 61, with 12 hidden layers of varying numbers of nodes and a final output layer of a single node. The activation function that I used for every layer except the last was Relu. which stands for rectified linear unit. The Relu function is:

$$Relu(x) = \begin{cases} x \geq 0, & x \\ x < 0, & 0 \end{cases} \tag{9}$$

Where the idea is to discard the negative nodes that are not helpful while allowing the useful nodes to equally contribute to the model.

The final layer does not have an activation function and outputs the sum of all the inputs from the previous layer.

Metrics

The main metric that I'm using to measure the success of the model is mean absolute error (MAE), which is:

[8] (Drelich, 2023)

$$MAE = \frac{\sum_{i=1}^{n} |error_i|}{n} \tag{10}$$

Where $error_i$ represents the difference between the estimated `*frac*` and the actual frac for any loan *i,* and *n* is the number of loans in the data. I also used mean square error (MSE) and root mean square error (RMSE). RMSE highlights potential outliers in the dataset because higher squared values increase faster and taking the square root normalizes it back to the same power as MAE. Looking at the difference between MAE and RMSE can show how consistent a model is with the size of its errors. The formulas for MSE and RMSE are:

$$MSE = \frac{\sum_{i=1}^{n} error_i^2}{n} \tag{11}$$

$$RMSE = \sqrt[2]{\frac{\sum_{i=1}^{n} error_i^2}{n}} \tag{12}$$

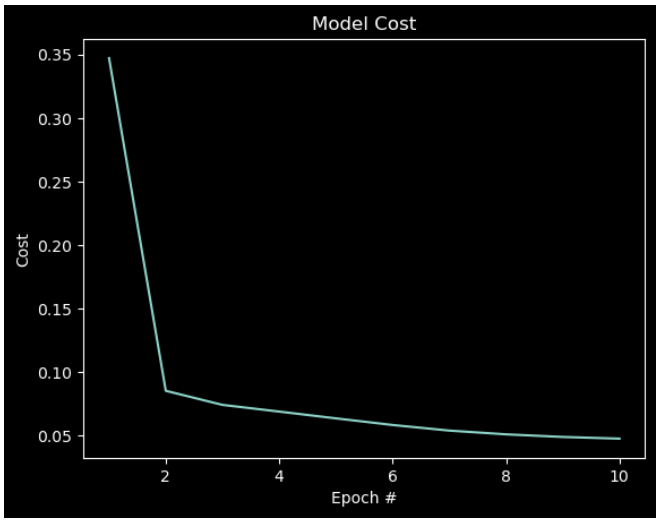
To judge the effectiveness of the model, I need to find the percentage error which is:

$$Percent\ Error = \frac{\sum_{i=1}^{n} \frac{error_i}{pred_i}}{n} \tag{13}$$

Percent Error is different from the MAE, because it controls for the size of the prediction for each error. An error of 0.2 on a predicted value of 1.7 where the investor has already made back his money may be less important than the same error on a predicted value of 0.9 where he is unsure whether he has recouped his investment.

Training

The model was able to train well, as shown in the graph below.



An epoch occurs each time the model runs through the entire dataset. The more epochs a model adds, the more chances it has to train the weights of the model to optimize the outcome. The cost is the difference between the model's prediction and the correct answer. The sharp decline in the cost during the first epoch and its gradual decline onward shows that there are clear patterns that warrant deep learning and that it does make sense to proceed.

