

ABSTRACT

This report documents our submission for the 2018 Central Florida Educators (CFE) Federal Credit Union Data Competition. We processed CFE vehicle loan data and performed exploratory data analysis (EDA), feature engineering, and classification of the data. EDA included validation, visualization, correlation analysis of the data. Feature engineering included combining features using contextual understanding of the data to potentially increase the predictive power of future models. Classification included model selection, training, and cross-validation testing.

CHAPTER 1: Exploratory Data Analysis

Exploratory data analysis (EDA) is used in data science to gain more insight into the data. Data validation, a crucial component of EDA, ensures statistics of the data are reasonable given the context. In this case, the context is bank loan data. EDA of the Central Florida Educators (CFE) bank loan data includes validation the training data and computing the correlation of the training data. Validation of the CFE data included numerical and visual analysis of the training data.

Numerical Feature Validation

Validation of numerical features was done using visual methods, mainly box plots to determine outliers and histograms to validate numerical feature distributions.

Outliers Outlier detection identifies entries that are much different than the rest of the group, allowing us to determine if the entry is a mistake or a valid entry. Generally speaking, good practice dictates retention of valid outliers in the data set. Box plots are an effective method for determining outliers.

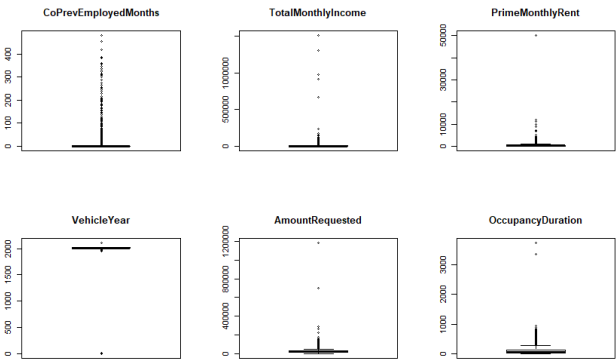


Figure 1.1: Box plots for CoPrevEmployedMonths, TotalMonthlyIncome, PrimeMonthlyRent, VehicleYear, AmountRequested, and OccupancyDuration.

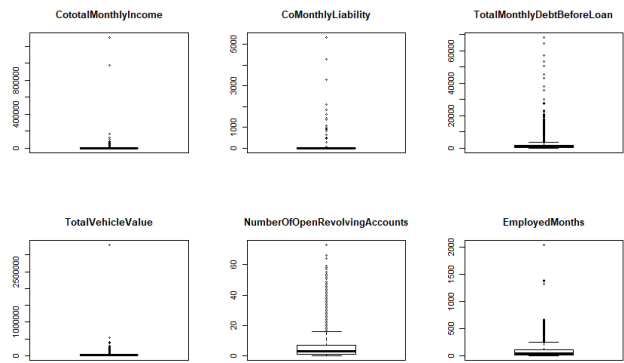


Figure 1.2: Box plots for CototalMonthlyIncome, CoMonthlyLiability, TotalMonthlyDebtBeforeLoan, TotalVehicleValue, NumberOfOpenRevolvingAccounts, and EmployedMonths.

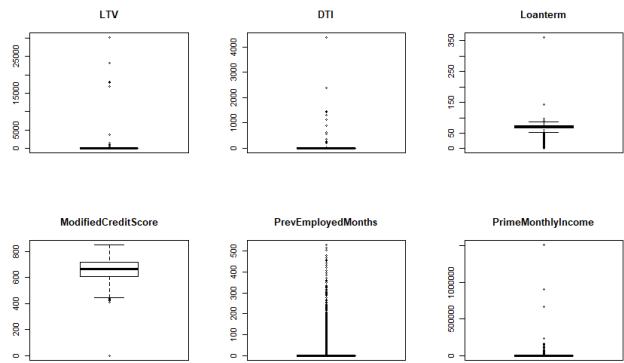


Figure 1.3: Box plots for LTV, DTI, Loanterm, ModifiedCreditScore, PrevEmployedMonths, and PrimeMonthlyIncome.

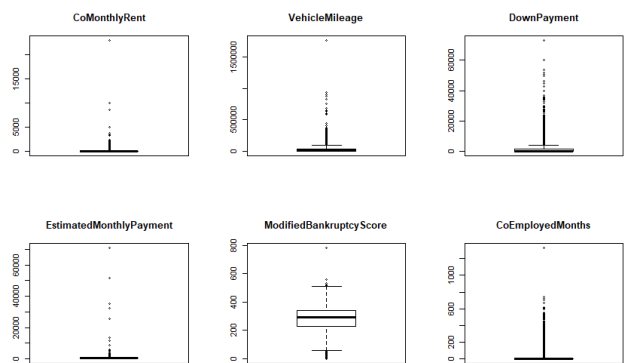


Figure 1.4: Box plots for CoMonthlyRent, VehicleMileage, DownPayment, EstimatedMonthlyPayment, ModifiedBankruptcyScore, and CoEmployedMonths.

In general, we can see that this data set contains a large number of outliers. This observation is important because it helps us determine whether we want to keep outliers or not. For the majority of the features, the outlier values looked acceptable. We can see the outlier box plots in Figure 1.1, Figure 1.2, Figure 1.3, and Figure 1.4. Seen in Figure 1.3, PrimeMonthlyIncome included a few suspicious entries such as entries making more than 50K-100K a month. Ultimately, we entertained these entries as valid possibilities.

Histograms for numerical features Histograms were generated for the numerical features for validation and to gain contextual understanding of the training data. Based on feature ranking results obtained later in the project, we created histograms of the highest-ranked features.

