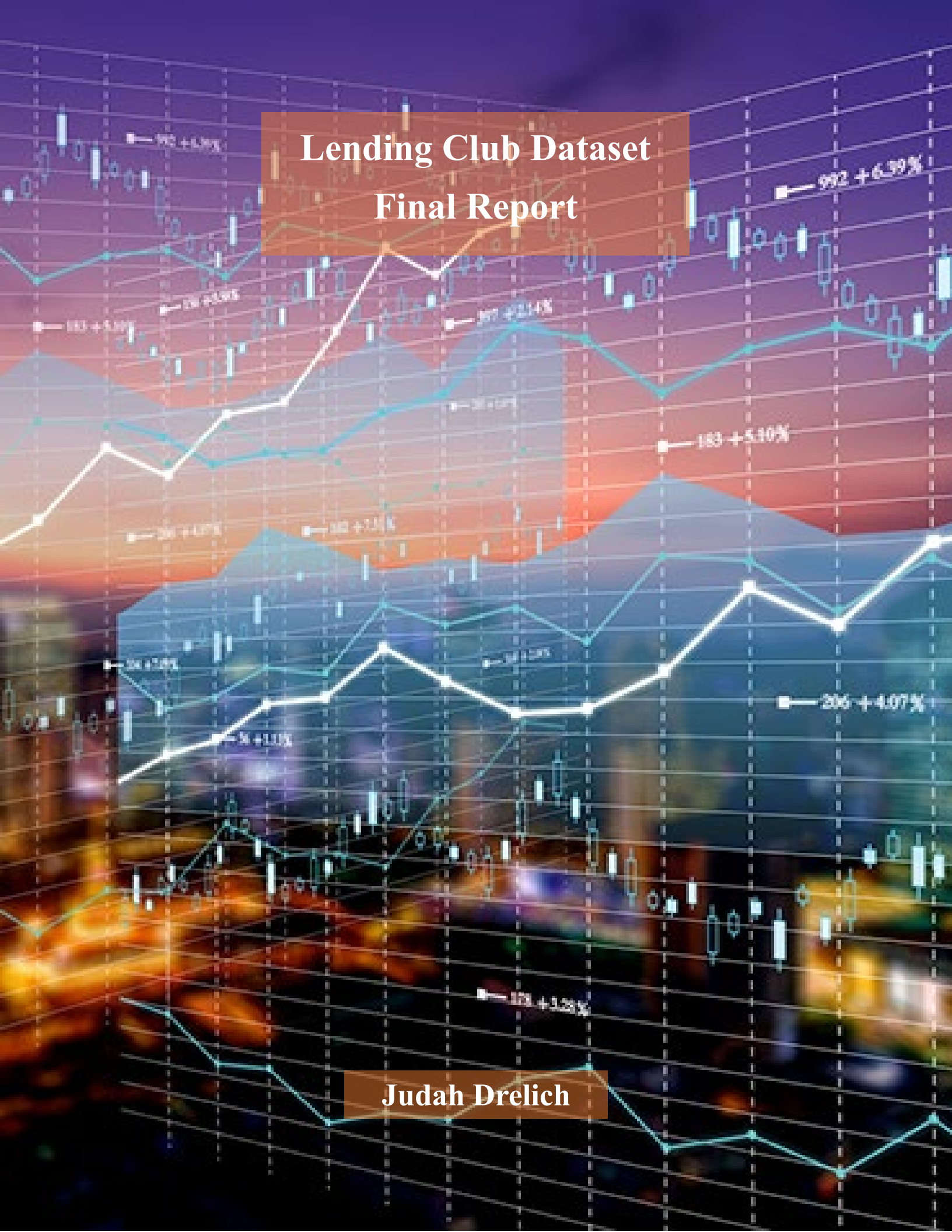


Lending Club Dataset
Final Report

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 investors and institutions to 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 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 use LIME, a machine learning explainer, to interpret the valuation of each loan. The best model 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 buyer to the seller until the buyer can repay the seller. Many Americans have entered into debt without having a clear way out. By Q2 2023, total credit card debt was $1.03 trillion and student loan debt of $1.57 trillion.[1] One way for people to 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] Household Debt and Credit Report - FEDERAL RESERVE BANK of NEW YORK (newyorkfed.org)
[2] Personal Loan Statistics: 2023 | LendingTree

Lending Club is an online service that matches borrowers to investors or institutions to facilitate loans at lower interest rates. For the borrowers, Lending Club offers a chance to either relieve debt or pay for an expense that they could not afford otherwise. For investors, Lending Club is an opportunity to make a profit while helping someone else. Lending Club offers some of their most stable loans financial institutions, however, most of the loans within Lending Club's portfolio are owned by private investors. 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

The first step in data wrangling is to separate the features (X) from the target variable (y). When I initially started this project, I had believed `*loan_status*` was the target variable. The 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 performed.

As I looked closer at the data, I realized that it was hard to predict `*loan_status*` with many 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 a 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 question that I am asking so that it is concurrent with the dataset. Instead of assessing the loans before origination, I can find the value of the remaining loan as of the time of the dataset. That way `*next_pymnt_d*` is a variable that will have happened and is viable in future models. market or evaluation for active loans that would be valuable for either loan investors or institutions who are thinking of selling off their loans for immediate liquidity.

The new target variable needs to be applicable 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 fraction of the amount paid back compared to the expected payback. 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 can create the fraction while the remaining columns will become 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 the letters and convert the value from a string into a number. Other variables are purely categorical such as `*emp_title*` which gave the type of profession the borrower worked. and needed to either be one-hot encoded or dropped if they had too many unique values.

One of the best tools for 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 can consolidated into a single cell.

- **Missing values:**

The imputation of values for this dataset is one of the most challenging aspect of this project and data science as a whole. There are many columns that have large swaths of missing values that need reasonable values filled in to 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 that borrowers never recorded. For instance, `open_il_12m` refers to the number of opened installment accounts in the last 12

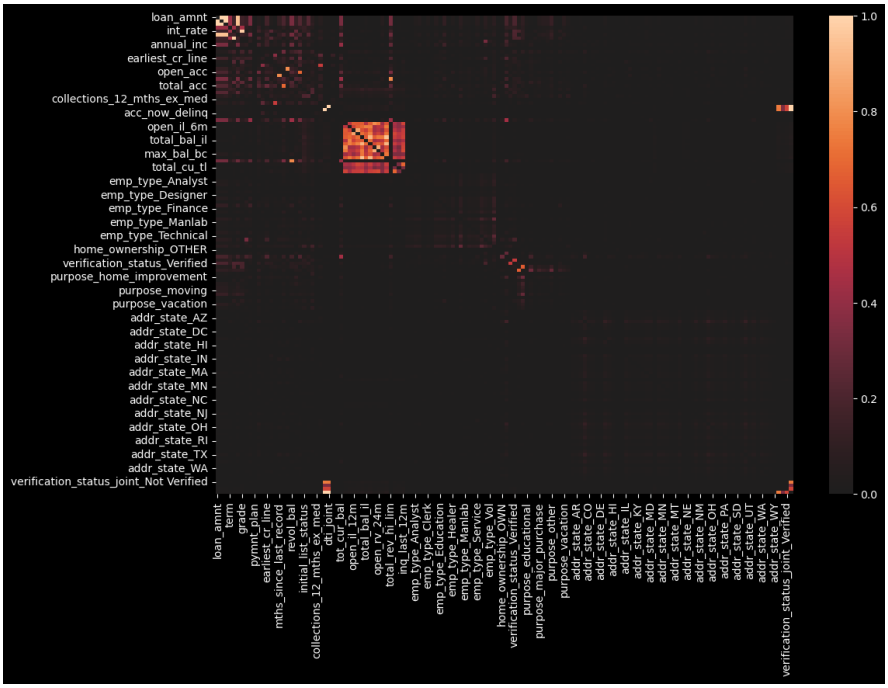
months. If there isn't any information on the number of installment accounts, then there likely aren't any installment accounts and the person left that part blank on the application.

