2021 Data Science Competition

Predicting Credit Default

Bryce Davis, Ted Woodsides, Taylor Last, Anderson Molter

Department of Statistics
University of Georgia

April 15, 2021

Contents

List of Figures

List of Tables

1 Introduction

Financial Institutions that lend money to consumers need a method to decide whether to approve or decline an individual for a loan. For the institution, giving a loan is an investment in the consumer. These institutions need a way to predict whether or not the consumer is a good investment for them- i.e will they pay back the loan in the agreed upon time frame. The time allotted to make payments varies by bank, but a common guideline is if the consumer does not make three consecutive monthly payments on their loan they are considered "defaulted". When this happens the consumer is sent to collections and the financial institutions have to spend time and energy to receive any sort of return on their investment, but even this method is not guaranteed to cover their loss. To avoid this inconvenience, institutions started using statistical models that use the financial history of a consumer to decide whether they are a good candidate for a loan. The purpose of this report is to model and compare two binary classification techniques to predict if a consumer will default on their loan within 18 months of opening an account. The methods of focus are logistic regression and gradient boosting, a form of decision tree classification.

2 Data and Pre-processing

For our analysis we utilize 3 data sets:

1. A training data set with 20,000 accounts
2. A validation data set with 3,000 accounts
3. A testing data set with 5,000 accounts

Each data set has a response variable (1 if the account defaulted, 0 if not) and 20 predictor variables. We began our initial investigation of the training data set by noting the data type of each variable, and looking into the existence of missing observations. Table 1 shows the data types of the predictor variables and figure 1 shows the distribution of missing data

Table 1: Data Types of Predictor Variables

Data Type	Variable Name
Discrete-Binary	ind_acct_xyz, Default_Ind
Discrete	non_mtg_acc_past_due_12_months_num, non_mtg_acc_past_due_6_months_num, mortgages_past_due_6_months_num, inq_12_month_num, card_inq_24_month_num, card_open_36_month_num, auto_open_.36_month_num
Continuous	tot_credit_debt, avg_card_debt, credit_age, credit_good_age, card_age, credit_past_due_amount, uti_card, uti_50plus_pct, uti_max_credit_line, uti_card_50plus_pct, rep_income

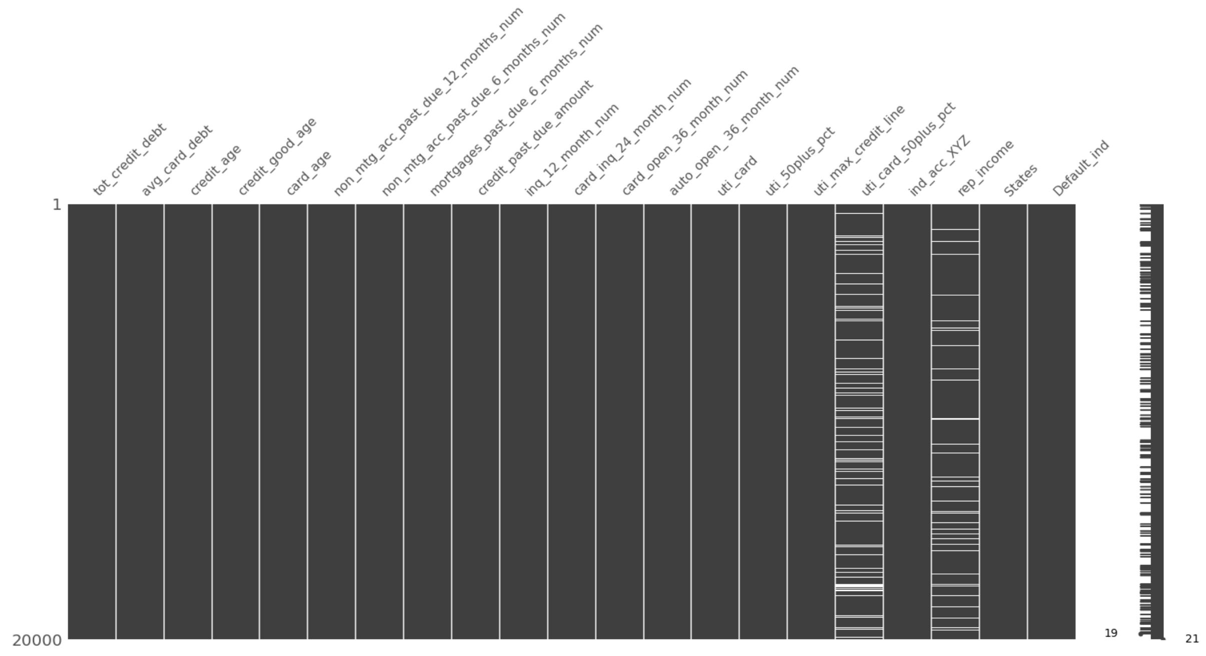


Figure 1: Distribution of Missing Data

2.1 Missing Data

Our data has several missing values in it, but all of those missing values fall under two predictors: *uti_card_50plus_pct* and *rep_income*. Dealing with missing values is a topic that is heavily debated in data science, so we spent a good amount of our time deciding on which method we wanted to use to handle them. We decided to explore the Multiple Imputation by Chained Equations (MICE) method and the K-Nearest Neighbor (KNN) methods. These methods are much more accurate than replacing with the column mean and they provide our model with much more information than simply removing missing values.