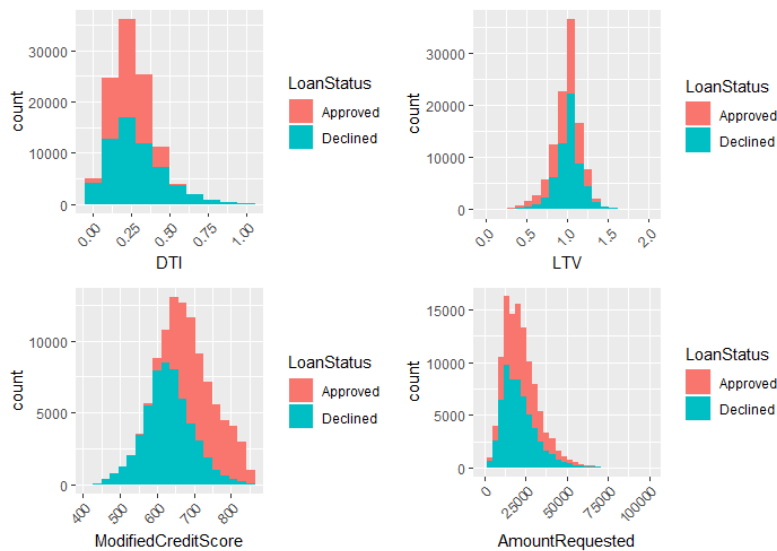


Figure 1.5: Histograms of DTI, LTV, ModifiedCreditScore, and AmountRequested. Y-axis shows divides the population count based on LoanStatus 'Approved' or Denied.

The histograms can be found in Figure 1.5, Figure 1.6, and Figure 1.7. In order to better visualize the histograms, we transformed the y-axis of certain feature histograms using \log_{10} . The histograms describe the distributions of chosen numerical features. The y-axis contains the population count with the LoanStatus feature separated by "Approved" or "Denied." This assists validation by allowing us to view if trends match with the context. For instance, looking at EstimatedMonthlyPayment in Figure 1.6, we can see that how there tend to be less data points with high monthly payments, and we can see that as the value of Estimated-MonthlyPayment increases, the rate of Approved/Declined loans starts to decrease slightly. This could indicate predictive power of the EstimatedMonthlyPayment feature. In this way, we gain insights and increase contextual understanding of our data.

Finally, removing outliers from our data set increased cross-validation accuracy by 1%, but we decided to keep outliers in our data set. None of the outliers seemed like errors, and given that the data set has many outliers, we believe removing them would be unconventional and unwise, as this may adversely affect the test error or ROC.

Imputation Imputation estimates values for missing data. Identification of missing values is an important step in EDA. Once identified, a missing data must be imputed or left alone.

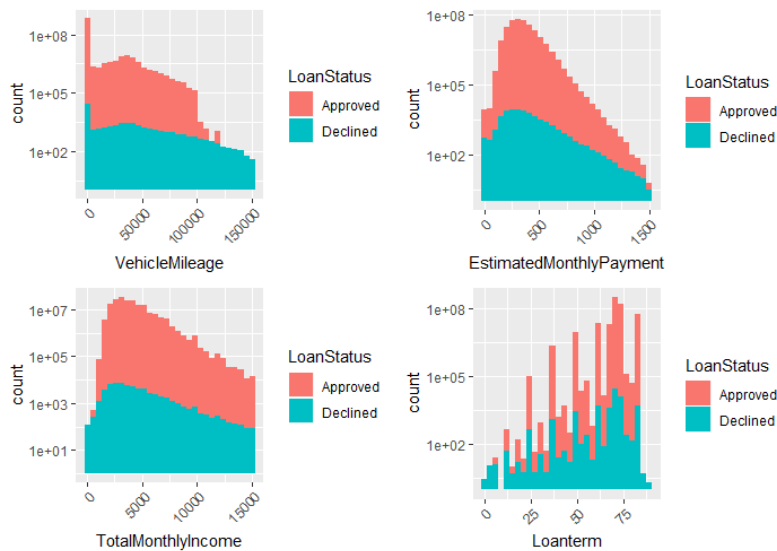


Figure 1.6: Histograms of VehicleMileage, EstimatedMonthlyPayment, TotalMonthlyIncome, and Loanterm. Y-axis shows divides the population count based on LoanStatus 'Approved' or Denied. Some histograms use a \log_{10} transformation on the y-axis.

This decision depends on context. The missing data in the CFE data set can be seen in Figure 1.8. This figure describes the missing data of 27 important features. The left side shows how much missing data feature has in proportion the remaining features. The right side shows the different combinations of features missing data and their percentage of occurrence. For example, the second row from the bottom in all blue represents the 0.25% of data points that have no missing data at all. The first row with two yellow columns all the way to the left represents the 0.65% of data that has values missing for CoMonthlyLiabilities and CoMonthlyRent.

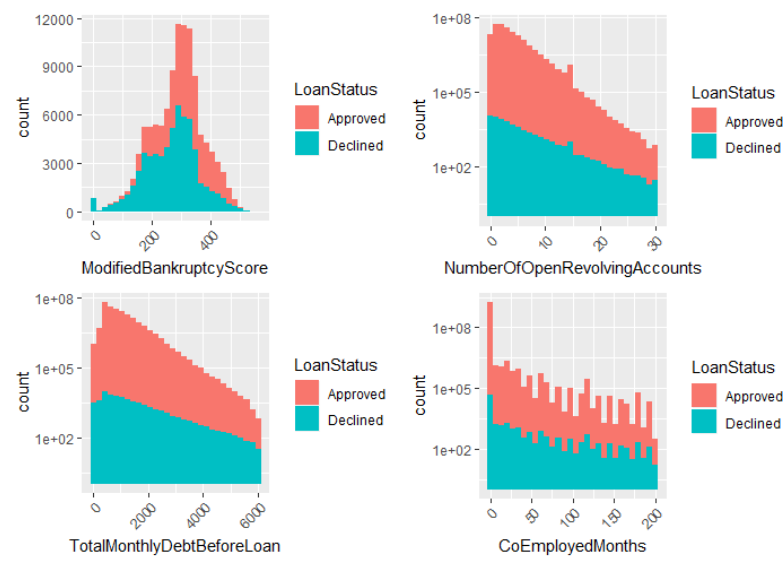


Figure 1.7: Histograms of ModifiedBankruptcyScore, NumberOfOpenRevolvingAccounts, TotalMonthlyDebtBeforeLoan, and CoEmployedMonths. Y-axis shows divides the population count based on LoanStatus 'Approved' or Denied. Some histograms use a \log_{10} transformation on the y-axis.