Values such as `mths_since_last_delinq` or months since last delinquent are different because if someone was never delinquent then zero is the exact wrong answer because it would imply that the borrower is currently delinquent. The higher the value, the better the outcome on their loan. I use the value of 1000 for missing `mths_since_last_delinq` values because 83 years' worth of non-delinquency is the equivalent of never being delinquent.

- **Multicollinearity**

Multicollinearity

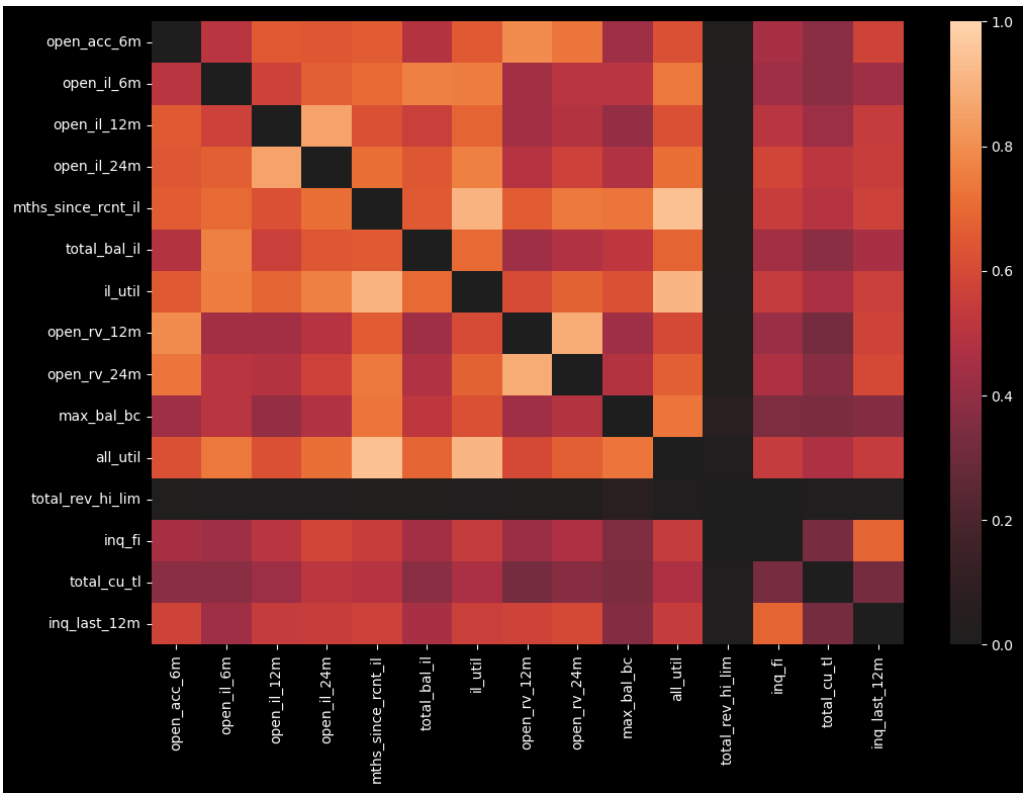
Multicollinearity is more than two features varying together. This is bad because it becomes hard to assign correlation or causation to any given feature since there isn't a clear mathematical distinction between either of the features. Below is the starting point of the multicollinearity for the entire 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 shows a concerning square from features 27 to 42. Cleaning up the collinearity 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.

The 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]



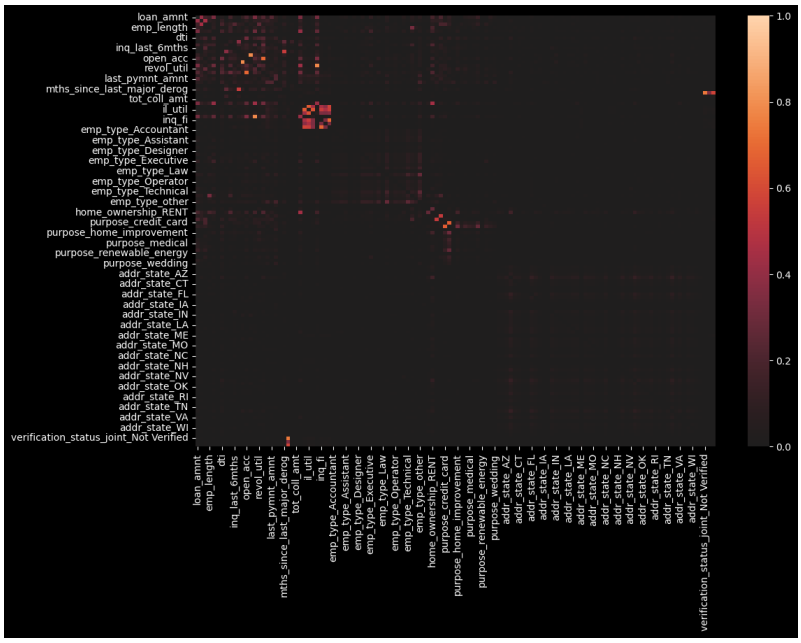
[3] Taken from the Final Report on Network Intrusions

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 open. Instead of having to drop two of these 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 took the average of 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 was 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 will be 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:

[4] How to Calculate VIF in Excel - Sheetaki
[5] Variance Inflation Factors (VIFs) - Statistics By Jim

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

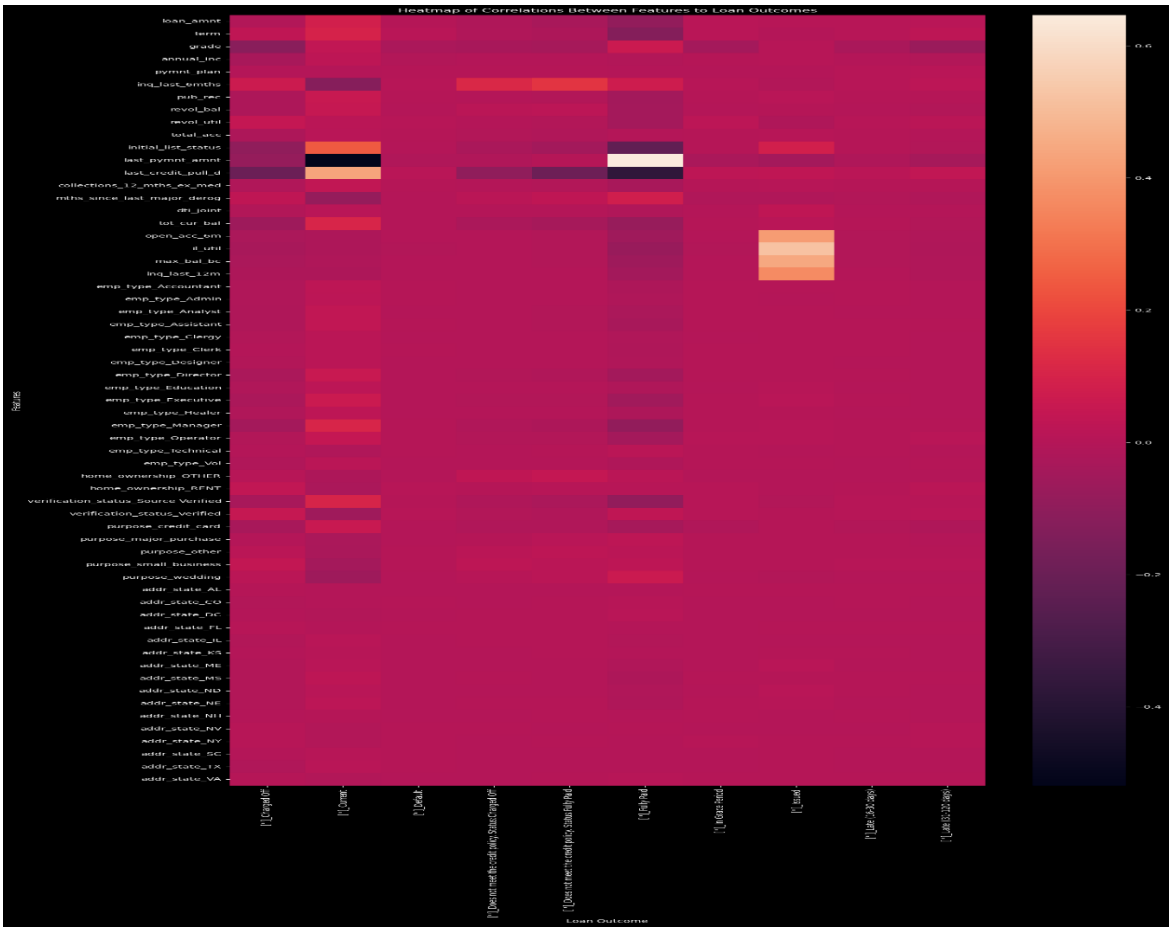
Lasso aims to reduce the loss in equation (3). If a feature is not important, then changing its slope will not move the regression line close enough to the data points to decrease the loss function with a non-zero value. If a feature is important, the regression line will move towards the data and minimize 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] Taken from the Final Report on Network Intrusions

