Benjamin Spiegel

9/22/2019

Capstone 1 / Final Project (Capstone Report)

**Executive Summary**

Using a dataset of demographic and credit card history the project code predicts the probability of default for customers in a month that is preceded by 6 months for which there is account history data available. The theoretical client is the credit card company. Predicting the likelihood of client default allows them to make financial decisions on customers such as cancelling credit cards, increasing rates on identified high risk customers, and sending customers to collections.

In order to solve this problem, I first used graphical and statistical analysis to understand how different variables were predictive of default. I then used Scikit-Learn’s machine learning algorithms to make predictions on individual customer’s default. The most effective algorithm used for the dataset was GradientBoost. The initial prediction of GradientBoost gave a recall of 35% paired with a precision of 70%. Unsatisfied that 65% of defaulters were false negatives and were predicted to make payments, I decreased the threshold for a borrower’s risk profile to be predicted as a default. In one modification, any borrower with a predicted probability of default over 30% was predicted as a default. Using the GradientBoost algorithm gave me a recall of 55% and a precision of 57%.

Given the nature of risk management and the particularly high costs of charge-offs to lenders this seems like a more reasonable result. After the change, over 50% of individuals who will default are correctly predicted. Also, a precision of 57% is a decent result as it means most of the predicted defaulters still actually defaulted. Furthermore, at the new threshold only 12.2% of non-defaulters were incorrectly predicted to default. This is not highly concerning.

|  |  |  |
| --- | --- | --- |
| **Machine Learning Model** | **Precision** | **Recall** |
| Gradient Boost predict\_proba>=.3 | .57 | .55 |
| Gradient Boost | .70 | .35 |

**Description of Dataset**

In the data wrangling portion of my credit card default prediction capstone project. I simply downloaded the csv file ‘UCI\_Credit\_Card’ and ran pandas’ ‘read\_csv’ on the comma separated file to turn it into a pandas DataFrame on the Jupyter notebook file.

The dataset in the file already had its categorical information represented with integers, it also did not have any issues with NAN values. However, in order to maximize the outcome of the capstone project, I added several additional features based on putting together information from multiple features.

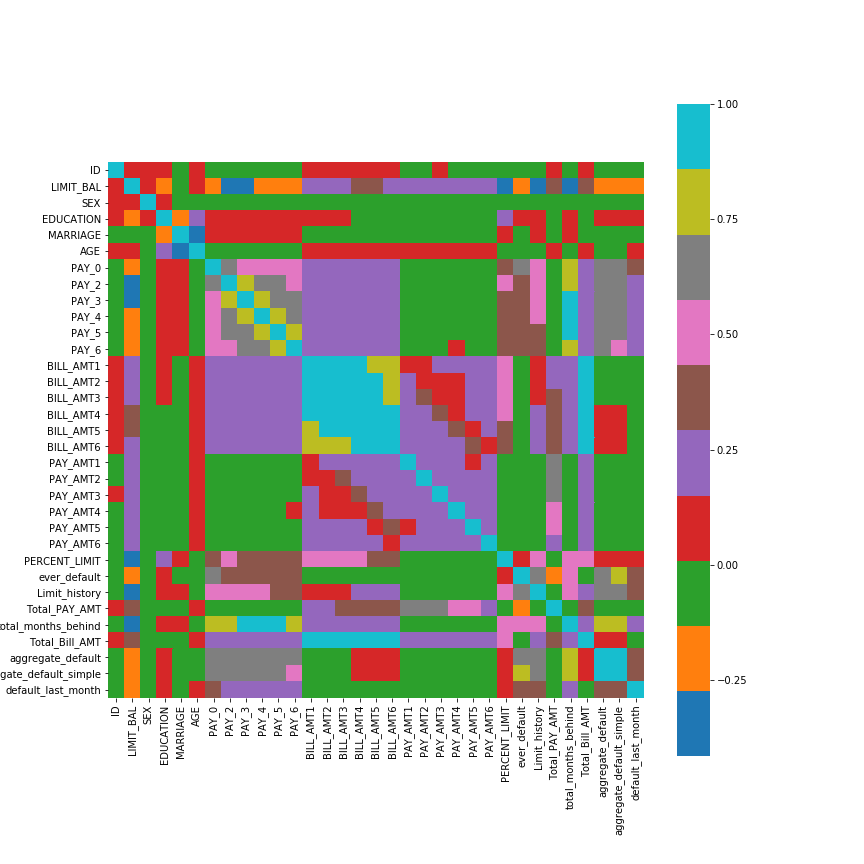
In general, my goal in adding features was to create simpler representations of the relationships between multiple features. One instance of feature creation looked over the 6-month default history of each client and creating a column that simply told whether or not the client had any history of default.  In order to do this, I created a smaller data frame with only the six columns representing each month’s default history. I than used the NumPy function ‘any’ on each row (representing 1 credit card customer) which gave a Boolean value of ‘True’ if the row had any positive numbers (denoting that the customer had defaulted) and ‘False’ if the customer had not defaulted. In order to allow the resulting column to be used by sickie learn I turned the Boolean values into integers by applying the following code:

|  |
| --- |
| DataFrame['Column'].astype('int’) |

In addition, I created a cumulative representation of the number of months that people had defaulted.  In order to do this, I used the six columns representing each month’s default history and mapped out all negative numbers. I did this because the negative numbers represented paying off the entire balance or no consumption. If the negative numbers had been included, they would have negated months of default, which would not be a good cumulative representation.

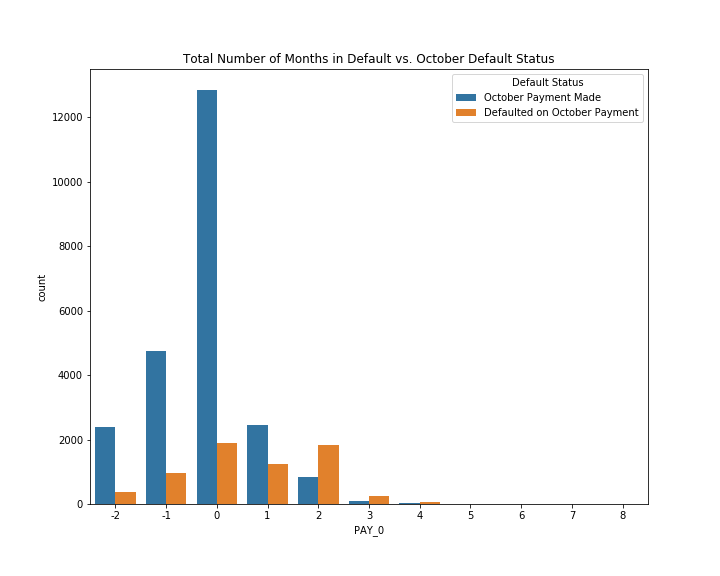
Initial Findings from Exploratory Analysis

The first visual aid I created was a Heatmap. I did this because with one piece of code and one illustration I got a baseline understanding of the correlation between all the different features and likelihood of default. I then knew which features and types of features were most useful and could make assumptions of cause and effect.



Of the feature in the original dataset, ‘PAY\_0’ was the most predictive. This feature tells whether or not an individual made the minimum payment on his/ her next to last bill. If an individual did not make a payment a positive number representing the number of months, they are behind is given. The only scheduled payment that comes after the one ‘PAY\_0’ describes is the payment we are predicting likelihood of default for. That this feature would be particularly predictive is not too surprising. It is showing the most recent payment behavior of an individual and most recent events are typically more predictive than events that happened further away in time. We do have the same information in the same format for the preceding 6 months, and while they are not as predictive, they all have strong predictive value.

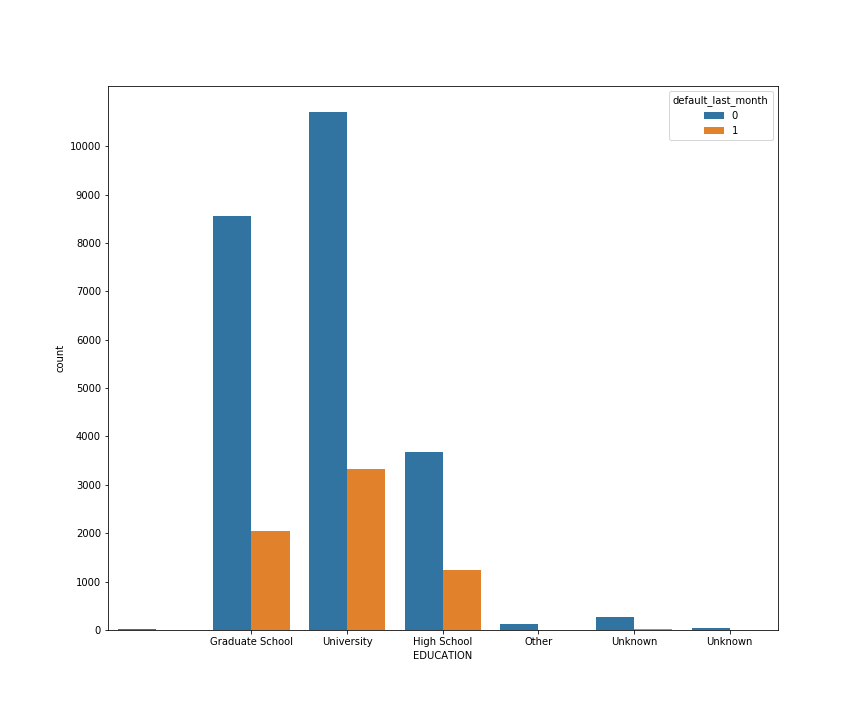
In the bar chart below, you can see that the more positive the value of PAY\_0 the more likely individuals are to default on a payment the following month. This is fairly intuitive since positive numbers represents how many months behind the payor is.