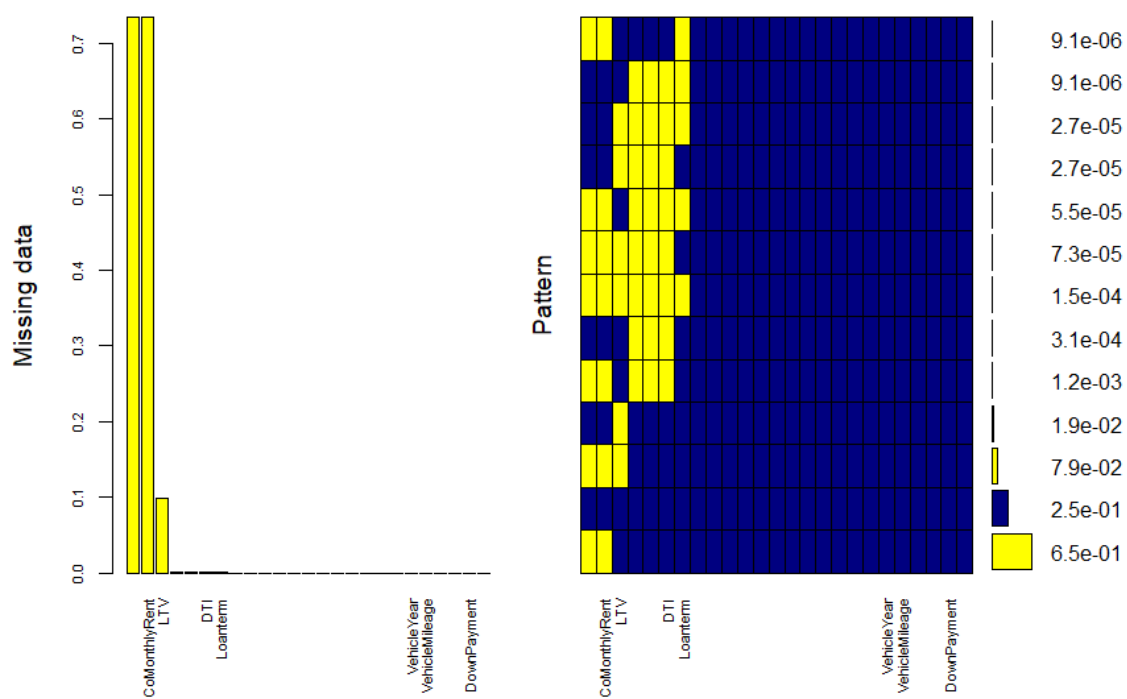


Figure 1.8: Describes the missing data of 27 important features. Team 2

We used two methods of imputation. One was imputation using the median and the other imputation by substituting '0' values. Median imputation was preferred in this case because it is more robust to outliers, which this data set includes a lot of. We used median imputation for LTV, DTI, Loanterm, EstimatedMonthlyPayment, and NumberOfOpenRevolvingAccounts. For CoMonthlyLiability and CoMonthlyRent, we simply replaced the "NULL" values with 0's. This was because we suspected the values were missing not at random and due to users not submitting a value. Therefore, it's safe to say that an entry with a NULL value for CoMonthlyLiability or CoMonthlyRent can also have a 0 value. This choice made our feature engineering easier.

Correlation Correlation, aka collinearity, determines the strength of a potential linear relationship between numerical features. Data containing correlated features often reduce the cross-validation accuracy of models. Correlated features must be identified and removed.

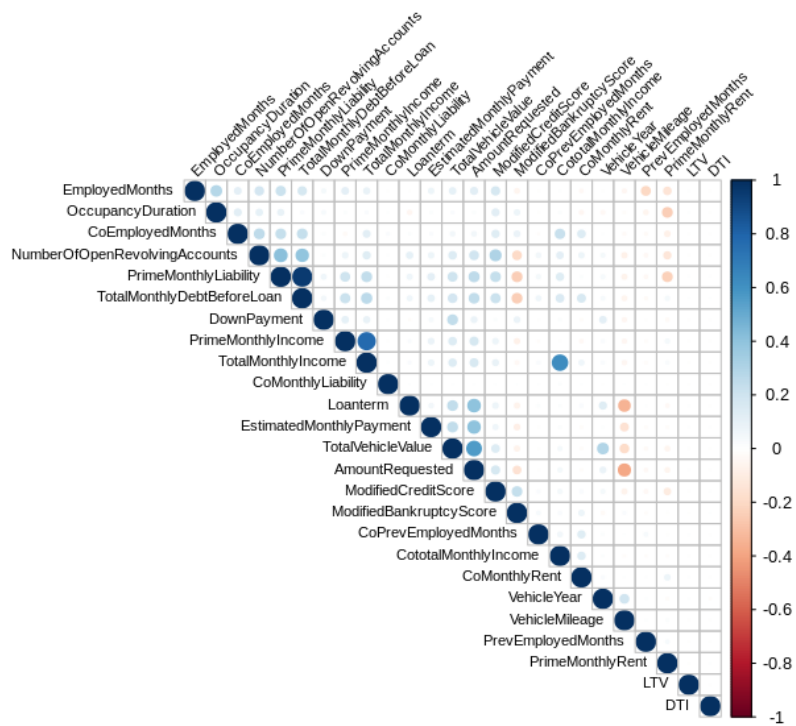


Figure 1.9: Upper triangle of a correlation color plot. The intensity of red/blue indicates the strength of the negative/positive linear relationship.

From Figure 1.9, we can see that PrimeMonthlyLiability and TotalMonthlyDebtBeforeLoan are strongly correlated. Thus, we decided to remove PrimeMonthlyLiability from the data set. Also, we can see that TotalVehicleValue and AmountRequested have a linear relationship, which makes sense. So do Loanterm/AmountRequested and vehicle Mileage.

Categorical Feature Validation

AppReceiveDate: The training and testing data sets both include loan applications from Jan 2015 to Sep 2018. We plotted AppReceiveDate from both data sets as Figure 1.11 show, and there is no pattern between AppReceiveDate and LoanStatus. Those applications should be randomly selected. Further more it didn't improve accuracy when we did cross validation on train data set, so we exclude AppReceiveDate feature when fed to our models.