2.1.1 MICE

As opposed to single imputations, MICE creates multiple imputations of the missing data to account for the statistical uncertainty in the imputations. To create a single dataset, MICE pools the results of several different multiple regression fits using different values for the missing observations. The most common popular method for MICE imputation is Predictive Mean Matching (PMM). PMM uses multiple regression and outputs a prediction that acts as a prior. Then, the algorithm finds a match in the data that is very close to its prediction and imputes that into the column. All of the imputed data points make up the posterior distribution displayed in the comparative density plots, Figures 2 and 3.

2.1.2 KNN

K-Nearest Neighbor imputation is useful for matching a missing data point with its ”nearest neighbors in a multidimensional dataset where the missing values are sparse. The intuition behind the method is that it is able to match the missing value to other values around it based on other predictors. Therefore, some multicollinearity is also helpful with KNN imputation. The KNN algorithm works by optimizing the amount of k neighbors and weighting each neighbor by some distance measure (Euclidean distance is commonly used). Then, the algorithm uses the weighted means to predict what value to impute in the new dataset.

2.1.3 Comparison of Imputation Methods

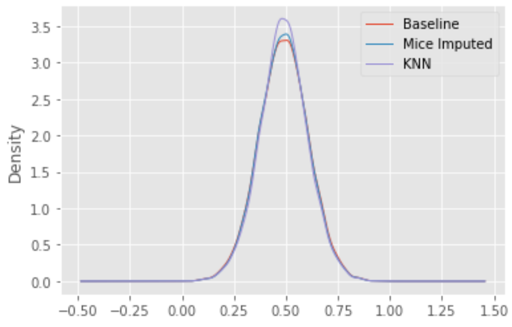


Figure 2: Imputed uti_card

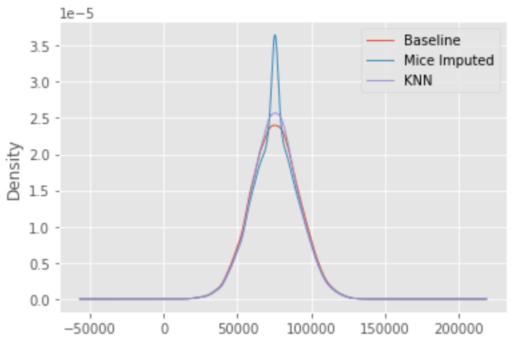


Figure 3: Imputed Rep_income

MICE imputation method fit the distribution more accurately than the KNN method for *uti_card_50plus_pct*, which is shown in Figure 2, while the KNN method fit the distribution more accurately for *rep_income*, which is shown in figure 3. Therefore, we decided to impute *uti_card_50plus_pct* with the MICE imputer and *rep_income* with the KNN imputer.

2.2 Class Imbalance

The datasets we were provided contain a large class imbalance. The amount of observations that default (encoded as 1) is significantly less than the amount of observations that do not default (encoded as 0). There are 18414 observations that are encoded as 0 and 1586 observations encoded as 1.
The class imbalance among the training dataset is visualized in Figure 4. The visualization emphasizes the disparity between the two groups. 92.07% of the data in the training dataset does not default.

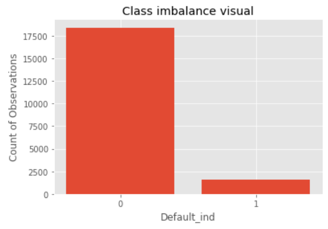


Figure 4: Vizualization of Class Imbalance

To handle the class imbalance, we considered several methods. The first two methods involved changing the size of the training data. We tried re-sampling the observations that default so the default class would be as large as the non-default class. However, this led to a significant decrease in model performance, so we decided to avoid it. We also tried to undersample non-default class; however, we lost too much information in the training data, leading to a decrease in model performance. Instead of changing the training dataset, we decided to change the metric we were evaluating the model on. If we kept accuracy as the model performance metric, we would be receiving an inflated estimate on model fit. If we classified all of the data as non-default, we would obtain an accuracy score of 92.07% due to the class imbalance. Therefore, we decided to explore other metrics based on the confusion matrix. These metrics include recall, precision, F1 score, and AUROC. These methods will be explained further in section 3.1.

