The Data

LendingClub is a US peer-to-peer lending company, headquartered in San Francisco, California. It was the first peer-to-peer lender to register its offerings as securities with the Securities and Exchange Commission (SEC), and to offer loan trading on a secondary market. LendingClub is the world's largest peer-to-peer lending platform.

Our Goal

I have historical data on loans given out with information on whether or not the borrower defaulted (charge off). My data consists of 396,030 entries. I'll build a model that can predict whether or not a borrower will pay back their loan. This way in the future when I get a new potential customer we can assess whether or not they are likely to pay back the loan. The "loan_status" column contains our label.

Section 1: Exploratory Data Analysis (EDA)

Goal is here to have an understanding for which variables are important, view summary statistics, and visualize the data.

Count plot - Loan Status

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. As we see in the count plot graph, our label column is imbalanced. It will definitely affect accuracy of our model's predictions on 'Charged Off' loans.

Distribution plot - Loan Amount

First thing we noticed in a distplot graph is that spikes happens at the even money amounts.

Heatmap - Correlation

This is our heat map. We can see various relationships between the features. There is almost perfect correlation between "loan_amnt" and "installment" features. I will explore these features further to understand whether this relationship makes sense or not.

Scatter plot - Installment vs Loan Amount

It pretty much makes sense that the installments and the actual loan amount would be extremely correlated. If you loan someone out one million dollars, you would expect that following some formula your monthly payment installments are going to be quite high. We can easily see in the scatter plot that there is a strong positive linear relationship between these variables.

Box plot - Loan Status vs Loan Amount

A box plot shows the relationship between the loan status and the loan amount. In general, it looks like they are pretty similar. If our loan amount is higher, we have a slight increase in the likelihood of it being charged off. It is harder to pay back larger loans than smaller loans. But they're extremely similar in the plot so this is not a key indicator of whether or not someone is going to pay off their loan.

Count plot - Grade and Subgrade

I explored the Grade and SubGrade columns to understand what the unique possible grades and subgrades. I visualized it using count plots.

This is showing the percentage of charged off loans. It looks like it's increasing as the letter grade gets lower. Best borrowers are given a grade of A. Giving loans to the people with lower grades is clearly the riskier than giving loans to the people with higher grades.

Count plot - Only G and F grades

I zoomed in on the little section of the plot that shows G and F subgrades to investigate if it’s worth to give loans to the people with subgrades of G and F.

It looks like F and G subgrades don't get paid back that often. I will isolate those and recreate the countplot just for those subgrades. We notice that if somebody is graded G5, the likelihood of fully paying off their loan is almost as same as being charged off on the loan.

Bar plot correlation between loan status and other numerical variables

I created a new column and named this as loan_repaid. I encoded the categorical values in our response variable (label column) using map() function and put these numeric values into newly created column (loan_repaid). After I created loan_repaid column, we used this column to create a bar plot showing the correlation of the numeric features to the new loan_repaid column.

We can see that interest rate has the highest (negative) correlation with whether or not someone’s going to repay their loan. If somebody has a extremely high interest rate, they are going to find it harder to pay off that loan.

Section 2: Data PreProcessing

Goal is here to remove or fill any missing data and remove unnecessary or repetitive features. Also, I will convert categorical string features to dummy variables.

Missing Data

I created a Series that displays the total count and percentage of missing values per column.

I examined emp_title and emp_length to see if it will be okay to drop them. There are too many unique job titles (173,105) to try to convert this column to a dummy variable feature. I removed that emt_title column.

I created a count plot of the emp_length feature column. I plotted out the count plot with a hue separating it as Fully Paid vs Charged Off.

This still didn't really inform us if there is a strong relationship between employment length and being charged off. I wanted to see the percentage of charge offs per category. It essentially informs us what percent of people per employment category didn't pay back their loan.

Bar plot - Percentage of Charged Off vs Fully Paid (Employment Length)

All the bars are almost the same height in the bar plot. There is not that much information or differentiation between employment lengths. I noticed that Charge Off rates are extremely similar across all employment lengths so I dropped the emp_length column.

I reviewed the title column vs the purpose column to see if this was repeated information. After I checked the entries in both column, I noticed that both column has basically same information so it makes sense to just drop the title column.

Imputation

After I dropped three categorical features which have missing values, there were only three numerical features left which have missing values are mort_acc, pub_rec_bankruptcies and revol_util. There are many different approaches to deal with missing values. We can delete the entries with missing values or we can fill in the missing values with calculated mean value using another column which correlates with the column with missing values or we can use built in imputation algorithm in Python such as SimpleImputer, IterativeImputer and so on. We can drop the variable if it has more than 50% missing values because imputation would not be reliable in those cases.

A better strategy is to impute the missing values, to infer them from the known part of the data. I used the IterativeImputer which models each feature with missing values as a function of other features, and uses that estimate for imputation. In the simplest form, we can say IterativeImputer examine other features which don't have missing values, and then predict the missing values using other features' information.

Categorical Variables and Dummy Variables

After I was done working with the missing data, I just needed to deal with the string values due to the categorical columns.

I listed all the columns that are not numerical then went through all the string features to see what I should do with them.

I converted the term feature into either a 36 or 60 integer numeric data type using apply() function.