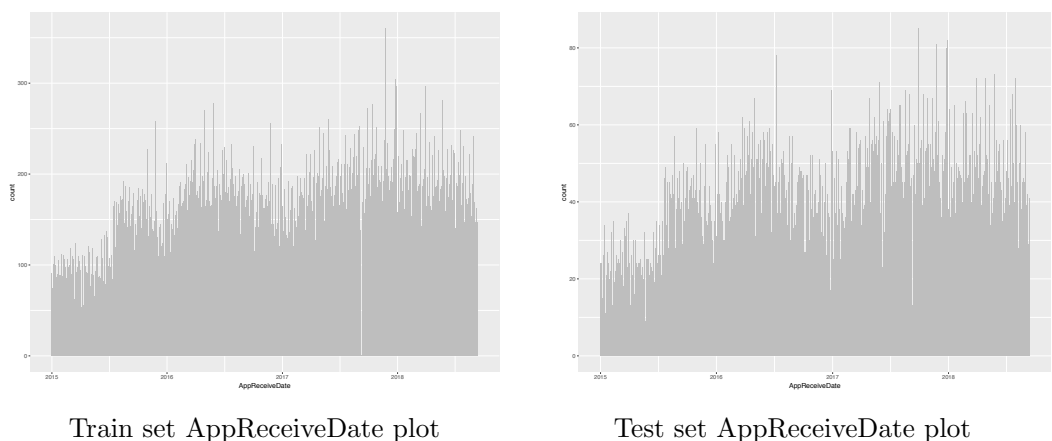


Figure 1.11: AppReceiveDate plot

LoanNumber: The LoanNumber features of train and test data sets have the same range and were randomly selected. It's irrelevant to loan decisions, so we excluded this feature for training out models.

LoanStatus: This is the response variable of our model, which has two value –'Approved' and 'Declined' to show if an application is accepted or not. There are 49466 cases are approved and 60388 cases were denied.

Source: From the Figure 1.12 we can see, 'LENDER' has higher chance(over50%) to get accepted, and 'CONSUMER' and 'GATEWAY' has higher chance(over 50%) to get declined. It seems important at moment so we include this feature in our model, and it was approved so during feature selection phase.

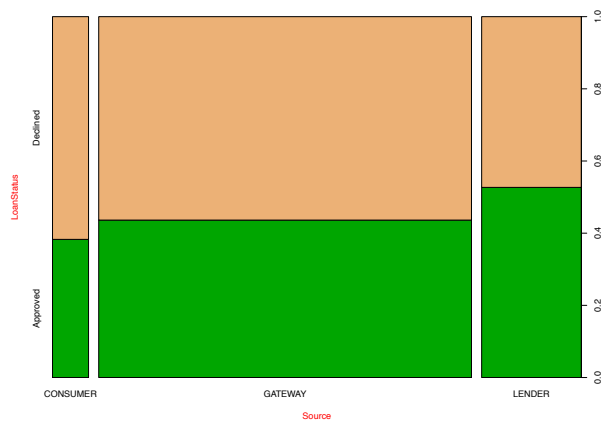


Figure 1.12: Source

EmploymentStatus: From Figure 1.13a we can see, different employment status has different approve rate. For instance, 'Employed' has higher declined rate than 'Retired'. Therefore this feature was used to train our models.

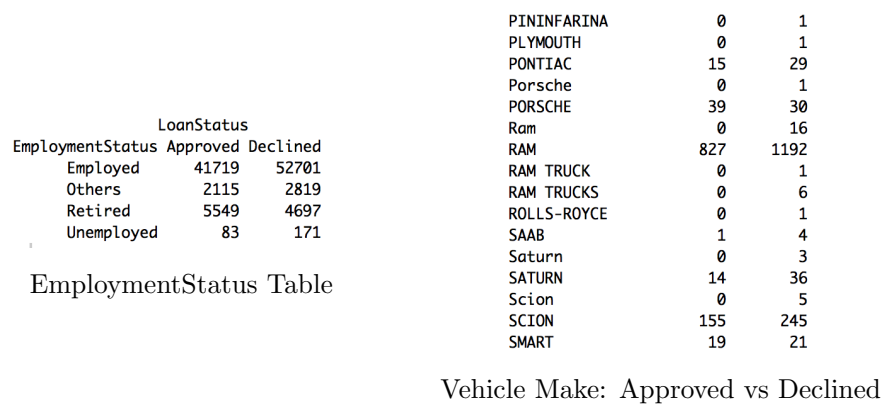


Figure 1.14: Various predictors vs LoanStatus

VehicleMake: Figure 1.13b are a sample of some vehicle makes, from it we can see the brands which are not so popular(less than 10 applications) and not luxury were tend to be

declined. Some brands have two values because of upper/lower case difference, like 'Ram' and 'RAM'. We will consider to combine those values, make them all to upper case. This feature seemed important at first impression, as some brands have lower approve rate(for example: RAM) than others. However, it turned out that without it we achieved more than 5% higher prediction accuracy with our models. One reason could be there are too many values(brands) and the algorithms we applied couldn't handle this feature well. Another possible reason is that, different approve rates between brands is the result but not a cause for application decisions. It wouldn't make sense to favor some brands than others as long as those brands are qualified. Therefore, car brands may not be much relevant to loan decision, and may bring noise to models and reduce accuracy.

RequestType: As Figure 1.15a displays, approve rates differ between values. For example, applications from 'INDIRECT' had just around 40% approve rate while 'CAR SALE' had around 60% approve rate.

OccupancyStatus: 'BUYING' higher the chance to be approved while 'LIVEWITHPARENTS' and 'RENT' had negative impact to approve. Those are expected, as generally home owner should have better financial situation than people who live with parents or rent a place. See details in Figure 1.15b.

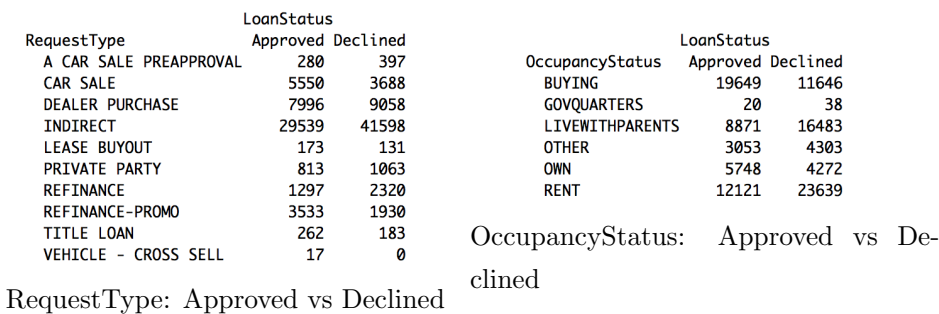


Figure 1.16: Various predictors vs LoanStatus

MemberIndicator: An applicant with CFE membership has higher approve rate, which is more than 60%, as shown in Figure 1.17.