Exploratory Data Analysis

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 easy to analyze.

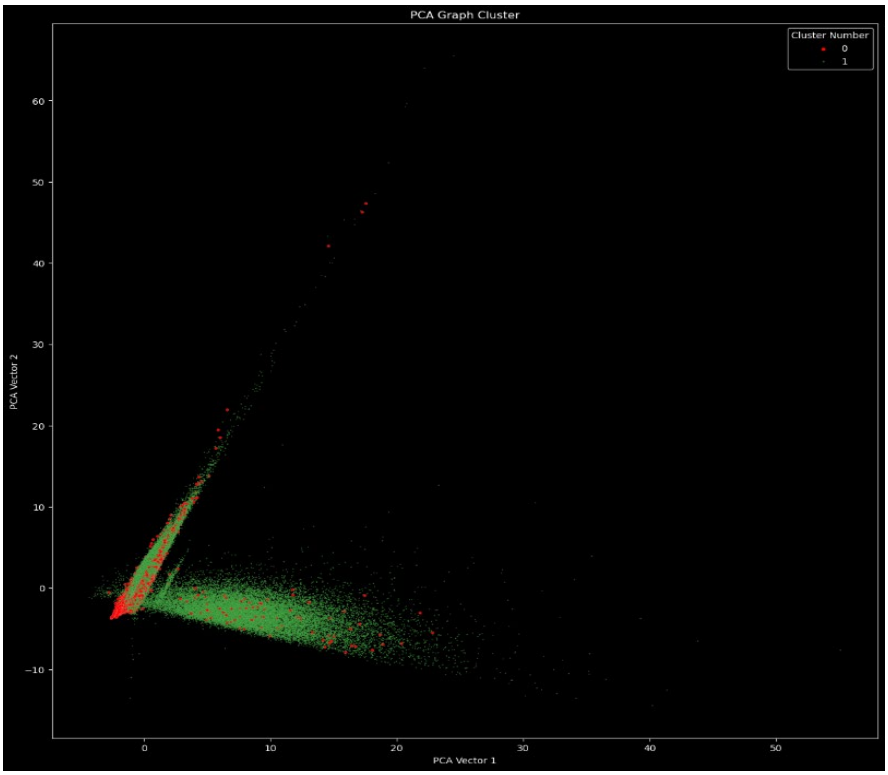


The graph above is a heatmap of the correlation between the features and each potential outcome of the loan. Most of the columns have correlations near zero because they represent infrequent loan outcomes. Overall, there are only a few features that have an impact on `*loan_status*`. The feature with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

Clustering

One way to try and see deeper patterns is to cluster the data based on its structure. Instead of looking at singular features that would pop in correlation metrics, now there can be multiple features that can combine together to set the data apart. Given the different outcomes in the `*loan_status*` column, it would seem evident that there should be at least three groups of borrowers. Those that fully pay on time, those that are late but pay in full including late fees, and those that cannot pay in full.

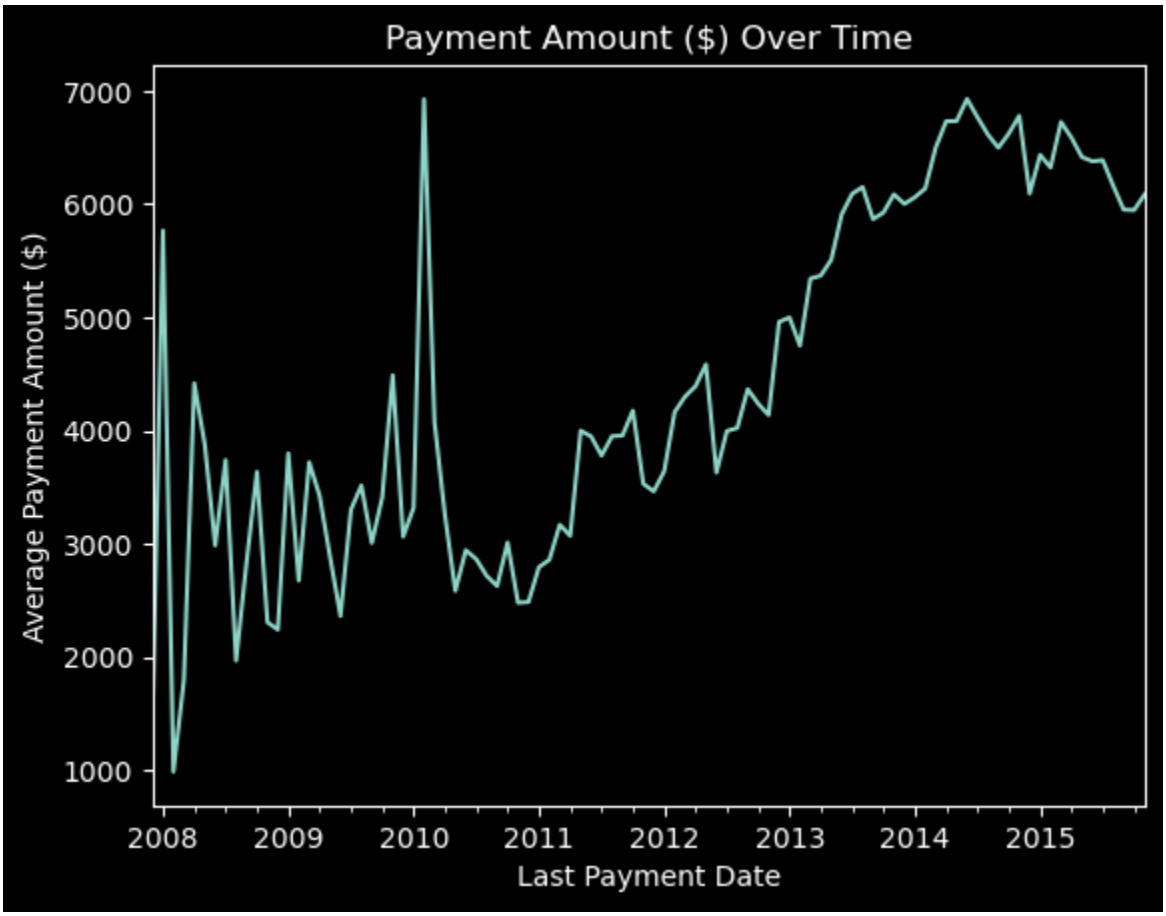
Before creating the clusters, I visualized the dataset with Principal Component Analysis (PCA). PCA is method that reduces the number of variables by selecting the features with the highest variances and combining them into a few vectors. This allows a person to visualize high dimensional data and see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