I already knew grade was part of sub_grade so I just dropped the grade feature. I converted the sub_grade, verification_status, application_type, initial_list_status, home_ownership and purpose features into dummy variables then concatenate these new columns to the original dataframe.

I created a new column called ‘zip_code’ that extracts the zip code from the address column then converted this new column into dummy variables.

I dropped the issue_d column because this would cause data leakage. I wouldn’t know beforehand whether or not a loan would be issued when using our model.

Earliest_cr_line appears to be a historical time stamp feature. I extracted the year from this feature using apply() function, then converted it to a numeric feature.

Train Test Split

I imported train_test_split from Scikit-learn. I dropped the loan_status column (label) since loan_repaid column is a duplicate of loan_status that we created earlier. Loan_repaid is already a numeric feature so we won’t need to convert it into dummies. I set X and y variables to the .values of the features and label.

I created my model using entire dataset (396,030 entries) but I added a line of code so you can use 0.1 or another percentage of it to train the model. It will take considerable time to train entire dataset.

I performed a train/test split with test_size = 0.2.

Normalizing the Data

As my data is not normally distributed, I used **MinMaxScaler** to normalize the data. I don’t want data leakage from the test set so I only fit on the X_train data. MinMaxScaler is used when data is not normally distributed. It responds well when standard deviation is small. MinMaxScaler subtracts the minimum value in the feature and then divides by the range. The range is the difference between the original maximum and original minimum. The default range for the feature returned by MinMaxScaler is 0 to 1.

Creating the Model

I used feed forward neural network to create my model. I built sequential model that will be trained on the data. The model consists of input layer, three hidden layers and output layer. Activation function for the input layer and hidden layers will be **'relu'** (rectified linear unit). It has been found to have very good performance, especially when dealing with the issue of vanishing gradient. It is often used as the default due to its overall good performance. Activation function for the output layer will be **'sigmoid'** because this is a binary classification problem. Each neuron will output a value between 0 and 1, indicating the probability of having that class assigned to it. The **'cross entropy'** loss function is used for classification. This cross entropy loss function assumes that our model predicts a probability distribution for each class. As I have binary classification, I will use **'binary cross entropy'**. I have different options for optimizer but I use **'adam'** optimizer. Adam optimizer is an adaptive learning rate optimization algorithm that's been designed for training deep neural networks.

I also pass in accuracy metrics in our model to see our accuracy during the training. I added **Dropout** layers and **Early stopping** into the model to prevent possible overfitting. I used **early stopping callbacks** based on our validation loss before it gets out of hand. Keras can automatically stop training based on a loss condition on the validation data passed during the model.fit() call. Dropout **Dropout** is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or *dropped out.* This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer.

Monitor is the quantity to be monitored, and that's validation loss here. In '**min'** mode, training will stop when the quantity monitored has stopped decreasing. **Patience** is the number of epochs with no improvement after which training will be stopped. **Epoch** is an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation. One epoch means we've gone through the entire dataset one time. To explore whether or not we are overfitting the training data, I passed in the **validation data** along for training. It means after each epoch of training on the training data will quickly run the test data and check our loss and accuracy on the test data so that way we can keep a tracking of how well performing not just on our training data but also on our test data. Based on that performance we go back and adjust our model. We repeated the adjustment process over and over again with adding more neurons or layers until we are satisfied. This test data will not actually affect the weights or biases of our network.

I created a grid search to optimize hyperparameters using itertools module. I try to find best hyperparameters for batch size, dropout, learning rate, patience and hidden layers' number of neurons. I received 89% as best accuracy.

Section 3: Evaluating Model Performance

We can actually take a look at our training history using model.history.history. I passed this into a data frame. I plot out loss, accuracy, validation loss and validation accuracy using the validation data that I passed in during the training.

This exactly the kind of plot that I want to see both validation loss and loss decrease.

Now the model has been fit and trained on the training data. The model is ready to predict labels on the test set. I got predicted values using our finalized classification model in Keras using the predict_classes(). This function is only available for Sequential models. I passed in our test data those are our test features and then I got predictions off of the test data.

I evaluated our model by comparing our predictions to the correct values. Typically in any classification task our model can only achieve two results. Either our model is correct in its prediction or our model is incorrect in its prediction. Once I had the model's predictions from the X_test data, I compared it to the true y values (the correct labels). At the end I have a count of correct matches and a count of incorrect matches. I organized our predicted values compared to the real values in a confusion matrix. The key classification metrics I needed to understand are: Accuracy, Recall, Precision, F1-Score. **Accuracy** in classification problems is the number of correct predictions made by the model divided by the total number of predictions. Accuracy is useful when target classes are well balanced. Our target column is not well balanced. **Recall** is the ability of a model to find all the relevant cases within a dataset. The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives. **Precision** is the ability of a classification model to identify only the relevant data points. **Precision** is defined as the number of true positives divided by the number of true positives plus the number of false positives. While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant. The **F1-score** is the harmonic mean of precision and recall taking both metrics into account in the following equation. We use the harmonic mean instead of a simple average because it punishes extreme values.

At the beginning of my report, I mentioned about having imbalanced labels. I had a lot less instances of 0 (charged off). There is a lot more full paid loans than charged off loans. We have much better prediction accuracy on full paid loans than charged off loans. We can easily understand it looking at Recall in classification report.

Next thing, I can try to create a model using borrowers who only have G or F grades. Because I would have more balanced label column.