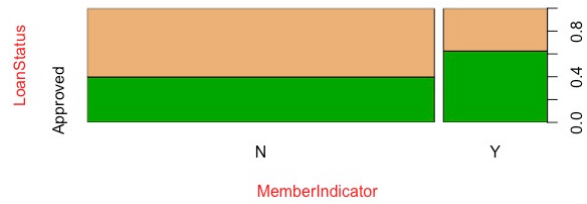


Figure 1.17: LoanStatus MemberIndicator

CoApplicantIndicator: There is no significant approve rate difference whether an application had a co-applicant or not. Though we included it when trained our models, and from the results we can say it has trivial contribution to our model and it doesn't bring noise like VehicleMake.

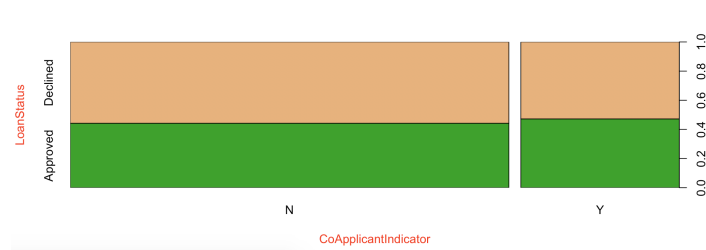


Figure 1.18: LoanStatus CoApplicantIndicator

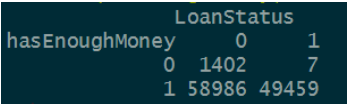
CHAPTER 2: Feature Engineering

Feature engineering is the process of combining existing features to create new features that increase the predictive power of a model. For the CFE data, we created the following new features: `isPaymentDeficit`, `DownToAmountRequested`, `isPotentiallyUnstable`, and `PaidOffRatio`. The following sections briefly describe each feature and the motivations behind creating them.

isPaymentDeficit

The logic behind `isPaymentDeficit` is relatively straightforward. It's a factor column that determines whether or not an applicant has enough cash flow each month to pay for the loan amount requested (`AmountRequested`). Indices in this column received a 1 if `EstimatedMonthlyPayment` was less or equal to the difference between `TotalMonthlyDebtBeforeLoan` and `TotalMonthlyIncome`, and 0 otherwise. The idea being that a 0 value indicates a deficit in monthly cash, rendering the applicant unable to afford the monthly payments. Adding this feature increased the overall accuracy approximately 0.2% using boosting.

To gain a better intuition of predictive potential of *isPaymentDeficit*, we created a logical negation of the column, *hasEnoughMoney*, and compared it to *isPaymentDeficit*. We wanted to see how `isPaymentDeficit` aligned with `hasEnoughMoney`. In other words, of the entries with approved loans, how many also had enough in cash flow each month to afford the estimated monthly payment. Figure 2.1 shows us the result. One of the most promising aspects of the resulting table is that, out of approximately 110K data points, only 7 entries that didn't have enough money were also granted a loan. Perhaps this will increase the accuracy of the model, or maybe it will improve the type 1 error of the model. The latter in this case being contextually very important.



| | LoanStatus | |
|----------------|------------|-------|
| hasEnoughMoney | 0 | 1 |
| 0 | 1402 | 7 |
| 1 | 58986 | 49459 |

Figure 2.1: Comparison of Loan

DownToAmountRequested

This feature represents the ratio of DownPayment to AmountRequested. Motivation for this feature was that applicants that paid a greater percentage of their loan up-front might have a lower chance of loan delinquency.

isPotentiallyUnstable

This feature represents the potential for an applicant to be unemployed during the loan repayment period. It is calculated by determining if PrevEmployedMonths is less than the Loanterm, potentially indicating a trend of transiency in employment. This feature was highly experimental and proved to be no use at all.

Finally, as discussed in the results section, adding these features increased accuracy approximately 0.2-0.3%. After adding the new features, we recomputed the correlation plot to detect any potential correlation between values. Figure 2.2 shows this plot.

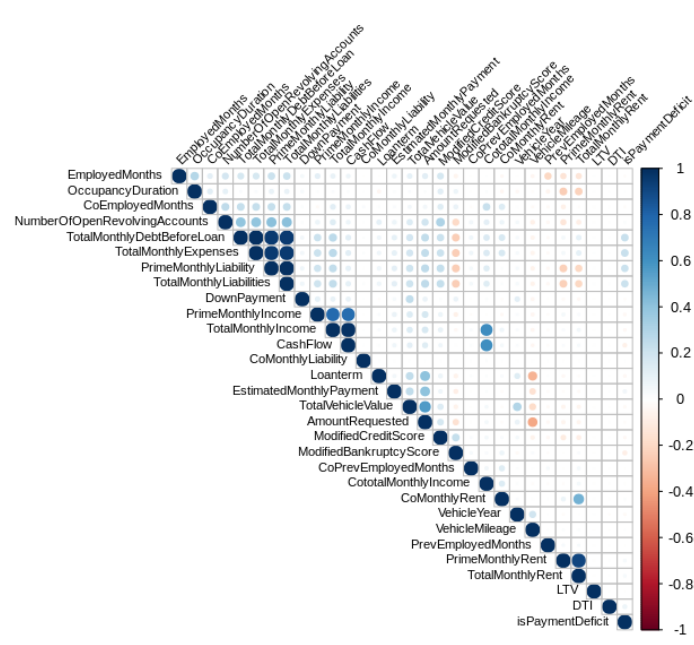


Figure 2.2: Correlation plot with additional features from feature engineering.