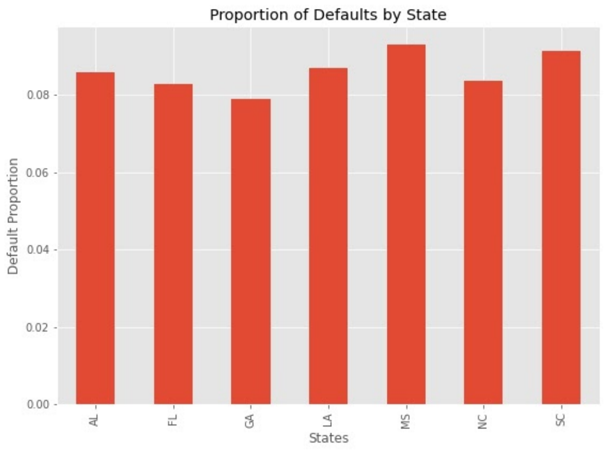


Figure 5: Defaults by State

2.3 Categorical Variables

Our datasets only had one categorical variable, which was the states variable. We explored the differences among the different observations in each state and determined there was not enough between-state variation to justify using it as a dummy variable.

As shown in Figure 5, all of the states have very similar proportions of defaulting. The mean default rate is centered around 0.0862, which is the default rate for the entire data set, and the between-state variance is only 0.00002. Figure 5 and the summary statistics of proportion of defaults by 'States' indicate that it is not an important indicator variable, and therefore, we can exclude it from our model.

2.4 Feature Engineering

A large amount of the predictors are not individually important to predicting whether or not a customer will default. Some of our predictors are heavily correlated with each other, which creates multicollinearity. The more that highly correlated features remain in our model, the higher chance we have at training our model on noise. Removing or manipulating correlated variables can improve model generalization and accuracy. Also, some of the variables include obvious outliers that are highly influential in predicting whether or not a customer defaults.To handle some of the issues we had with the predictors, we combined variables that had little to no effect on the data by themselves, but when combined, they significantly improved the model. Reported income and total credit debt are two predictors we chose to transform due to their lack of correlation with defaulting.

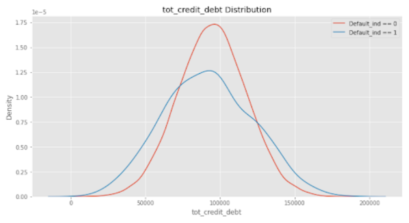


Figure 6: KDE of tot_credit_debt

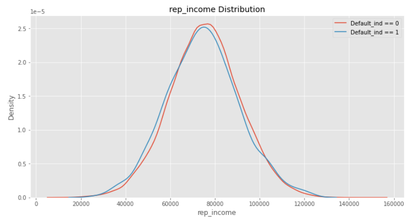


Figure 7: KDE of rep_income

Based on the two distributions comparing each default class to the predictors, we see that *tot_credit_debt* for the non-default class has a mean that is centered around the same mean as the default class, but the variance is significantly less. This could be due to the disparity between sample size. Looking at *rep_income*, we notice that the distributions for both default classes are almost identical. The Kernel density estimation (KDE) plots in Figure 6 and figure 7 display that neither one of these predictors has a large shift depending on the default indication. To deal with these two predictors, we created two new predictors for the data: *income_debt_ratio* and *bad_debt_ratio*. The *income_debt_ratio* is the average card debt divided by reported income, and the *bad_debt_ratio* is the average card debt divided by the total credit debt. The income debt ratio represents how much a customer has in credit card debt (monthly) per dollar of reported income (annually). The bad debt ratio is the proportion of credit card debt to overall debt. We felt these predictors were a good representation of how the customer handles their money. A customer with a large percentage of unsecured debt is generally less financially stable than those who have their debt mainly in stable assets like real estate.

2.5 Outliers

Overall, our dataset is pretty well-behaved after the data imputation methods that removed missing values; however, some of our predictors had anomalies. The predictor we focused on the most was average monthly credit card debt (avg_card_debt).