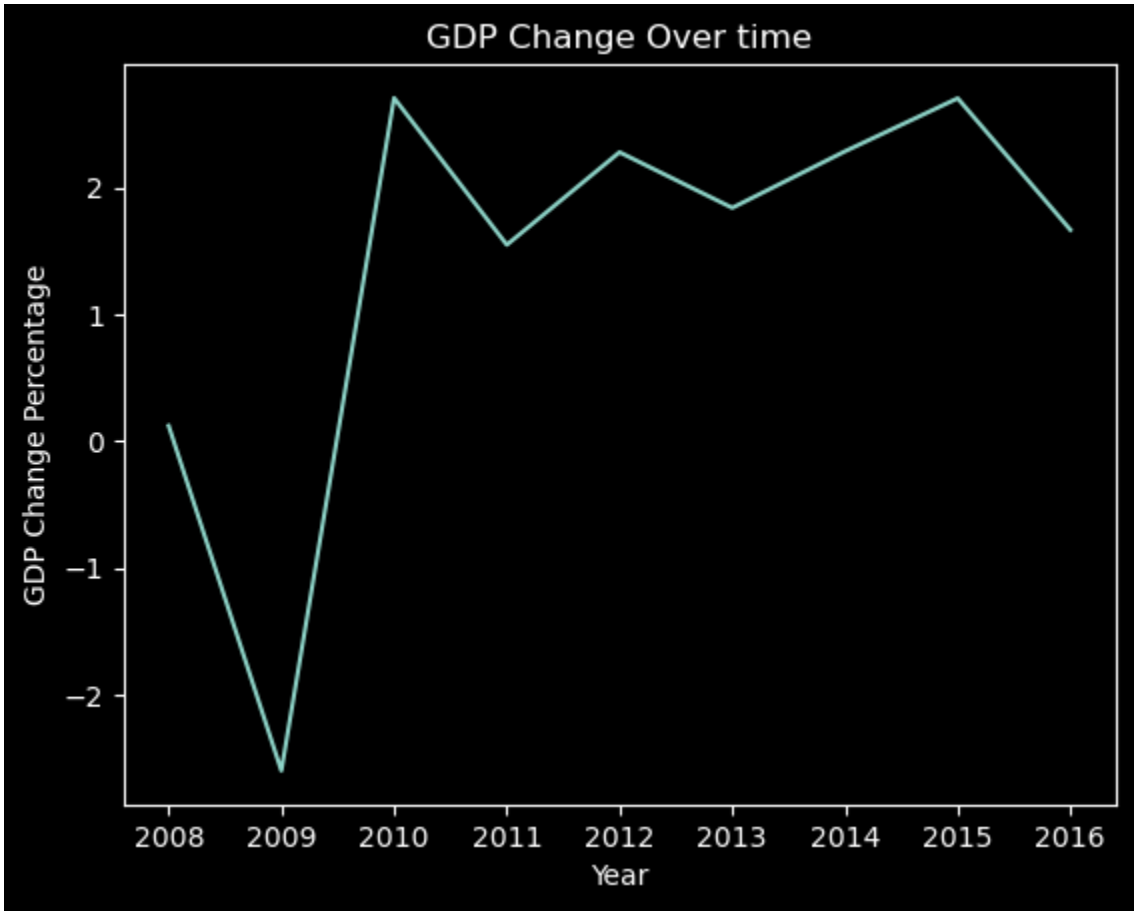
The graph above shows that while the bad loan outcomes in red congregate around the left perpendicular vertex, the points are too dispersed throughout the structure to continue clustering in good faith.

External Factors

Any financial system that occurs over many years and involves people's critical financial resources is going to be subject to greater economic forces. For borrowers, the state of the economy can impact the level of liquidity that they 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. To do this 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, some of the models will have errors that will never disappear.

[7] (Bank, 2022)

Preprocessing

Only a part of the dataset has loans that have finished. While the end goal of the project is to create a market with active loans, the only way to evaluate the strength of the model is through data points that already have a result. Therefore, I can separate out the finished loans from the unfinished loans by the `*loan_status*` column that describes the current state of the loan.

Originally, I had thought that `loan_status` was the target variable but the active loans had too much information that came after the time of origination. I pivoted to trying find the amount that the loans are worth for active loans. To do this I can calculate the expected return on each loan from Lending Club. Lending club does not compound interest for its loans, so the formula is:

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

I can then calculate the amount that has been paid back by this formula:

$$Actual\ Payment = Total\ Payment + Total\ Late\ Fees + Recoveries$$

Where `Total Payment` is the amount that has been paid back from the loan, `Total Laxzte fees` are the late fees that the borrower would have incurred if they were late with a fee. `Recoveries` is everything that that the company was able to recover after the loan was charged off.

`frac` is the fraction in which

$$frac = \frac{Actual\ Payment}{Expected\ Payment}$$

The active loans are slightly different in that the expected payment is what is remaining on the loan instead of the entire loan. To calculate this I did:

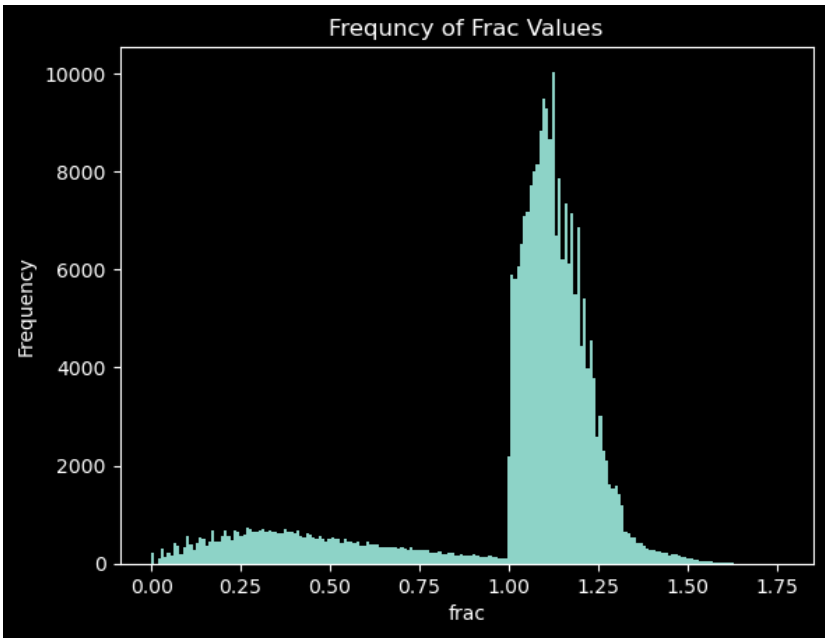
$$Expected\ Payment\ =\ Principal\ *\ (1 + Interest\ Rate\ *\ Years)\ -\ Actual\ Payment$$

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.