The original data feature ‘Limit\_BAL’ was also strongly correlated. This feature tells the credit limit that the credit card company has offered to each individual. Credit card companies are willing to give higher credit limits to individuals who are more capable of paying their debt and have a positive credit history. People with higher limits were found to be less likely to default in the dataset.

In order to get the best quality predictions, I created new features from relationships between existing features to demonstrate meaningful data that the algorithms could make use of. A couple of these features were shown by the heatmap and other graphs to be quite predictive of whether customers would default. Of the features that I created, the most predictive were ‘ever\_default’ and ‘aggregate\_default\_simple’. The feature ‘ever\_default’ simply demonstrates whether a consumer has ever defaulted in the past. ‘aggregate\_default\_simple’ is a value equal to the total number of months defaulted per each individual over the 6 months of data. Both values were shown to have strong predictive correlation’s with default, as shown by visualizations such as the heatmap and the RandomForestClassifier’s feature importance values.

Looking at demographic features there were certain features that were correlated with higher or lower likelihood of default. These features are helpful because as demographic features they are independent from financial history, and their information is not repetitive with other features. The demographic information in the dataset included age, gender, education, and marital status. Educational level had some correlation with default. Looking at education you can see certain groups are more likely to default.



In order to analyze the correlation between categorical groups and likelihood of default I used seaborn’s Countplot. I also used the pandas’ groupby function to show the percentage of default for by borrower’s highest attainment of education. This is the specific code used:

|  |
| --- |
| df2.groupby(['EDUCATION’]).default\_last\_month.value\_counts(normalize=True) |

Using the code shown above, Individuals with Graduate Degrees, College Degrees, and Highschool Degrees were shown to have a likelihood of default of 19%, 24%, and 25%, respectively. Interestingly people classified as other, while a small group, had a much lower likelihood of default with only a five to six percent likelihood of defaulting. I used the same tools to show the correlation between gender and likelihood of default. The results showed males had a 24.2% likelihood to default in the final month, while females had 20.8 percent likelihood to default in the final month.

Finally using the same groupby methods shown above I looked at how marriage effects default rate. For the final month of data, married people had a 23.5 % chance of defaulting, while single individuals had a 20.9 % chance of defaulting.

**Results and Machine Learning In-Depth Analysis**

Since the question I am trying to solve has binary variables; credit card users will either make a payment in the last month of the data or they will not. It was necessary to choose Machine Learning Algorithms made for categorical predictions.

In order to execute the algorithm on the data I separated the data into two sets.  I created the DataFrame ‘X’ which was included all the features but the label being predicted, and I created the series ‘Y’ which tells whether or not individuals made payments on the credit card in the final month. From the X category I also removed the ‘ID’ feature because it was only purpose was as an index value. Leaving the ‘ID’ feature in the data could negatively affect the model with its random noise.

I initially used the KNeighborsClassifier model from sci-kit learn. The results were quite underwhelming. To determine the best hyperparameters I used GridSearchCV with five cross folds. The model found that the best number of neighbors to consider in designating each individual outcome was 11.

After fitting the model to the training set and predicting the test data, the model gave a score of .768. Indicating that the model was accurate nearly 77% of the time. This number is misleadingly high and poor representation of the algorithm’s ineffectiveness. This is because 78% of the credit card users did not default in the last month. If the algorithm simply predicted all users did not default it would be correct 78% of the time. This is not far from what actually happened as the algorithm predicted non-default 94.7% if the time.

Precision and Recall are better indicators of the model’s effectiveness. The model gave a recall of 11% and a precision of 42. This means, for predicted defaults only 42% actually defaulted, all other predictions were false positive. Additionally, looking at recall, of all individuals that actually defaulted only 11% were predicted to default. This model is essentially useless.

After running KNeighbors, I used Scikit-learn’s random forest model. This performed significantly better than KNeighborsClassifier. Random Forest’s initial prediction had a 35% Recall and 70% precision. This indicates, of those who would default in the last month, 35% were predicted to default. On the other hand, precision was 70%, meaning of those predicted to default, 70% in fact did default.   Additionally, my accuracy score was 82% when using only the test data created by train\_test\_split. I verified this score by using cross\_val\_score and also got an accuracy sore of 82%.