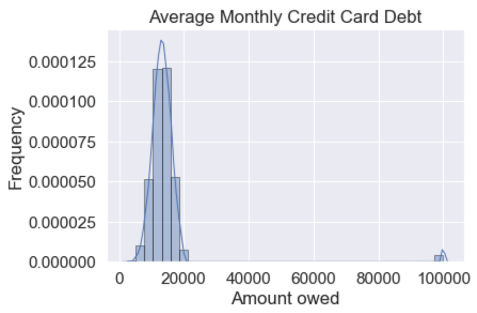


Figure 8: Distribution of Monthly Card Debt

The distribution for average monthly card debt appears to have most of its data in the range of 0 to $20,000$, but there are 212 observations that appear at $99,999$. At first, we thought it could be a mistake, and therefore, we would treat them as outliers. When we investigated further, we found that approximately 45% of the customers with average monthly card debt of $99,999$ defaulted, while only 7.5% of the rest of the observations in the training set default. As a result, we deemed these outliers to be too influential to the model and decided to keep them.

3 Dimension Reduction

3.1 Multicollinearity

When looking at the correlation matrix for our data, we notice that we have multiple groups of strongly correlated with one another. Intuitively, this makes sense because most some of our predictors are the same measure but different time spans.
Looking at Figure 8, we see that credit age, good credit age, and card age are all related with one another. All of the overdue payments on credit, such as *non_mtg_acc_past_due_12_months_num* and *credit_past_due_amount* are all correlated. The debt predictors are correlated with each other and so are the utilization predictors. To deal with the multicollinearity in our data set, we debated using Principal Component Analysis(PCA) for dimension reduction. Although, we believed a Factor analysis would allow for a very intuitive interpretation of these variables while also scaling them and capturing their multicollinearity.

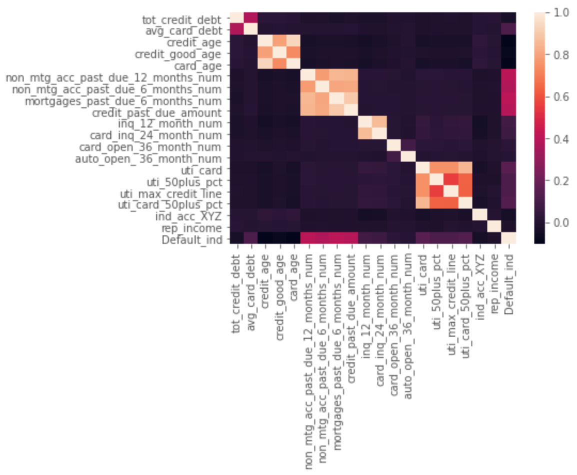


Figure 9: Correlation Heatmap of Original Dataset

3.2 Factor Analysis

Factor Analysis seeks to identify latent, or hidden, variables associated with the observed variables in your dataset. For example, there is no direct measure of human intelligence, like you could measure height or weight. Though, if we had record of ones GPA and SAT scores, we could probably estimate their intelligence as an influential, hidden factor associated with these metrics. Similarly, unlike income or amount of debt, we cannot directly observe ”credit worthiness”. Through factor analysis we hope to estimate the ”hidden” variables that are influential in ones observed credit data. Factor analysis partitions the variance into common and unique variances based on the original data. It allows variance to be spread out among factors as opposed to most of the variance being contained in the first couple principal components. This usually makes factor analysis intuitive to interpret because it will give correlations between factors and the original predictors. In our dataset, chose factor analysis because the factors loadings mapped very nicely onto our predictors.

Factor analysis works by taking the eigenvalues and eigenvectors from the variance-covariance matrix of the training dataset. The vectors are then rotated using a specified method. In our analysis we used ‘varimax’ and ‘minres’, which arranges the vectors orthogonal to each other by minimum residuals or least squares. Once the eigenvalues were set, we looked at a scree plot to determine how many factors to take. Since the first 5 factors accounted for most of the variation we wanted to capture, we selected 5 factors. These factors are then fitted on our training, validation, and test data set to scale and transform them to follow the composition of our factor loadings matrix. The loadings in the factor matrix (figure 10) represent the correlation between the factor and the corresponding predictor.

The factor loading heatmap displays which factors correspond with each predictor (shown