Modelling

EDA

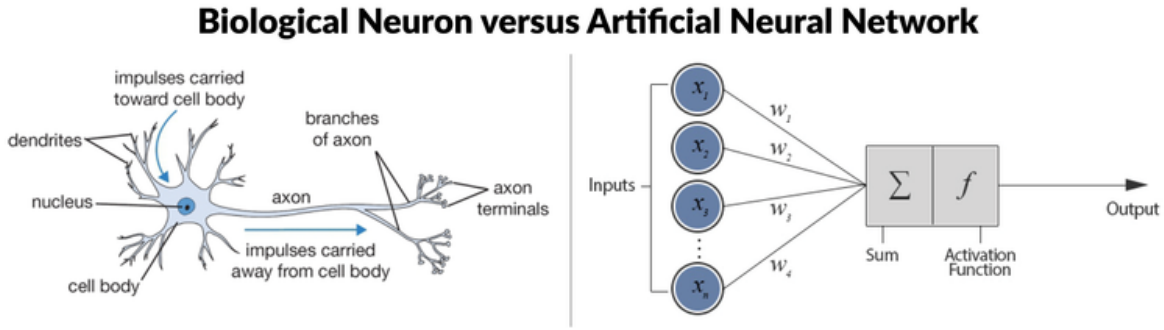
Before I start the modelling sections, it is important to know how a little bit more about the `*frac*` variable. Below I created a histogram of the distribution of frac to gain an idea of what the model is trying to predict.



Most of the borrowers paid off their loans so the frequency around 1 jumps. There is also a decline the closer `frac` gets to 1 because it is less likely that a loan that is close to being paid off won't be fully paid off rather than a loan that the borrower has no hope of paying off will not be paid off.

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:

$$Output = f(\sum_1^n w_i x_i) + b \quad (7)$$

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, 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:

$$Relu(x) = \begin{cases} x \geq 0, & x \\ x < 0, & 0 \end{cases} \tag{8}$$

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 of 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:

$$MAE = \frac{\sum_{i=1}^{n} |cost_i|}{n} \tag{9}$$

Where $cost_i$ 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. This gives a sense of the total amount of error that is occurring without the errors cancelling out due to some being negative and some

[8] Taken from the Final Report on Network Intrusions

being positive. I decided not to use mean square error (MSE) nor root mean square error (RMSE) which are:

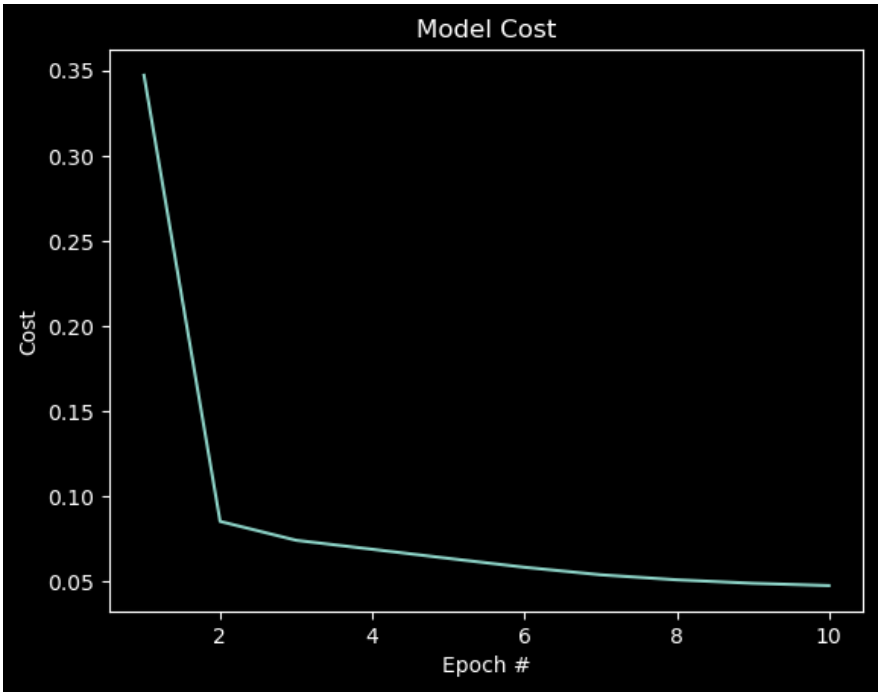
$$MSE = \frac{\sum_{i=1}^{n} cost_i^2}{n} \tag{10}$$

$$RMSE = \sqrt[2]{\frac{\sum_{i=1}^{n} cost_i^2}{n}} \tag{11}$$

Because the results are significantly harder to interpret. While adding a square does increase the impact that outliers or larger errors have, to judge the effectiveness of the model I need to find the percentage error. With RMSE its not clear what that percentage means whereas with MAE on average the model will be off by the percentage error.

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 gets 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 first epoch and gradual decline onward shows that there are clear patterns that warrant deep learning and that it does make sense to proceed. The training produced an MAE of 0.13 which given that the average value of `*frac*` is around 1 should mean that the percent error is close to 13 percent.

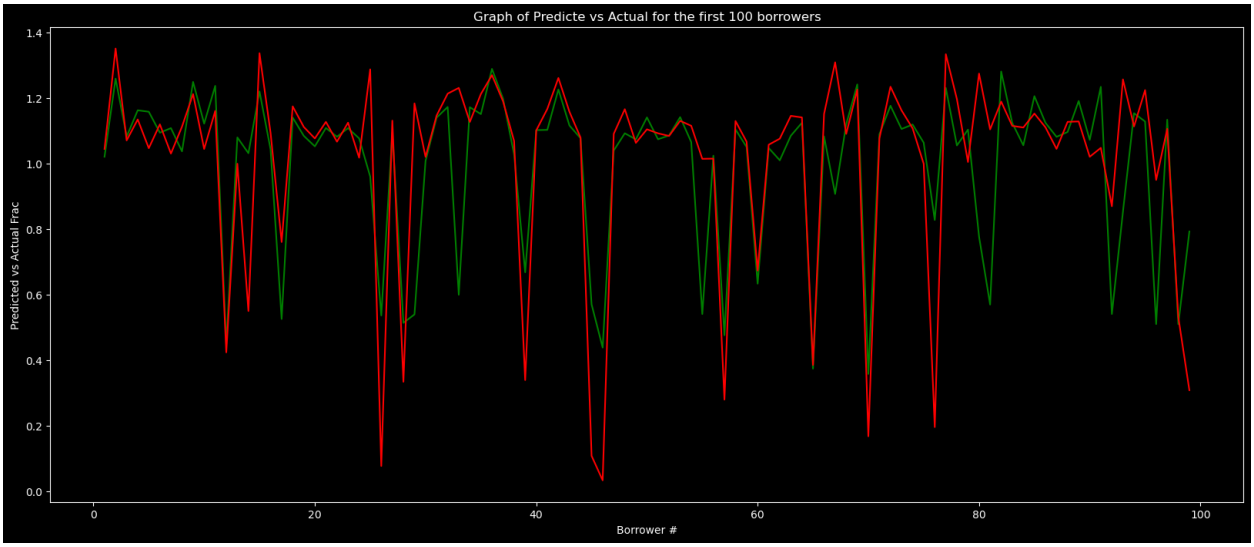
Cross Validation

To show that model is generalizable it is important to use cross validation as a way of testing the model without using the test set. This way I can see how well the model performs when it is able to use new data but it will not have the biases created by training on test data.

I ran the cross validation 5 times and each of them 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 model.