Cross Validation

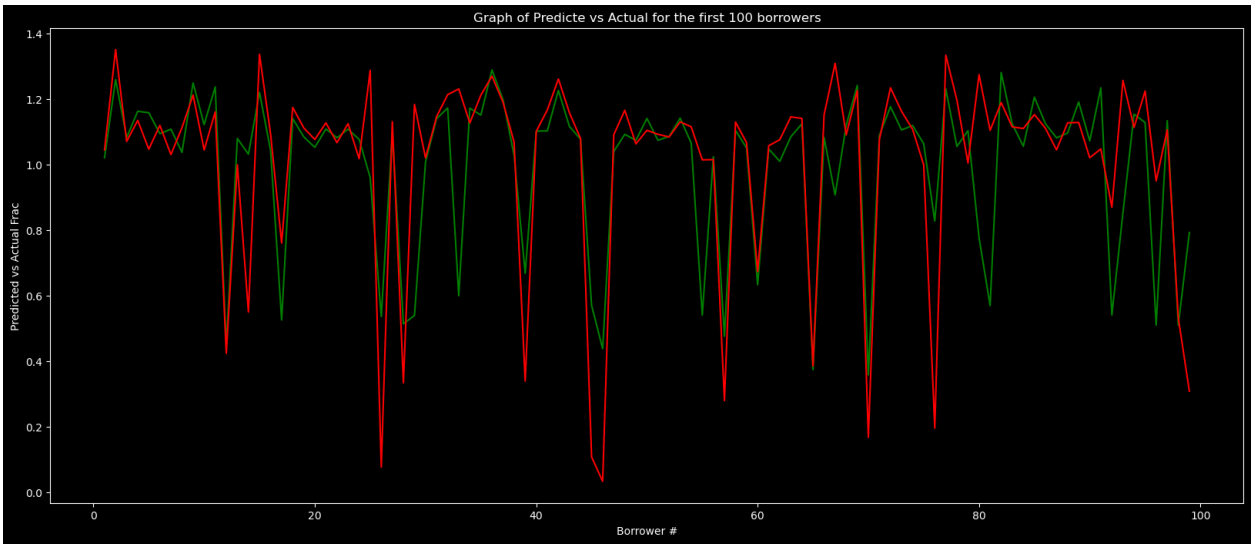
To show that a model is generalizable, it is important to use cross validation as a way of assessing the model without using the test set. This way I can see if the model is able to find the signals in the data without using the noise of the training data.

I ran the cross validation five times and each time had an MAE in the 0.17 to 0.2 which is slightly higher than the MAE of the training set but not egregious enough to stop using the model.

Testing

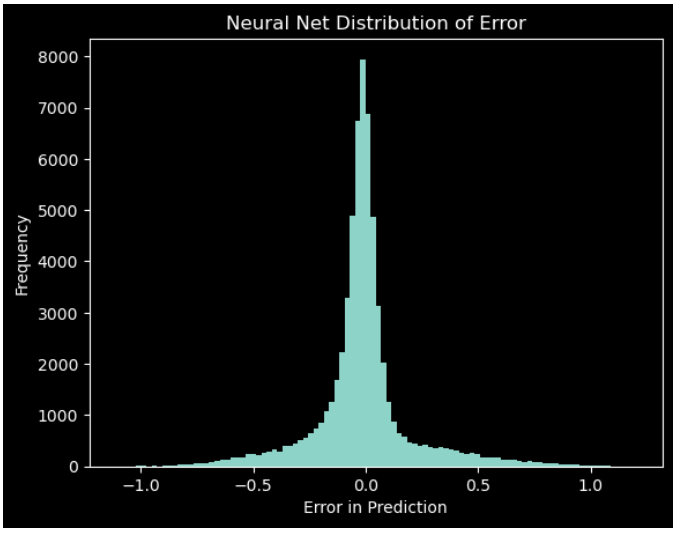
The model had an MSE = 0.0490, RMSE = 0.2213, MAE = 0.1386, and Mean Percent Error of14.6275%. While it is easy to understand MAE and percent error, the root mean square error is more concerning. This outcome shows that the model is most likely accurate for a lot of guesses but is very wrong on others.