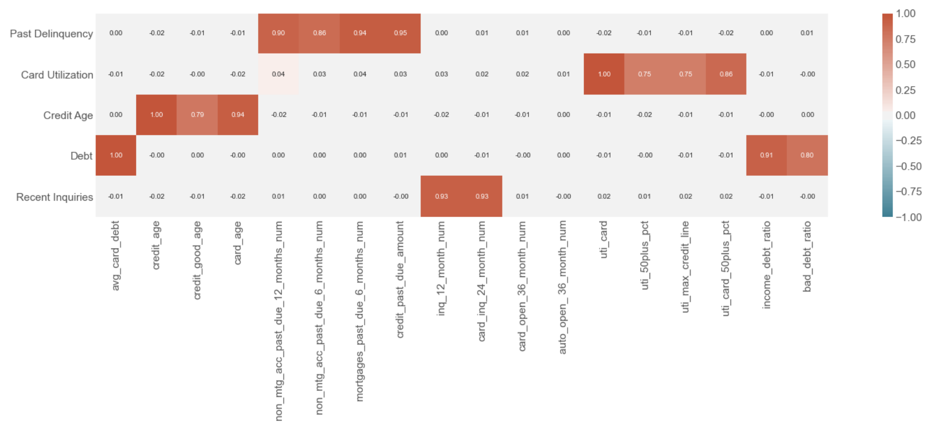


Figure 10: Factor Loading Heatmap

in Figure 10). The higher numbers, represented in orange, show where most of the variation for each factor is coming from. Since we were easily able to group the predictors into five factors, we chose to call the first factor past delinquency, which contains the four predictors that measure the credit past due. We named the second factor card utilization, which contains all of the predictors that show how much a customer utilized their credit. The third factor is credit age, which shows how age predicts defaulting. The fourth factor is debt, which includes all the predictors that have metrics on debt. Finally, the fifth factor is recent inquiries, which corresponds to how many inquiries are being made on behalf of the customer.
These five factors have a intuitive parallel to how credit worthiness is determined in the world today, with credit scores. The FICO Credit score is calculated with five factors: payment history (35%), amounts owed (30%), length of credit history (15%), new credit (10%) and credit mix (10%). Of course, our factors will not perfectly calculate the customer's credit score, but factor analysis has allowed our model to act as a surrogate approximation of something similar. The magnitude of these five factors will determine if we predict a default or not.

4 Logistic Regression Model

After the Factor Analysis was completed we fit the factors on the training data using a logistic regression model. Once this model was fit and we had the prediction output from the model, we assessed the performance of the model by comparing our predicted values to the $Default\_ind$ values (our reference value) in the validation dataset. Our model takes an observation and then outputs a probability of default between 0 and 1. Due to our class imbalance, setting our threshold decision boundary (where values greater than the boundary are labeled "Default") to 0.5 was not optimal. Rather than arbitrarily testing thresholds, we built a function that iterated through a range of possible thresholds from

$.01 - .99$ and for each thresholds sought to maximize metrics that we deemed as good indicators of model performance.

4.1 Performance Metrics

The performance metrics we investigated were F1 Score, area under the receiver operating characteristic curve(AUCROC), precision, and recall. These are all metrics based on the number of true positives, true negatives, false positives, and false negatives. After researching the metrics we decided to use F1 Score because of its robustness for class imbalance (3). It also strikes a good balance between both precision and recall because focusing solely on either would result in lost revenue for the financial institution. The F1 Score is calculated using the following formula:

$$\text{F1} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

With the performance metric selection we ran the function described earlier to find the threshold that maximized our F1 score. The threshold that was selected by the function was .21

4.2 Assumptions

Logistic regression does not have the same linearity, normality, and homoscedasticity assumptions as a linear model; but there are still requirements to meet before doing model inference.

1. The dependent variables must be binary - met by $Default\_ind$
2. The observations must be independent of each other - each person's debt is independent
3. The predictive variables should not show evidence of multicollinearity - this issue is addressed in section 3.1
4. Sufficient Sample Size - met by our training set with $n = 20,000$

4.3 Model Results

After selecting the probability threshold we ran the model on the training dataset. We used the F1 score as well as a confusion matrix to analyze the model success. The maximized F1 score =.48 and the confusion matrix is shown below.

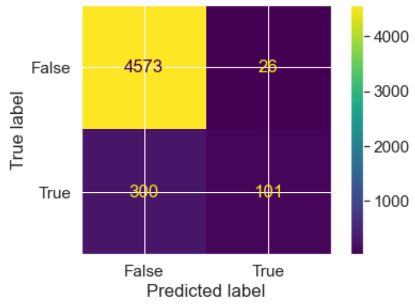
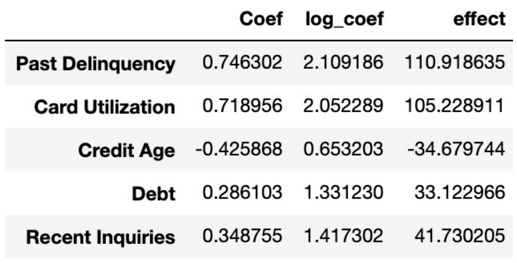


Figure 11: Logistic Regression Confusion Matrix

The confusion matrix shows that we had 4573 True negatives, 300 False negatives, 101 True positives, and 26 False negatives. In this context a positive served as someone who defaulted and a negative is someone who did not. The model only predicted ”Default” 127 times out of 5000.

4.4 Model Interpretation

Figure 12 shows the 5 factors described in section 3.2 that were used in our logistic regression model. The log coef column represents that variables odds ratio. The magnitude of the effect column indicates the overall leverage the variable had on prediction.