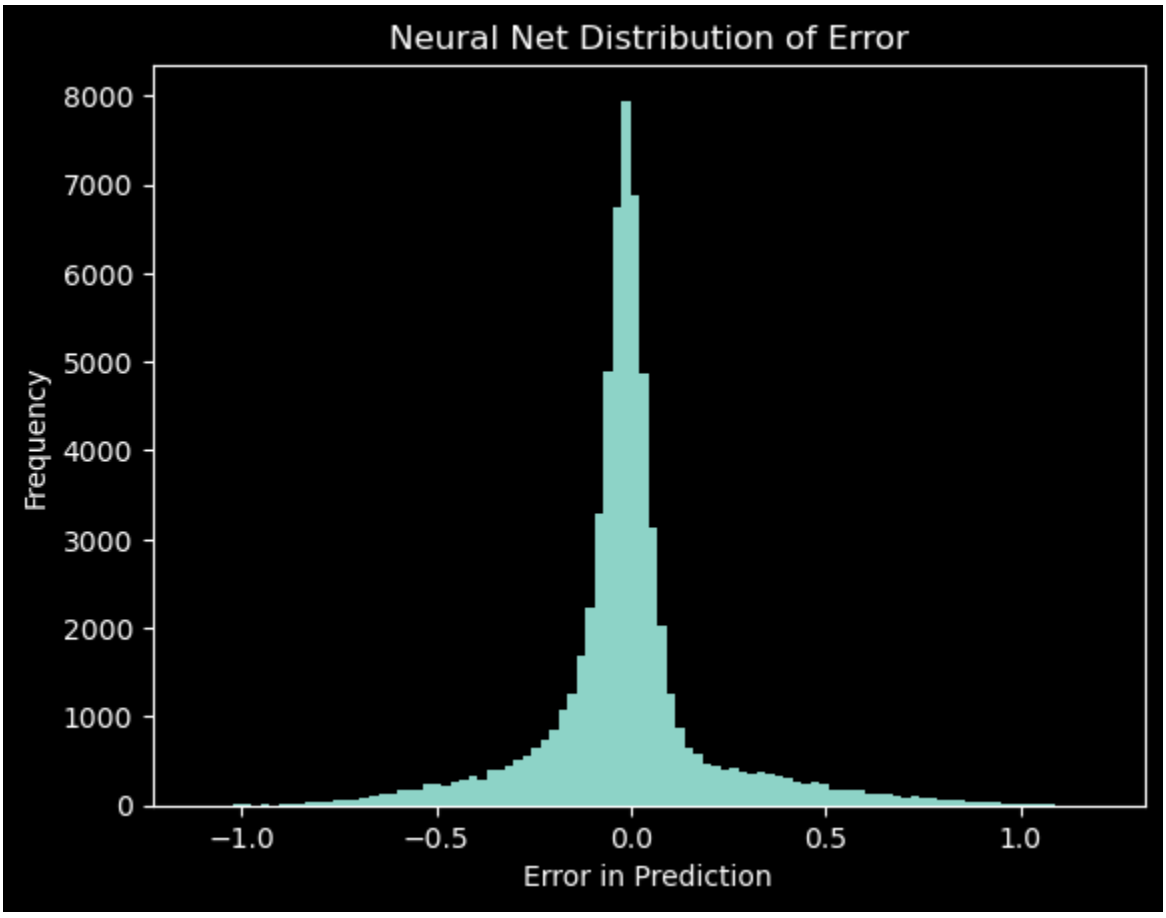
Testing

The model had an MSE = 0.0490, RMSE = 0.2213, MAE = 0.1386, and Mean Percent Error ≈14.6275%. While it is easy to understand MAE and percent error the root mean square error is more concerning. Root mean square is used to highlight potential outliers in the dataset



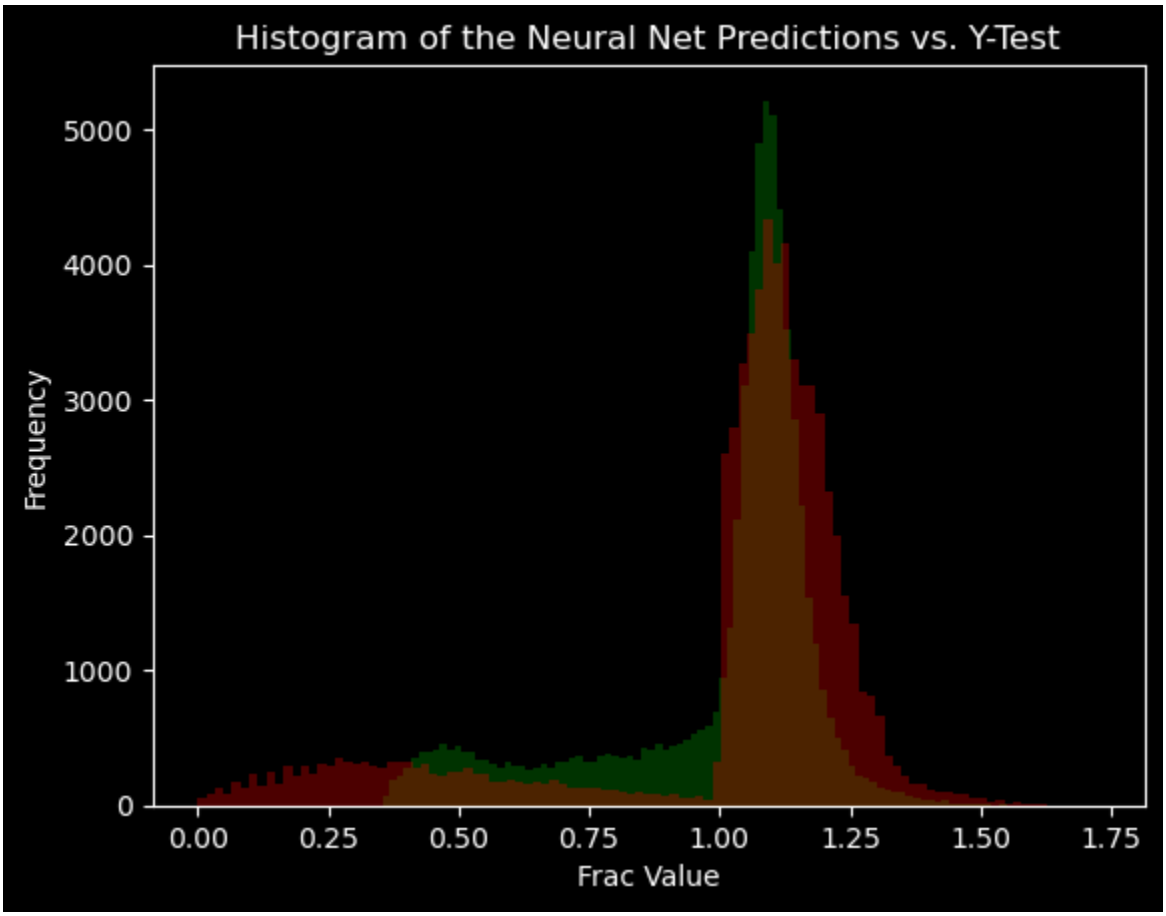
since higher squared values increase faster while taking the square root equalizes it with MAE. This shows that the model is most likely accurate for a lot of guesses but is really off on others. The graph above shows the first 100 loans with predictions in green and actual values in red. What sticks out is that while the predictions are able to follow the same shape as the actual values, there are some values that it either significantly under or over estimated. Further research would reveal if these cases were easily diagnosable by a financial professional and don't need a better model or if the model is not performing well enough.

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



It is reassuring that there is no 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.

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 had an awfully hard time predicting loans that did not recoup at least all of the expected investment. Given an outcome like this it is possible to consider a second level of models that could transform the underperforming loans into the distribution of the actual loans.

LIME

LIME stands for Locally Interpretable Model Explanation. It is an attempt to explain deep learning models in a way that humans can understand. While the full process of a model's prediction may never be known, at the simplest levels there is a way to have a basic understanding of the variables that the model cares about. LIME does this by evaluating the model's prediction compared to a hypothetical linear instance of the model at the point of the observation in question.