The graph below shows the first 100 loans with predictions in green and actual values in red. What sticks out is that while the predictions are following the same shape as the actual



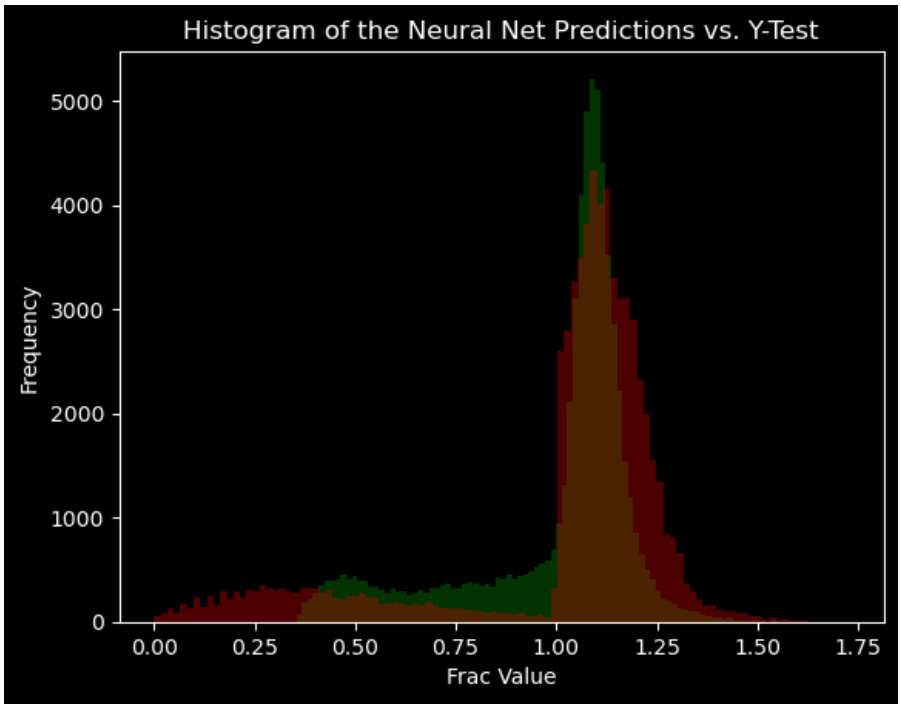
values, there are some values that it either significantly under or overestimated. Further research would reveal whether the outliers are easily diagnosable by a financial professional or if the model is significantly underperforming.

Below is a histogram of the error for the neural network. This gives a much better visualization of how the model performed, and the number of outliers that are in the model.



It is reassuring that there is little heteroscedasticity as that means that there isn't an intrinsic flaw with the model. There appears to be a lot of values outside of -0.5 and 0.5, which is concerning given that the largest values of `*frac*` is 1.76 which is only three and a half times as big as those errors.

A more direct way of looking at the results from the model is to overlay a histogram of the predictions onto the actual values.



Interestingly, the model difficulty predicting loans that returned less than the expected investment. Given this outcome, it is useful to consider a second level of models that can transform the underperforming loans into the distribution of the actual loans.

LIME

LIME stands for Locally Interpretable Model Explanation. It attempts to explain deep learning models in a more understandable form. While the full process of a model's prediction may never be known, an explanation of the model can reveal the importance of each variable. LIME does this by evaluating the model's prediction compared to a hypothetical linear instance of the model at the point of the observation.

To create the local instance, LIME creates a binary representation of an observation x:

$$x \in \mathbb{R}^d \rightarrow x' \in \{0,1\}^{d`} \tag{14}$$

LIME uses binary because it is an obvious way to record whether the model considers a variable. The binary representation x' may have significantly more dimensions depending on the type of data in x. LIME creates an arbitrary number of random samples z_i' through perturbations by changing some of the values in x' to 0. Then we can use an inverse mapping function to convert all the binary perturbed samples back into nonbinary samples. The samples are weighed by the following kernel:

$$\pi_x(z) = e^{\frac{-D(x,z)^2}{\sigma^2}} \tag{15}$$

Where $\pi_x(z)$ is a weighting function with respect to the observation x, and z is one of the non-binary perturbations. D is a distance function between the original observation and the sampled point. This ensures that the closer the sample is, the more impact it will have on the explanation. Since the original model is complex and nonlinear, the farther away the point is, the less it is going to explain the linear model.