	Coef	log_coef	effect
Past Delinquency	0.746302	2.109186	110.918635
Card Utilization	0.718956	2.052289	105.228911
Credit Age	-0.425868	0.653203	-34.679744
Debt	0.286103	1.331230	33.122966
Recent Inquiries	0.348755	1.417302	41.730205

Figure 12: Factor coefficients

It is important to note that these are the factor loading transformed variables, so the interpretation is not 1:1 to the original dataset.

1. Past Delinquency: For every one unit increase in the Past Delinquency factor, the odds that the person defaults are 2.11 times as large as the odds the person does not default when all other variables are held constant.
2. Card Utilization: For every one unit increase in the Card Utilization factor, the odds that the person defaults are 2.5052289 times as large as the odds the person does not default when all other variables are held constant - our most significant factor.

3. Credit Age: For every one unit increase in the Credit Age Factor, the odds that the person defaults are .653203 times as large as the odds the person does not default when all other variables are held constant. This is the only variable negatively correlated with defaulting.

4. Debt: For every one unit increase in the Debt factor, the odds that the person defaults are 1.331230 times as large as the odds the person does not default when all other variables are held constant.

5. Recent Inquiries: For every one unit increase in the Debt factor, the odds that the person defaults are 1.417302 times as large as the odds the person does not default when all other variables are held constant

5 Gradient Boosting Model

XGBoost (1) (Extreme Gradient Boosting) was chosen to fit this particular dataset due to its resiliency to class imbalance. XGBoost has an extensive list hyper-parameters that allow for strong performance even in situations with a high class disparity, unlike random forests. Although, feed-forward neural networks are extremely powerful, they are particularly useful when there is some form of hierarchical structure within the data, such as text or images. For a classification task such as this one a neural network seems a bit overkill especially given the focus on explain-ability. So, XGBoost felt like a clear choice for this dataset. While XGBoost lacks the ease of explanation that classical logistic regression would, it should make up for it in performance and customization.

5.1 Data Pre-Processing

For use in gradient boosting decision trees, the data was fit slightly differently than in the logistic regression model. Firstly, decision trees are inherently scale-invariant. It does not matter if the data is standardized, or normalized; the decision tree will split in the same way. Additionally, bad debt ratio and income-debt ratio were not added to this model. Unlike logistic regression, gradient boosted decision trees are not log-linear models, or any form of linear model. Decision trees are non-linear and can therefore capture non-linear relationships such as income-debt ratio. Factor analysis was not performed on the data used in the XGBoost model. Factor analysis was to aid in our ability to interpret the data in the logistic regression model and would not be of much use in a black-box model because decision trees do not have parameter coefficients. Table 2 displays the final variables used in the XGBoost classifier.

5.2 Hyper Parameter Optimization

As mentioned before, XGBoost has an extensive amount of hyper parameters. This allows XGBoost to be used in a large variety of situations and thus has become a go-to of many

Variable Name	Explanation
past_due_acct_sum	Row-wise sum of past due accounts columns
past_inq_sum	Row-wise sum of past inquiry columns
mean_card_util	Average of all utilization columns
mean_credit_age	Average of all credit age columns
avg_card_debt	Column from original dataset
ind_acc_XYZ	Current bank customer indicator

Table 2: Explanation of XGBoost model variables

Variable Name	Value
base_score	0.5126
gamma	6.1425
learning_rate	0.0408
max_depth	35.3378
min_child_weight	8.5961
reg_alpha	0.0778
reg_lambda	0.0847
scale_pos_weight	21.411

Table 3: Optimal Hyper Parameters

data scientists and Kaggle competition connoisseurs. A common method for optimizing hyper parameters is to use GridSearch. This method specifies a range of possible hyper parameters (a grid) and then finds the best subset of all combinations. While effective, it is computationally a brute force method and is extremely time consuming - especially with XGBoost's large number of possible hyper parameter situations. In contrast, Bayesian optimization, a.k.a Sequential Model-Based Optimization (SMBO), implements hyper parameter optimization by building a probability model of the objective function that maps the parameter input values to a probability of a loss. This surrogate probability model is easier to optimize than the actual objective function. The concept is to limit evaluation of the actual objective function by spending more time choosing the next values to try instead of actually trying them all. Bayesian Reasoning means updating a model based on new evidence, and, with each evaluation, the surrogate is re-calculated to incorporate the latest information(2). Thus, the longer the algorithm runs, the closer the surrogate is to the actual objective function. This was computed using the BayesianOptimization package. In our own testing, GridSearchCV would take 8 hours to fully search the same parameter space that the Bayesian Optimizer searched in just 20 minutes. The Bayesian Optimized hyper parameters are found in table 3.