CHAPTER 3: Model Selection and Fitting

With linear data set, it is best to fit it into a linear regression model rather than nonlinear, and vice versa. If you have a linear correlated data set, you need a simple model like linear regression. Even the best CNN will give you a poor result. So to choose an appropriate model is one of keys for a good performance. Our data most likely has a complex decision boundary, and give the small number of features compared to the total number of data points, using more flexible models to classify the data was acceptable. We used logistic regression, lasso regression, random forests, boosting, and neural networks. Cross validation (cv) on models was performed using mainly two splits: 50/50 and 95/5 of the training data set, which means 50% or 95% of data was used to train our models, and left was used to test. And there are further k-fold cross validation applied on the 50% or 95% training data, depending on the model we used.

Lasso regression

Logistic regression, a classifier that uses a sigmoid function to output a probability of the response variable's class, can perform classification on a data set with numerical and categorical features. Lasso regression, is logistic regression with an added constraint that bounds the coefficient estimates beneath the sum of their L1-norm. This sometimes results in coefficients values of 0, reducing the size of your feature space. We trained the model on the 29 features that remained after EDA and data cleaning.

We performed cross-validation and computed the associated test error to tune the parameter lambda. Figure 3.1 is a plot of test error on different lambda values. After we found the best lambda value with smallest validation test error, we fitted our model with whole training data set. Figure 3.2 shows some coefficients of the model with certain coefficients valued at zero as a result of the L1-norm of lasso regression..

Once the model was built, we predicted the resulting class using the models estimated output. First we set prediction criteria to 0.5: Applications would be predicted as "Approved" when prediction probability is greater than 0.5. Figure 3.3a is the confusion matrix of prediction and test data responses with a criteria 0.5, we can see the accuracy is 80.6%. Type 1 error rate is 9.1% and type 2 error rate is 10.2%. Interestingly, without imputation of DTI and

LTV, the accuracy is basically the same at 80.7, no significant decrease.

There are times when measures other than cv accuracy of a model are important. For instance, type 1 Error, aka specificity, is a measure of the false positive rate yielded by a model on a validation set. In other words, the proportion of the negative data points in a data set that were falsely identified as positive. This metric can be very important depending on the context. In this case, it is important to know the model's type 1 Error so that CFE approves only the qualified applicants. Setting a higher prediction criteria reduces the type 1 error and increase type 2 error. After trying several different values, we chose 0.7. A threshold of 0.7 resulted in no significant change in accuracy and a reduction in type 1 error from 10.1% to 4.1%. The compensation is that type 2 error rate went up. So which criteria to choose is based on institution's priorities. Figure 3.3b shows confusion matrix when prediction criteria is 0.7.

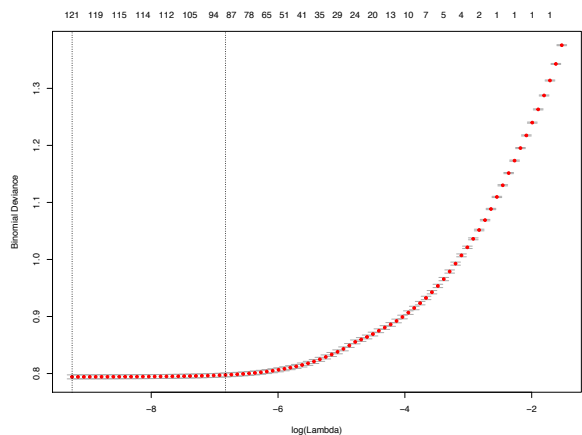


Figure 3.1: Test error vs lambda

| | |
|----------------------------|---------------|
| PrimeMonthlyIncome | -1.659212e-05 |
| CototalMonthlyIncome | -3.159937e-05 |
| TotalMonthlyIncome | -4.009041e-05 |
| CoMonthlyLiability | 1.769952e-02 |
| PrimeMonthlyRent | 8.277041e-04 |
| TotalMonthlyDebtBeforeLoan | 2.703951e-04 |
| VehicleYear | 8.678294e-05 |
| VehicleMakeACURA | 1.836236e-01 |
| VehicleMakeALFA ROMEO | 2.167921e+00 |
| VehicleMakeASTON MARTIN | . |
| VehicleMakeAudi | . |
| VehicleMakeAUDI | -9.968201e-02 |
| VehicleMakeBENTLEY | -4.765845e+00 |
| VehicleMakeBMW | -1.327969e-01 |

Figure 3.2: Model coefficients

| | observed.classes | | | observed.classes | |
|---|------------------|----------|--|---|-------------------|
| predicted.classes | Approved | Declined | | predicted.classes70 | Approved Declined |
| Approved | 17982 | 4478 | | Approved | 13714 2034 |
| Declined | 5032 | 21894 | | Declined | 9300 24338 |
| > mean(predicted.classes == observed.classes) | | | | > mean(predicted.classes70 == observed.classes) | |
| [1] 0.8074353 | | | | [1] 0.7705018 | |

Confusion Matrix with prediction criteria 0.5

Confusion Matrix with prediction criteria 0.7

Figure 3.4: Confusion matrix result comparison for different criteria values

Random Forests/Boosting

Random forests are an ensemble learning method for classification that operates by constructing multiple tweaked decision trees to overcome the overfitting of a single decision tree. It also reduces model variance while building decorrelated trees.

We trained two random forest models. One with all 29 features that remained after data cleaning, and another with 6 randomly selected features at each split when building a tree. The cv accuracy of those models is 86%, which increased 6% compared to logistic regression model. This implies that tree-based models are a good choice to handle our data set. Boosting is one popular tree-based algorithm, which grows trees sequentially: each tree is grown using information from previously grown trees. We built a boosting model applied gbm library, with 29 features and initially set 5000 trees, depth to 5 for each tree, and it achieved an accuracy of 88%. This was promising, so we continued to tune boosting parameters and tried to get a better performance.

There are 3 critical parameters to tune for a boosting model: The number of trees B , the shrinkage parameter λ and the number d of splits in each tree. First we build a model with 10000 trees, shrinkage parameter $\lambda = 0.1$ and tree depth $= 3$, as well as 3-fold cross validation. After trained, we plotted the cv error vs tree number as Figure 3.5 shows. Cv error reached lowest point when tree number was 2468. So we continued to build boosting models with 5000 as tree number.