We need Fidelity and interpretability to understand a model. Fidelity is how close the new local linear model is to the original model. In equation form it is

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in Z} \pi_x(z)(f(z) - g(z'))^2 \quad (16)$$

Where $f(z)$ is the result that would have happened if the sample point had been in the original model, and $g(z')$ is the result of the binary perturbation in the simpler explainable model. This equation sums the squares of the difference between the models and then weighs each point on how far it is from the original observations. If eq (16) is a small number, then it is considered a faithful adaptation.

Interpretability can be regarded as a measure of the complexity of a model. For example, a decision tree with three layers would have a complexity of three and would be simple to understand, but a neural network with 500 nodes would have a complexity of 500 and be impossible to understand. We can represent complexity as the following:

$$\text{Complexity(g)} = \Omega(g) \quad (17)$$

As such the LIME is the following equation:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}}(\mathcal{L}(f, g, \pi_x) + \Omega(g)) \tag{18}$$

Where the ξ represents a function closest to the original model while also being the simplest. To find the variables of interest from this construction, we can use Ridge regression. The way to represent the complexity of a linear model is by the number of coefficients,

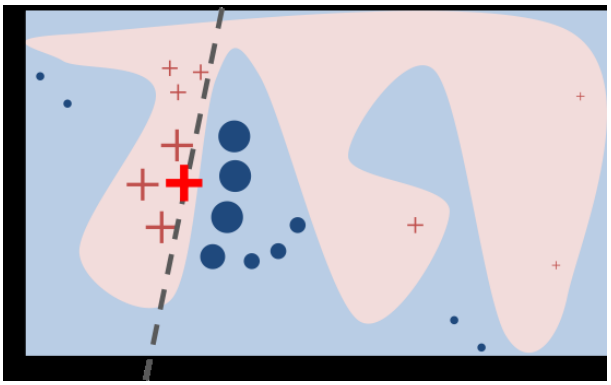
$$\Omega(g) = \lambda \sum_{i=1}^{n} \beta_i^2 \tag{19}$$

Where β_i represents the coefficient of the *i*th variables in the local linear model. When combined the entire equation becomes:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \left(\sum_{z,z' \in Z} \pi_x(z)(f(z) - g(z'))^2 + \lambda \sum_{i=1}^{n} \beta_i^2 \right) \tag{20}$$

This equation bears a shocking resemblance to the equation for lasso regularization[9]. The middle term is a Sum of Square Distances or SSD, and the other term is a penalty term with all the coefficients of a linear model. By performing ridge regularization instead of lasso, I can find the features that are the most important and the amount of impact that they have on the observation.

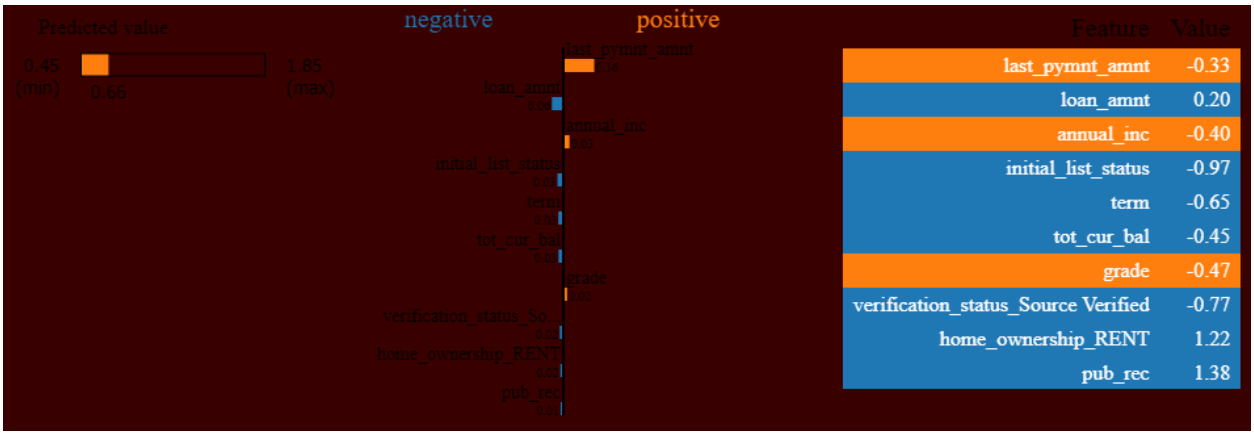
Below is a graphical representation of how LIME works:



[9] See eq.(3) above.

The white vs. blue parts represent a complex non-linear model, whereas LIME creates the simple dashed black line, which is much easier model to interpret. The observation in question is the bolded red plus sign that the model is trying to explain. When we examine close to the point of interest, a linear model is effective but as the model curves and turns it is no longer helpful. [10]

With the LIME python package, I can quickly use LIME to explain any given loan and determine its most impactful features. Below is an example of the output of the LIME package:

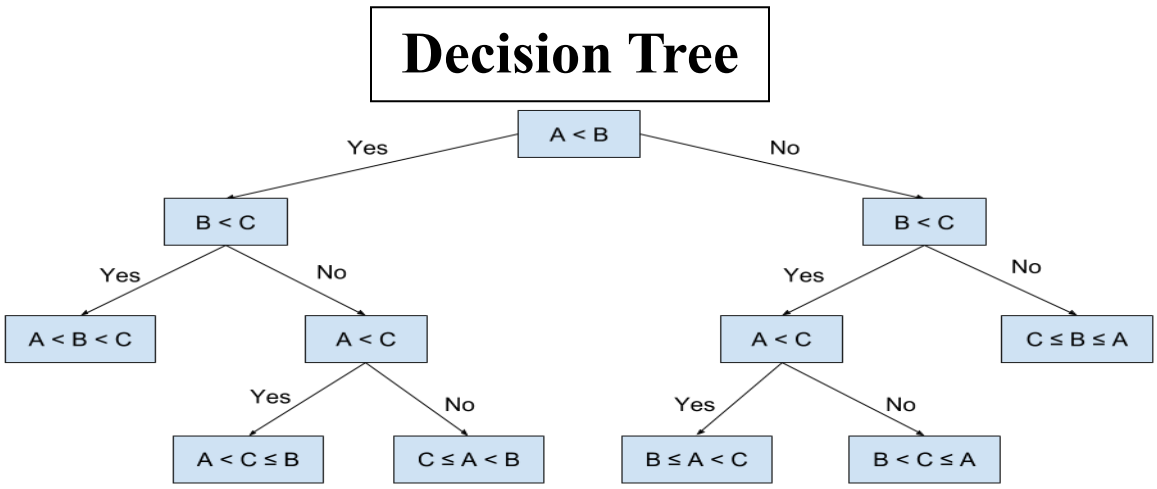


The predicted value on the left stems from the possible values that the inverse mapping function creates when transitioning from the binary vectors to numerical vectors. The graphical representation shows off the importance of the features from the ridge regression. Finally, the last column shows the actual values that were in the observation that underwent the LIME analysis. Since I scaled every value to allow the models to capture all the relationships between the variables, the values themselves do not mean as much. If an explanation were shown to an investor, I could use the original values of the loan instead of the processed scaled values that are devoid of real world meaning.

[10] The LIME section is inspired by the paper "Why should I trust you?" which forms the basis for the LIME method (Marco Tulio Ribeiro, 2016)

Random Forests

"Random forests are a collection of decision trees that predict a target variable based on the features of the dataset. Below is an example of a decision tree:



When a decision tree is applied to a dataset, it has specific ways of creating decisions. If the data are binary, then the decision tree will create two branches for each of the binary values. If the data are categorical then the decision tree can choose any number of the categories to create another branch. If the data are numerical, then the decision tree creates a cutoff value where all observations with values below the cutoff go to one branch, and the rest go to another branch.