In the initial results, the algorithm had a precision of 70% and a recall of 35%. Interpreting these numbers, it was quite accurate for individuals it predicted to default (precision). The algorithm predicted only 207 false positives while the other 474 individuals predicted to default did default. However, the amount of false negative were quite high as a proportion of actual defaulters. In total, 879 of the 1,353 individuals who defaulted were predicted to make payments.

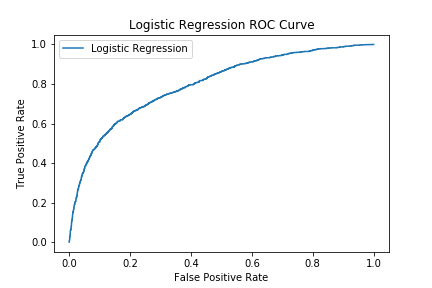
Considering this, The algorithm’s results were modified to sacrifice some precision accuracy for improved recall accuracy. This was done by lowering the threshold for risk profiles to be predicted as defaults. This essentially creates a wider net with whom is considered a risk. Decreasing the model’s precision in order to increase it’s recall may be preferable because defaults are expensive to credit card companies. If casting a wider net with whom is considered a risk can decrease defaults than it may be worth it, even with a higher false positive ratio. In order to decrease the predicted probability of default at which individuals would be predicted to default, I began by using the method ‘predict-proba’, which gave me the algorithm’s predicted probability of default for each borrower in the final month.

|  |
| --- |
| y\_proba = rf.predict\_proba(X\_test) |

With a given probability of default we can then apply a new threshold to predicted probably. In this case we set the threshold to .3 so any probability of default equal or higher results in categorizing customers as predicted default.

|  |
| --- |
| y\_pred\_by\_proba\_30 = (y\_proba[:,1] >=.3).astype('int') |

After changing the probability threshold to .3, we look at the changes to our results. By using this method, recall and precision change from 37% and 69% to 55% and 57% respectively. This means that after changing the probability threshold the majority of actual defaults were correctly predicted, and the majority of predicted defaults still actually defaulted. Depending on the scenario and the cost to the lender of default vs the lender’s profit per non-default one could more precisely determine the optimal default probability to set threshold at. The relationship between how the model’s true positive rate and false positive are affected by changing probability threshold is shown in the plot below.



In addition to running RandomForestClassifier, I also ran Scikit-learn’s AdaboostClassifier and GradientBoostingClassifier. Adaboost performed the worst of all three of the models. It had a precision of .68 and a recall of .33 with its default parameters.

On the other hand, GradientBoostClassifier had a precision of .70 and recall of .35, identical to that of RandomForestClassifier. The difference in performance was so close it was essentially indistinguishable. The Random Forest Algorithm had an average cross validation score of .8206, and GradientBoost Classifier had an average cross validation score of .8215. Indistinguishable again.

The greatest difference for the two models occurred when using the function predict\_proba and a threshold of .30 for default classification. At this probability threshold RandomForest had a recall of .55 and a precision of .54. Gradient boost boost classifier had an identical recall of .55 but with a precision of .57. Therefore, while both models predicted actual defaults at the same rate, our gradient boosting algorithm predicted significantly less false positive (less compliant individuals were predicted as defaults). This indicates our Gradient boosting classifier performed slightly better at this threshold.

|  |  |  |
| --- | --- | --- |
| **Machine Learning Model** | **precision** | **Recall** |
| Random Forest | .70 | .35 |
| Random Forest predict\_proba>= .3 | .54 | .55 |
| AdaBoost | .68 | .33 |
| Gradient Boost predict\_proba>=.3 | .57 | .55 |
| Gradient Boost | .70 | .35 |

Of note, I used scikit learn’s GridSearchCV to hyper-parameterize both the Random Forest and GradientBoost model. Any results concerning either of these two models were run with their parameters optimized. Because AdaBoost clearly performed worse, there was no reason for it’s optimization.

**Conclusions**

This project proved successful at predicting credit card default on a monthly basis. Through running multiple different machine learning algorithms, modifying the default probability threshold and optimizing hyper-paramaters, the best performing model correctly predicted 55% of defaults and 88% of non-defaults.

Retrospectively there were shortcomings of the data. Data that extended over a greater length of time would allow more long term conclusions. Specifically, we do not know the proportion or which specific accounts that had high risk profiles ended up as bad debt that was written off by the lender. Having this information would allow for projections as to when and at what probability of default it would be best to take actions to avoid further risk, as well as what the potential losses from defaulting accounts and saving from cancelling or modifying the credit card terms would be.