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

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

This binary representation x' may have significantly more dimensions depending on the type of data that is in x. The reason that it becomes a binary representation is that lime is trying to find the impact of each of the variables. Representing them in binary will show whether they have a clear effect or not. Then LIME creates an arbitrary number of random samples z_i' through perturbations by changing some of the values in x' to 0. Then we use an inverse mapping function to convert all the binary perturbed samples back into nonbinary samples. The samples weighted by the following kernel:

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

Where $\pi_x(z)$ is a weighting function with respect to the observation x. 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 but the explanation needs to be simple and linear.

To find a model that can interpret other models there needs to be two things, fidelity, and interpretability. Fidelity is represented by how close the new local linear model from the sample is to the original model. In equation form it is

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

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 it is a small number, then it is a faithful adaptation.

Interpretability can also be measured by the complexity of a model. For example, a random forest could have 10 layers and would have a depth of 10, Neural networks could have a 500 nodes and have a complexity of 500. The more complex the model, the harder it is to find reasonable explanations for its conclusions. As such the LIME is represented by the following equation.

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

Where the ξ represents a function that is the 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 the number of coefficients,

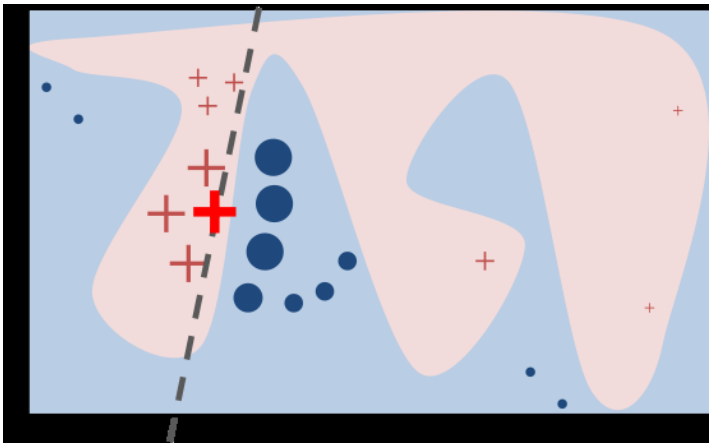
$$\Omega(g) = \lambda \sum_{i=1}^{n} \beta_i^2$$

Where β_i represents the coefficient of the ith variables in the local linear model. When combined the entire equation becomes.

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

This equation bears a shocking resemblance to eq. (3) which is the equation for lasso regularization. The middle term is a Sum of Square Distances or SSD, and the other term is a penalty term with all of the coefficients of a linear model. While it isn't Lasso, since the coefficients are squared, It is perfect for Ridge regularization which uses a squared penalty term. By performing ridge regularization, we can attain the similar results as Lasso and understand not just which features are the most important, but why they are the most important.

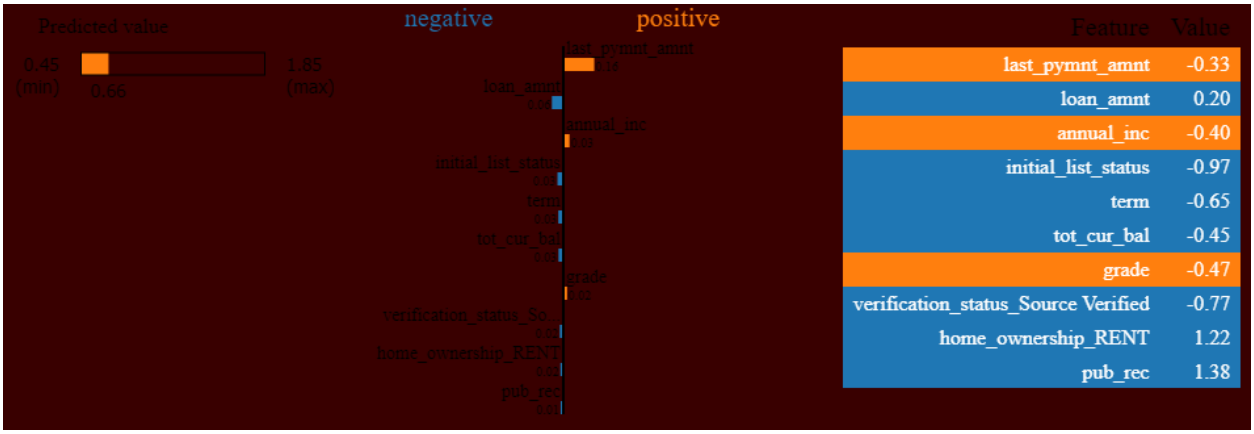
Below is a graphical representation of how LIME works, the white vs. blue parts



represent a complex non-linear model whereas LIME is creating the simple Dashed black line as a 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 appears to be effective but as the model curves and turns it is no longer helpful. [9]

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

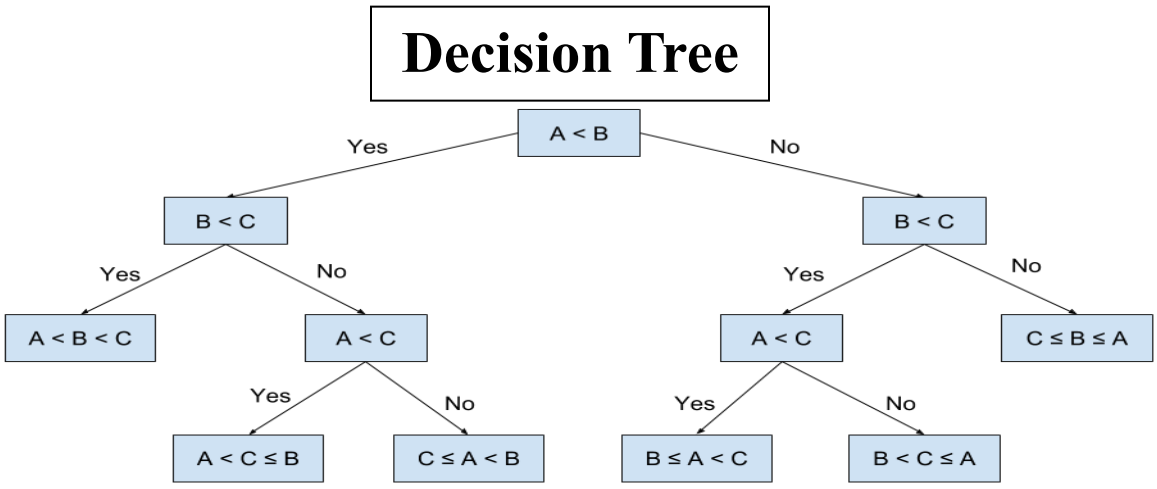


The predicted value on the left is created by the possible values that the inverse mapping function creates when transitioning from the binary vectors to numerical vectors. The graphical representation shows of the importance of the features from the ridge regression. Finally the last column shows the actual values that were used in the LIME analysis. With numerical data is less meaningful than categorical data or binary data but it is still interesting.