"The decision tree then partitions the least homogeneous feature of the dataset as measured by the Gini index which is:

$$Gini = 1 - \sum_{i=1}^{n} {p_i}^2 \quad (12)$$

Where p_i is the probability of each of the classes and n is all the classes. The decision tree will continue to iterate through this process until it reaches 100% purity and has nothing left to

partition. When all the observations in each group are the same, then the y-class that is the most popular for that group becomes the prediction of the model.

"This can lead to overfitting as the decision tree has learned all the patterns of the training data that may or may not be helpful. To fix this problem we can create a forest of decision trees with bootstrapped datasets. Bootstrapping is when the data is randomly sampled with replacement to create another dataset that has the same parameters as the original dataset but does not carry the same noise. When all the trees in the forest have finished their processes, they vote on the correct prediction. The prediction that the greatest number of trees support is the prediction of the forest.

"There are many different hyper-parameters for a random forest model. Some examples are:

- Number of decision trees
- Homogeneity metric
- Maximum depth of an individual tree

The only hyper parameter that I had time to test was the number of decision trees because the calculations took significantly more time as the number estimators increased. It is possible that there are some hyper-parameters that would have produced significantly better results, but I did not have the time to find them."[11]

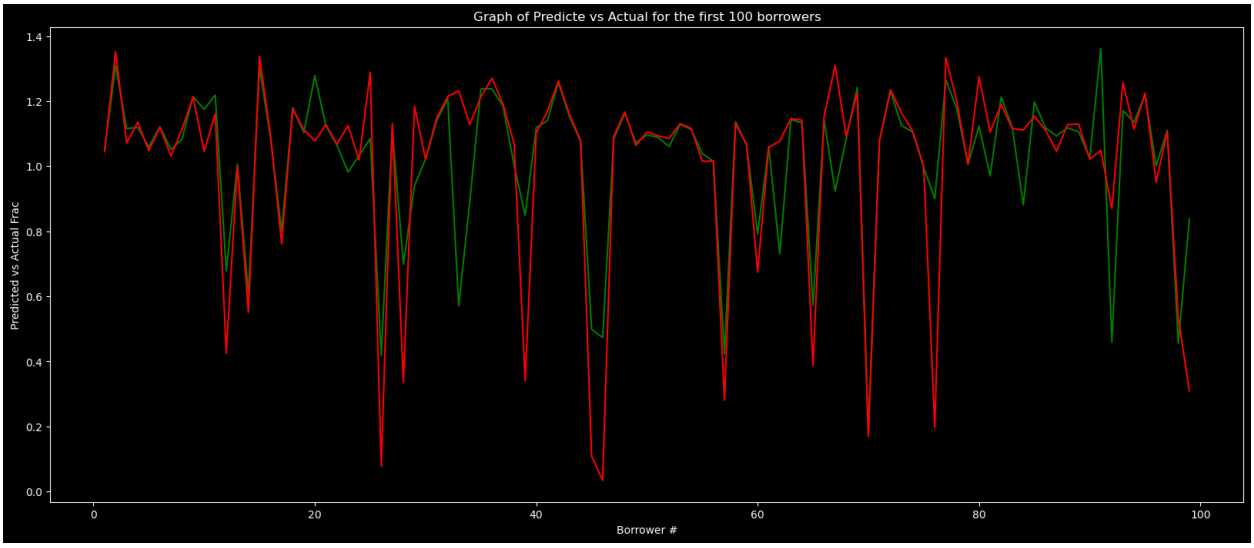
When trying to find the number of trees for the forest, I tested 5, 50, 100, 150, 200, 300, and 600 estimators. After 600 estimators adding estimators does not provide significant

[11] This was taken from the section on Random Forests in my Network Intrusions report (Drelich, 2023)