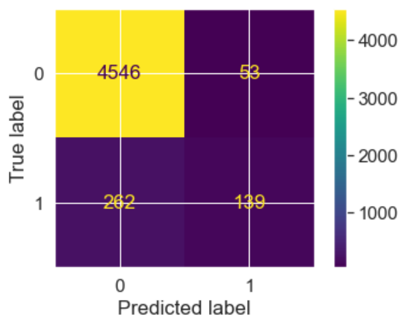


Figure 13: XGBoost Confusion Matrix

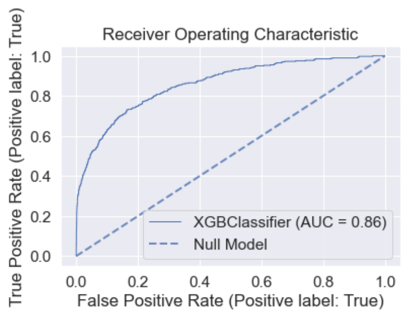


Figure 14: XGBoost ROC

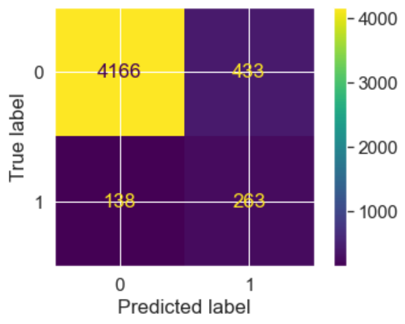


Figure 15: Optimized Confusion Matrix

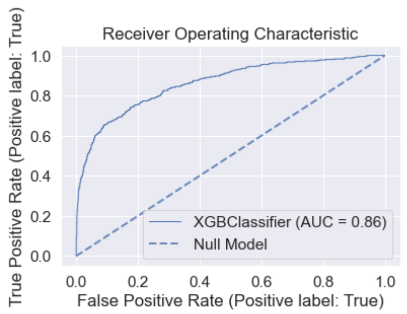


Figure 16: Optimized XGBoost ROC

5.3 Results

The initial results we already strong from the XGBoost fit with no specification of hyper parameters. This can be seen in figures 13 and 14. The original fit was similar to that of our logistic regression model - with a very low false positive rate. Hyper parameters were tuned to fit the ”logloss” evaluation metric. Log Loss heavily penalises classifiers that are confident about an incorrect classification and consequently we see a large shift in our model predictions. The optimized model had a lower f1 score than its original fit with a value of 0.899 and an identical ROC AUC of 0.86. There was a large increase of false positives to 433. The number of false negatives dropped significantly to 138. These are all shown in figures 15 and 16. Overall prediction accuracy was lower in the optimized model than in both logistic regression and the XGBoost default setting. However, we believe this has resulted in a superior modeling method for credit default.

6 Model Choice

Although both methods performed far better than a coin-flip at predicting credit default, we believe the optimized XGBoost model is the best. Accuracy is not everything, especially given our class imbalance. If we just predicted no default every time, we would

already be 93% accurate. In this business setting, other metrics are far more valuable. The XGBoost model was far superior at eliminating false negatives. In the credit context, every false negative means the bank is losing a large amount of money on the default. We want to minimize this as much as possible. The XGBoost model still accepts the majority of people applying for credit but takes a "better safe than sorry" approach. The confusion matrix in figure 15 displays the balance this model takes between restricting access of credit to new customers and reducing the number of mis-classified non-defaults. This is also known as "recall". The disadvantage of XGBoost comes with its "black-box" nature, there is no way to explicitly to discuss parameters like we can in logistic regression. Luckily, packages such as LIME (Local Interpretable Model-Agnostic Explanations) and SHAP(SHapley Additive exPlanations) exist to help alleviate the problem of machine learning interpretation. (Sections 7 and 8)
The process of using this model on future applications is actually very simple. The data transformations are saved as an SK-Learn pipeline, and the model has already been pre-trained. Single or multiple observations can simply be passed into the model.predict() function and its estimated class probabilities are exported. XGBoost automatically molds the decision boundary to 0.5 while simultaneously accounting for class imbalance - negating the need for boundary manipulation. Due to the binary nature of this dataset, observations with a predicted probability greater than 0.5 are labeled as defaulted and vice-versa.