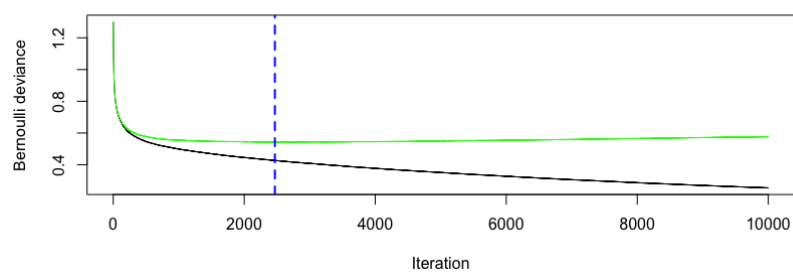


Figure 3.5: CV error vs tree number

One advantage of gbm library is that it can handle mixed data set directly, and drawback is that it requires intensive computation resource. It's time consuming to manually find a good setting of λ and d , so we set up a loop to try different combinations: $\lambda = 0.01$ or 0.1 , $d = 3$ or 5 . To speed up the tuning process, we only used 4 combinations, performed 2-fold cross validation. Figure 3.6 shows the minimum cross validation errors and their tree numbers for all 4 λ and d combinations. We can see models with 5 depth trees have lower error, so we chose depth equal 5. For shrinkage parameter, models with 0.01 shrinkage have slightly higher error, but their optimal tree number is 4999, which implies that the models haven't reached lowest error before we stopped training. So we chose λ to 0.01 and increased number of trees to 7000 for our final solution model. This approach can be applied to find a best result of multiple parameter combinations, adding more parameters or/and different values of each parameters.

| | shrinkage | interaction.depth | optimal_trees | min_RMSE |
|---|-----------|-------------------|---------------|-----------|
| 1 | 0.10 | 5 | 1308 | 0.7395426 |
| 2 | 0.10 | 3 | 2448 | 0.7411819 |
| 3 | 0.01 | 5 | 4999 | 0.7455226 |
| 4 | 0.01 | 3 | 4999 | 0.7602532 |

Figure 3.6: Boosting parameter tuning result

Boosting model gives us the best performance, as it has a good prediction accuracy, false positive rate and false negative rate are in reasonable range. So we chose it as final model. Further more it's easy to understand: it's a decision tree-based model, which mimics human decision process: for example, when an agency review an application, he/she checks if the credit score high enough. If so, he/she may look at debt to loan index. He/she will decline the application if the index is high, otherwise continue to check other features for the final decision.

As mentioned above we applied gbm library to implement boosting, it can handle mixed data(numeric and categorical) directly and allows us to tune many parameters. The drawback is the computation time: it took 20/30 minutes to train one model. In future we will consider to try other boosting library like xgboost, which need to convert categorical features into one-hot dummy variables before feeding data to models, but this package need significant less computation time.

Like we did for logistic regression, we also used two thresholds: 50% and 70% to check false positive rate and false negative rate. Figure 3.8 shows Confusion matrices with different thresholds. If false positive rate is critical to CFE, we would suggest using higher threshold 70%, which reduces accuracy 2%. But it also reduces false positive rate from 7% to 3.5%, and can help CFE to reduce risk.

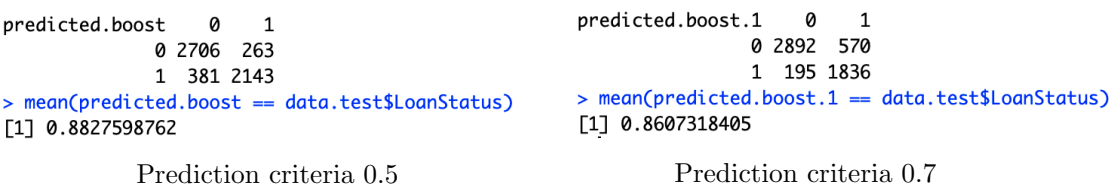


Figure 3.8: Confusion matrix comparison for boosting models

Finally, the gbm library gives us feature ranking table like Figure 3.9 that shows us the relative importance of the features. We can see feature ModifiedCreditScore and DTI are most important features for the model. Figure 3.10 shows us this ranking in graph. Based on this table, we can conclude that our engineered features didn't have much impact on the model.

| | var | rel.inf |
|-------------------------------|-------------------------------|----------------|
| ModifiedCreditScore | ModifiedCreditScore | 57.45204634753 |
| DTI | DTI | 13.20460846023 |
| EstimatedMonthlyPayment | EstimatedMonthlyPayment | 4.10811302500 |
| AmountRequested | AmountRequested | 3.89263536231 |
| VehicleMileage | VehicleMileage | 3.78650731701 |
| LTV | LTV | 2.40609935557 |
| MemberIndicator | MemberIndicator | 2.32276472486 |
| RequestType | RequestType | 1.95996219000 |
| Loanterm | Loanterm | 1.65364047179 |
| TotalMonthlyIncome | TotalMonthlyIncome | 1.51187614375 |
| OccupancyStatus | OccupancyStatus | 1.07594181339 |
| NumberOfOpenRevolvingAccounts | NumberOfOpenRevolvingAccounts | 1.00871596352 |
| CoEmployedMonths | CoEmployedMonths | 0.96588652095 |
| TotalMonthlyDebtBeforeLoan | TotalMonthlyDebtBeforeLoan | 0.80928696593 |
| VehicleYear | VehicleYear | 0.76332488862 |
| ModifiedBankruptcyScore | ModifiedBankruptcyScore | 0.74369495889 |
| EmployedMonths | EmployedMonths | 0.63557897633 |
| TotalVehicleValue | TotalVehicleValue | 0.62131082734 |
| Source | Source | 0.22178325496 |
| OccupancyDuration | OccupancyDuration | 0.19167858278 |