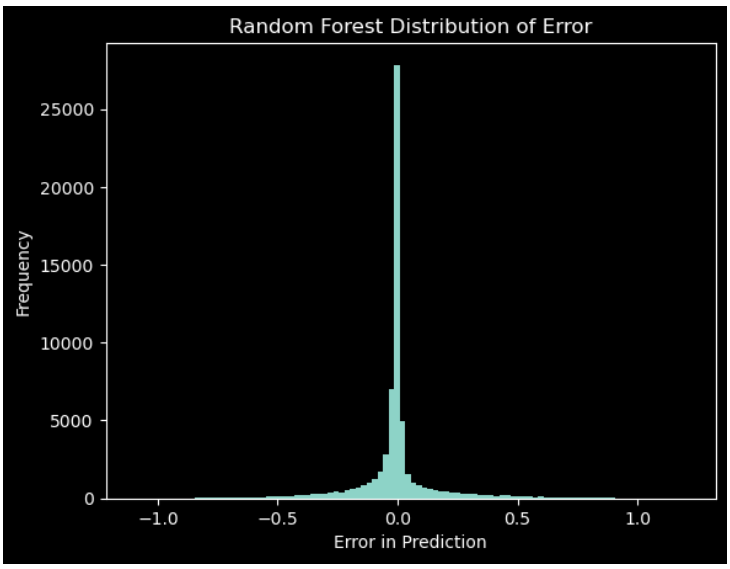
improvement to the model. Given the amount of time that it takes for my computer to evaluate the different numbers of estimators, 600 is the maximum practical amount available to me.

Testing

The Random Forest model had a MSE = 0.0309, RMSE = 0.1758, MAE = 0.0872 and a Mean Percent Error of 13.6849%. The discrepancy between the RMSE and the MAE shows that there are a lot of outliers. Below is a representation of the first 100 loans with the predictions in green and the actual values in red:

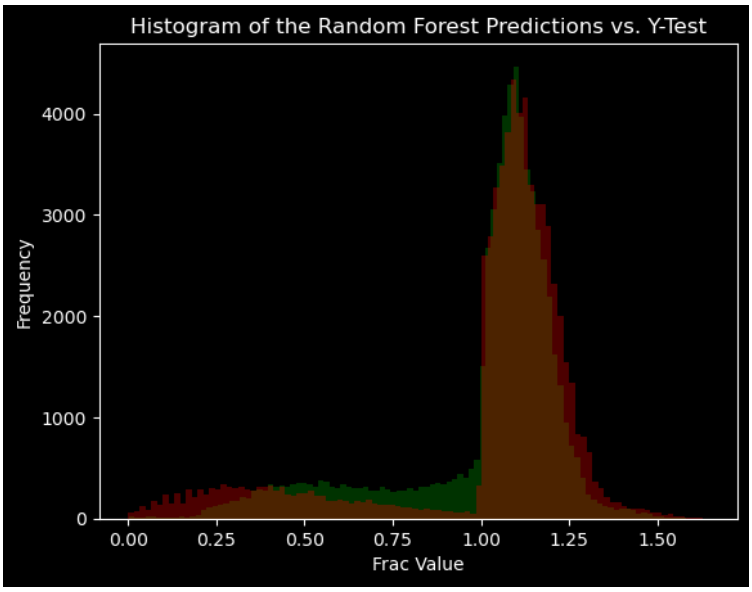


Interestingly for every single observation the model has the correct pattern but gets the amplitude wrong. This matches the RMSE discrepancy, where many of the predictions are correct but some of the wrong estimates are severely wrong. Below is a graph of the distribution of error for the random forest.