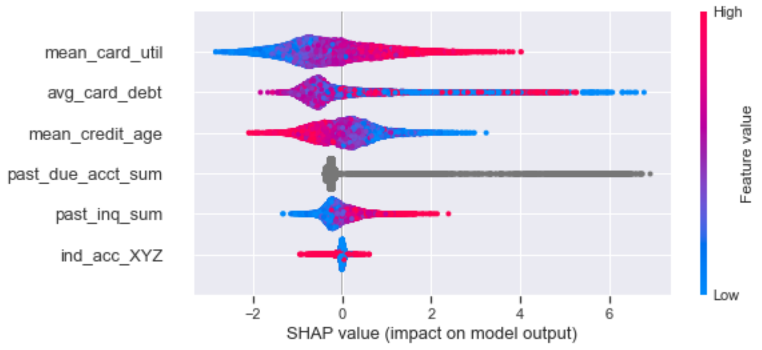


Figure 17: SHAP Summary Plot

7 Favorable Treatment?

One question to consider regarding creditworthiness is whether banks are biased towards giving loans to existing customers, as opposed to new ones. In our data, there is an indicator variable inc_acc_XYZ that tells us whether an applicant already has an account with the bank they are applying at. We chose to use SHAP values to measure the impact of the inc_acc_XYZ variable. SHAP values calculate the importance of a feature by comparing what a model predicts with and without the feature. However, since the order in which a model sees features can affect its predictions, this is done in every possible order, so that the features are fairly compared (4). Figure 17 shows the SHAP summary plot with inc_acc_XYZ compared to other significant variables over the entire model fit. Additionally, the LIME and SHAP explanations in section 8 also display a minuscule influence from inc_acc_XYZ in regards to individual predictions. From these figures we can clearly indicate that inc_acc_XYZ is having little to no affect on model predictions, thus having an existing account with a bank does not effect ones likelihood of getting approved for a loan.

8 Customer Explanation

If a credit card application is rejected through our XGBoost model and a customer demands an explanation we can use the LIME and SHAP packages to display exactly why our ML model made its decision. To demonstrate, we take a customer that was predicted to default on their credit. Their LIME and SHAP explanations are shown in figure 18 and 19. As you can see, this customer was predicted to have a 99% chance of defaulting. Their sum of past due accounts was the main factor in the decision with their credit utilization also weighing heavily. Their credit age was in their favor, but was outweighed by the risk of their past defaults. Both LIME and SHAP are consistent in their explanations. We can provided personalized advice on how to improve a rejected customer's chances of getting accepted for credit in the future and explain to them how their past behavior is influential in automated decisions made by financial institutions.

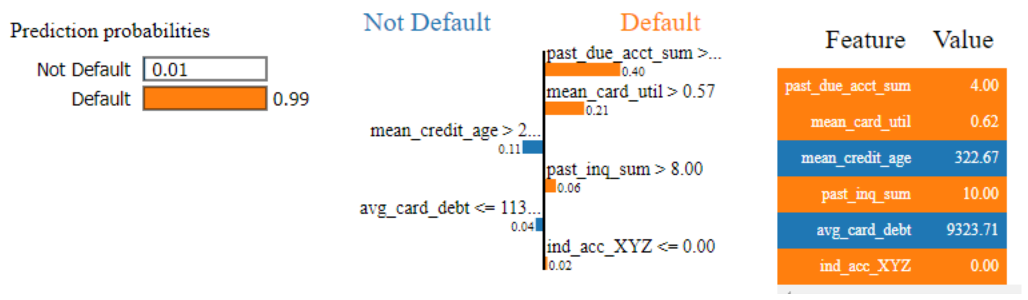


Figure 18: LIME Default Explanation

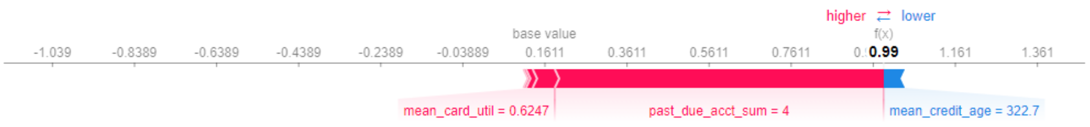


Figure 19: SHAP Default Explanation

References

[1] Chen, Tianqi and Guestrin, Carlos. XGBoost: A Scalable Tree Boosting System Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[2] Ian Dewancker, Michael McCourt and Scott Clark. Bayesian Optimization Primer SIGOPT

[3] Antonio Maratea, Alfredo Petrosino, Mario Manzo. Adjusted F-measure and kernel scaling for imbalanced data learning

[4] Pablo Casas A Gentle Introduction to SHAP values in R

[5] A Unified Approach to Interpreting Model Predictions Scott M. Lundberg, Su-In Lee

[6] Why Should I Trust You?": Explaining the Predictions of Any Classifier Marco Tulio Ribeiro and. Sameer Singh and. Carlos Guestrin Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August

[7] Scikit-learn: Machine Learning in Python Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. Journal of Machine Learning Research, 2011