Figure 3.9: Boosting features ranking

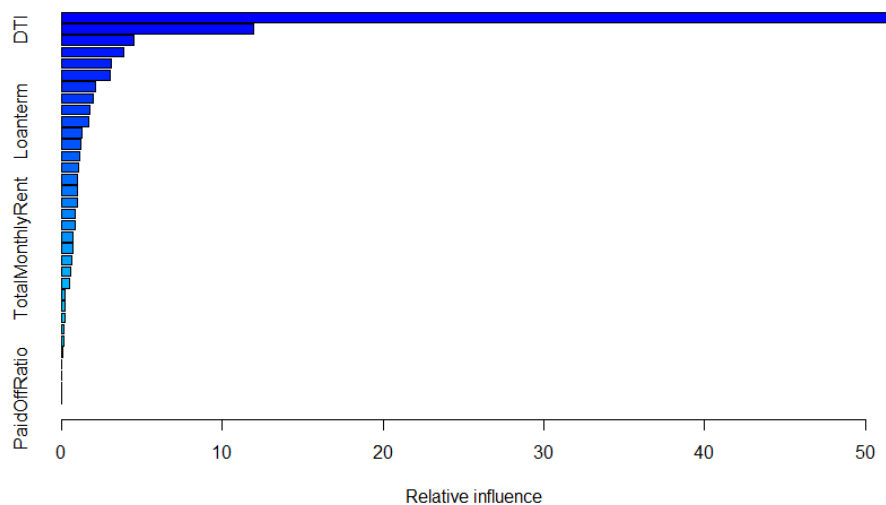


Figure 3.10: Boosting features ranking

Neural Network

The state-of-the-art Artificial Neural Networks (ANN) is best known for classification tasks. The advantage of ANN is that it has the ability to detect all possible interactions between predictors variables and discover hidden patterns. Neural network could also do a complete detection without having any doubt even in complex nonlinear relationship between dependent and independent variables.

In this experiment, we train the neural networks model using **neuralnet** package in RStudio. In **neuralnet** package, all numerical features requires to be normalized and applicants with missing data are ignored automatically. Therefore, after the data cleaning process, we need to impute the empty features and normalize those numerical features between 0 and 1. The **neuralnet** package doesn't take discrete features into account for modeling, so we need to convert those categorical inputs into dummy variables.

Once the data is formed correctly, we split the data into training and testing sets with cross validation. In order for training our neural network model, we used the back-propagation

algorithm and the sum of squared errors (SSE) cost function metric to measure how well the model performs. Figure 3.12 is a sample of neural network model visualization with 10 nodes in one layer. The model can be tested with different numbers of layers and different numbers of nodes in each layer. we came up with the confusion matrix table as shown in Figure ?? the confusion matrix table and accuracy for a simple experiment of 1 neural network layer with 5 nodes in it. More layers and nodes can potentially increase the performance of the results.

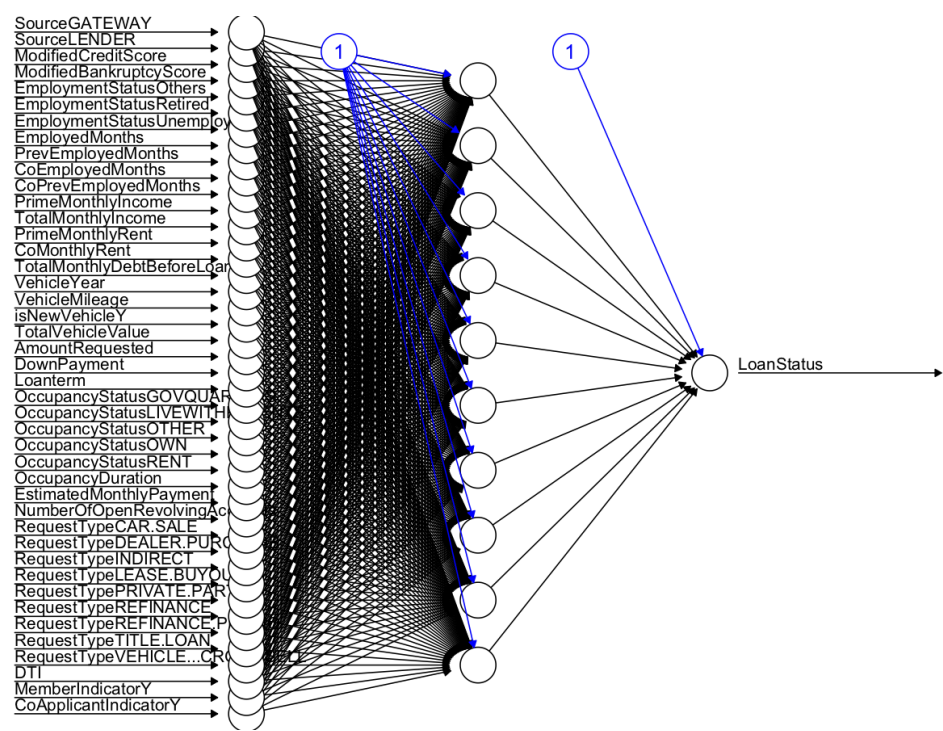


Figure 3.11: Sample Neural Network model visualization

```
[1] "Confusion matrix for NN:"  
> print(nnCM)  
  
pred  0  1  
  0 79 28  
  1 27 66  
> NNaccuracy <- sum(diag(nnCM))/sum(nnCM) #accuracy  
> cat('Accuracy for NN: ', NNaccuracy*100, "%")  
Accuracy for NN: 72.5 %
```

Figure 3.12: Sample Neural Network result for 1 layer with 5 nodes

CHAPTER 4: Conclusions and Future Work

We started with 80% accuracy using logistic regression, and tried different algorithms, applied feature selection, feature engineering and parameter tuning. At end we achieved our best performance: 88% accuracy with Boosting model. We think it's a good result considering the time limit and our academic workloads.

In the future, we would like to try using xgboost, a faster library for boosting. We believe our feature engineering could be improved upon. It is hard to tell if the features given were relatively weak for the data set, or if more creative feature engineering could be done. It is hard to determine which, but we lean toward the latter.

Also, while R is easier to use up front, the majority of R libraries run on a single-core, making Python more suited for running data-intensive algorithms at higher speeds. A perfect example is R's neural network libraries. Neural networks are notoriously computationally intensive and are often computationally infeasible without a GPU. We suspect most neural networks created in R will be either insufficient or computationally intractable given R's architecture. Even though neural network models are uninterpretable, and the higher the complexity of the model the more opaque the model becomes, we would like to try a more complex model to increase prediction accuracy.