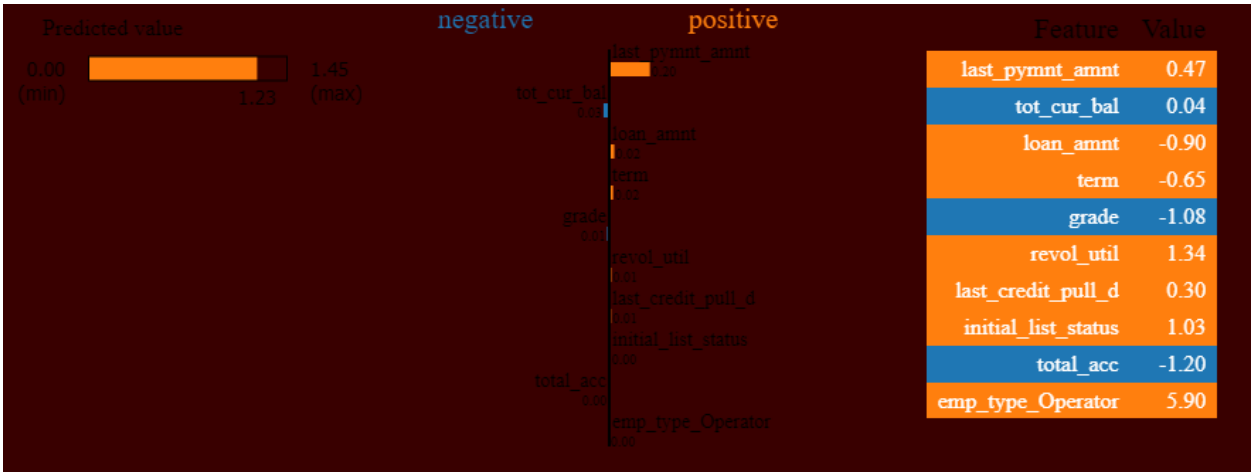
It is stark how much more concentrated all the values are around zero, but the outliers still remain.

Finally, I can compare the histograms of the actual values to the predictions from the random forest.



This is better than the neural network, but the model again is struggling to predict values near one. The overlap of values over one is interesting. If I had used a classification model that predicted the borrowers that will pay the loans in full, then it would have scored very highly.

Lastly, I have the LIME output of a random loan using the random forest model:



The random forest model used the `last_pymnt_amnt` more than the neural network and was able to get a better score.

Conclusion

This was not an easy dataset to analyze. The simplest question: “Is investing in a given loan a prudent use of money?” did not have enough features that would be available at loan origination to create a viable model. By thinking deeply about the type of analysis the dataset could support I was creating a market that could be helpful to current and prospective investors.

The most important methods from this paper are:

- Locally Interpretable Modelling Explanation (LIME)
- Random Forest Modelling

With most of the dataset comprised of current loans, it is impossible to evaluate the accuracy and precision of supervised learning methods. As such it is crucial to have an algorithm like LIME that can explain the model so that if someone is curious as to why the current valuation of a loan is a given number, they will be able to look behind the curtain.

The best model was Random Forest. Which isn’t surprising since random forests are better at predicting tabular data with simpler models. It could be that If I was able to better tune the hyperparameters of the neural network I might have had a different result, but this is the best that I could do.

References

Bibliography

Bank, W. (2022). *GDP growth (annual %) - United States.* Retrieved from The World Bank: https://data.worldbank.org/indicator/NY.GDP.MKTP.KD.ZG?locations=US

Drelich, J. (2023). *Network Intrusions.* Washington D.C.: Github.

Federal Reserve Bank of New York. (2023). *Household and Credit Debt.* New York: Research and Statistics Group.

Frost, J. (2020, December 6). *Variance Inflation Factors (VIFs)*. Retrieved from Statistics By Jim: https://statisticsbyjim.com/regression/variance-inflation-factors/#comments

Marco Tulio Ribeiro, S. S. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Arxiv*, 3-4.

Menor, D. (2022, December 15). *How to Calculate VIF in Excel*. Retrieved from Sheetaki: https://sheetaki.com/how-to-calculate-vif-in-excel/

Schulz, M. (2023, August 25). *Personal Loan Statistics: 2023*. Retrieved from lendingtree: https://www.lendingtree.com/personal/personal-loans-statistics/