[9] (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 decision trees are applied to datasets, they have specific ways of creating decisions. If the data is binary, then the decision tree will create two branches for each of the binary values. If the data is categorical then the decision tree can choose any number of the categories to become another branch. If the data is 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 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

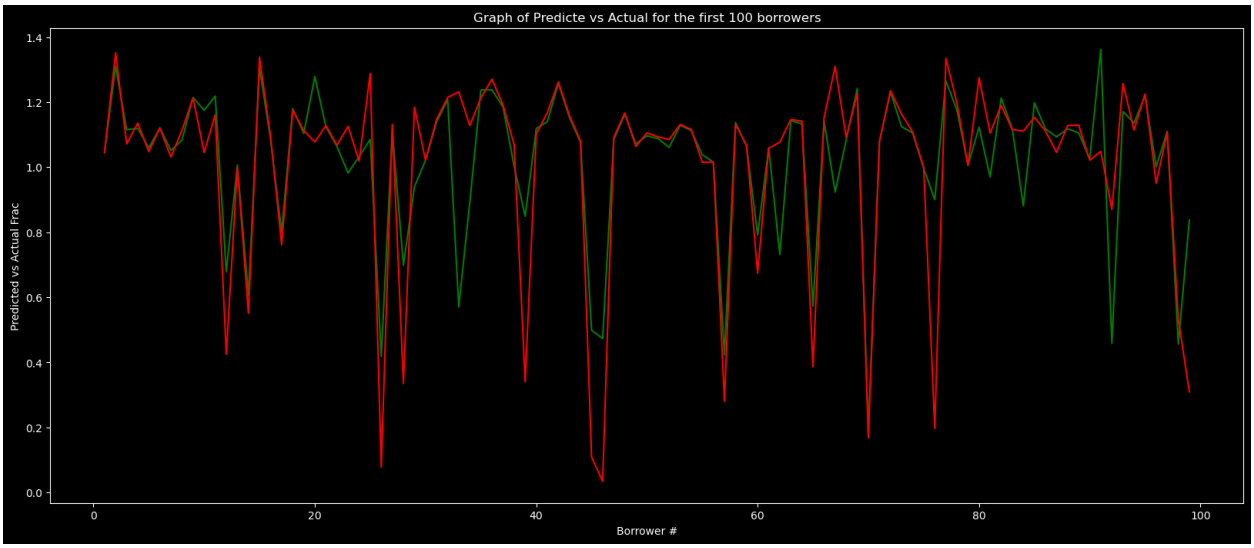
The only hyper parameter that I had time to test was the number of decision trees because the calculations took significantly more time as the number estimators increased. It is possible that

there are some hyper-parameters that would have produced significantly better results, but I did not have the time to find them."[10]

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

Testing

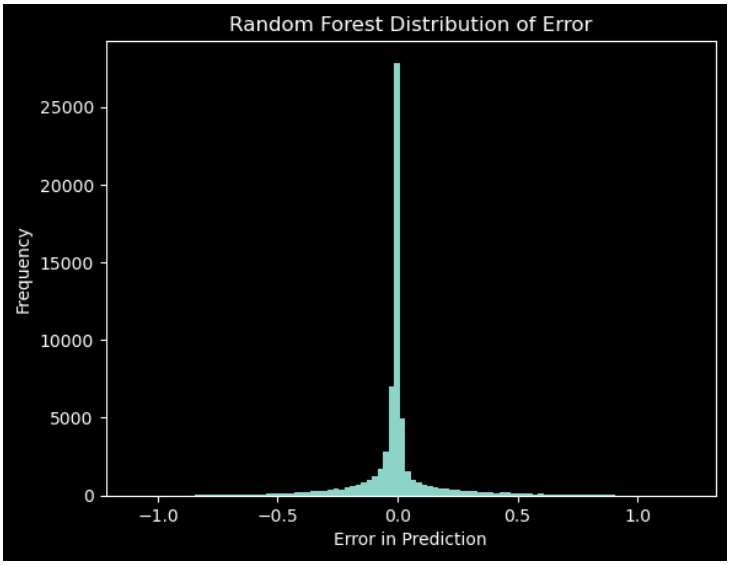
The Random Forest model had a MSE = 0.0309, RMSE = 0.1758, MAE = 0.0872 and a Mean Percent Error of 13.6849%. The discrepancy between the RMSE and the MAE shows that there are a lot of outliers. Below is a representation of the first 100 loans with the predictions in green and the actual values in red.



What is interesting is that 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

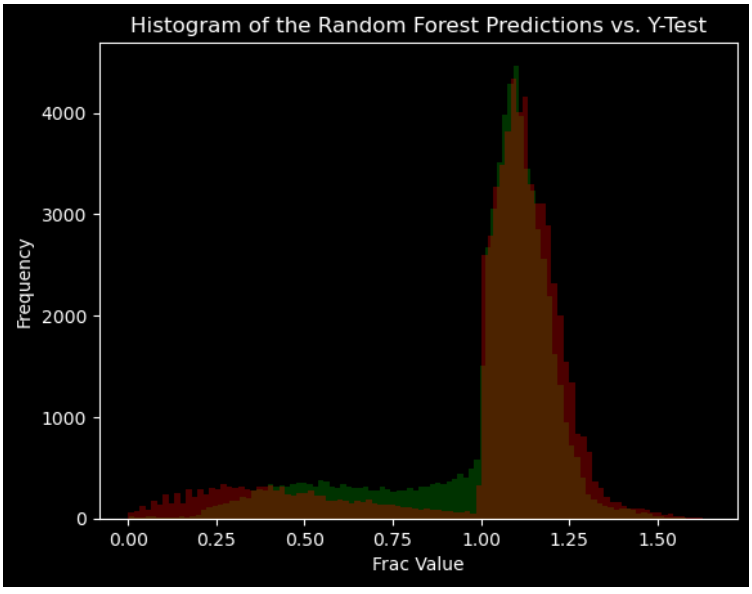
[10] Taken from the Final Report on Network Intrusions

correct but some of the wrong estimates are severely wrong. Below is a graph of the distribution of error for the random forest.



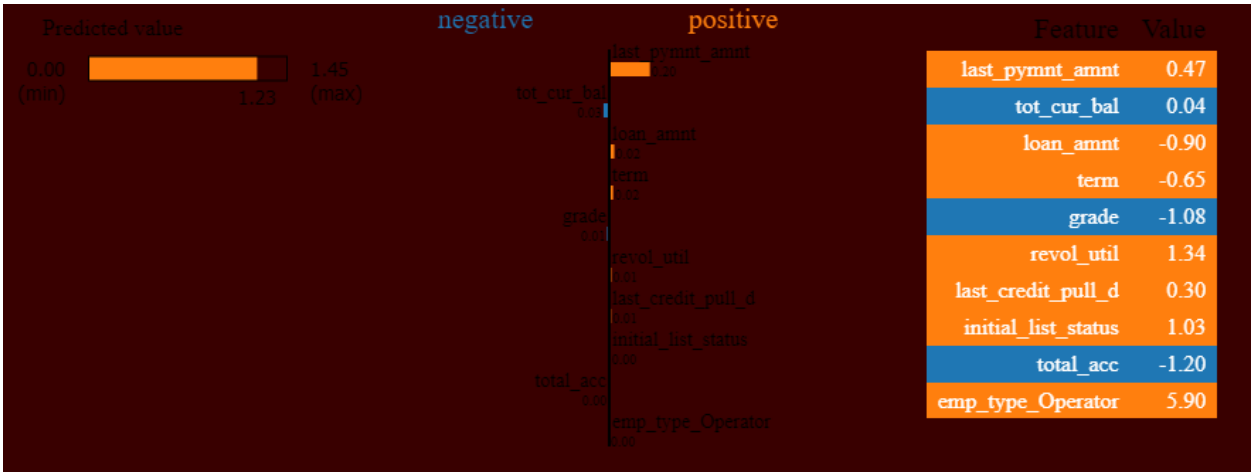
It is stark how much more concentrated all the values are around zero but the outliers are still there.

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 distribution of values over one is interesting. If this had been a classification model where it was trying to Identify which loans will be paid back or not, it would score very highly.

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



Interestingly 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 give an explanation of 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. I’m surprised that it beat out the neural network but there was no depth limit for the Random Forest and it had 600 estimators. It is possible that the Neural Network could have been improved with a lot more layers.

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

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