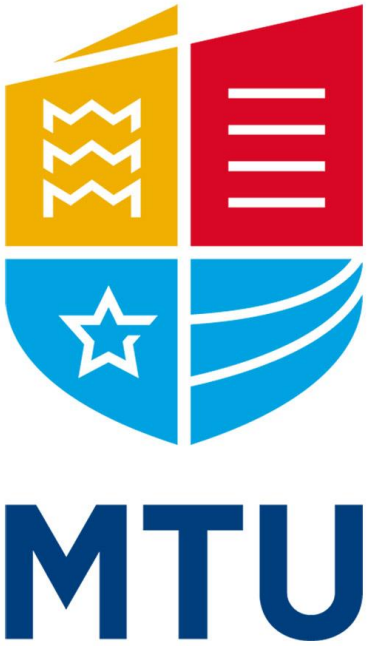


Credit Risk Classification Report

Job Thomas Thekkekara

R00195427

For the module DATA8001_24010 Data Science and Analytics as part of the

Master of Science in Data Science and Analytics, Department of Mathematics, 06/12/2021

Declaration of Authorship

I, Job Thomas Thekkekara, declare that this assignment and the work presented in it are my own. I confirm that,

- This work was done wholly or mainly while in candidature for the Masters' degree at Munster Technological University
- Where any part of this assignment has previously been submitted for a degree or any other qualification at Munster Technological University or any other institution, this has been clearly stated
- Where I have consulted the published work of others, this is always clearly attributed
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work
- I have acknowledged all main sources of help

Signed: Job Thomas Thekkekara

Date: 25-11-2021

Contents

1. INTRODUCTION

Data has the potential to transform business and data driven decision making is gaining more importance than ever in today's world. With the advancements in computing capabilities, data analytics and machine learning are used across all industries to improve businesses. Financial industry is at the forefront of this starting from fraud detection to risk management. One such important application is credit risk scoring. Credit risk scoring is used by financial institutions to ease the decision-making process of accepting or rejecting a loan application. There are several advantages for this which include: faster decisions, less human intervention and bias, reduced cost and so on. A credit risk scoring model is developed by checking the historical personal and financial data of customers. Classification algorithms are commonly used to check if the credit score is good or bad.

Classification is a process in which a set of predictor variables are used to predict the class of a target variable. In credit risk scoring model the target variable would be good or bad credit standing. Numerous classification models are available for this. But the most commonly used algorithms are decision tree and random forest algorithms. These models come under supervised learning. That is, labelled target variables are available for training the model.

This report goes through the process of data cleaning, exploratory data analytics and model building of credit standing using decision tree and random forest classification models and evaluating the performance of these models. It also goes through data privacy, protection and to identify suspicious patterns in the data

2. DATA

The data set was obtained from a financial institution for the purpose of assessing the credit worthiness of future potential customers. The dataset contains the details of 790 customers and their past credit standing. 14 variables are available for each case, including the target variable. The dataset was obtained in a Microsoft Excel file. It was loaded into R Studio for further analysis.

Following were the variables present in the data set, their corresponding data types and subcategories.

Variable	Data Type	Description and Subcategories
ID	Number	Unique ID for each case
Checking Acct	Character	Type of checking account customer has. Type: 0Balance, High, Low, No Acct
Credit History	Character	Past credit record of customers. Type: All Paid, Bank Paid, Critical, Current, Delay
Loan Reason	Character	Reason to apply for loan. Type: Business, Car New, Car Used, Education, Furniture, Large Appliance, Other, Repairs, Retraining, Small appliance.

Savings Acct	Character	Type of savings account customer has. Type: High, Low, MedHigh, MedLow, No Acct
Employment	Character	Type of employment customer has. Type: long, Long, Medium, Retired, Short, Unemployed, Very Short
Personal Status	Character	Marital status of the customer. Type: Divorced, Married, Single
Housing	Character	Type of house owned b the customer. Type: Rent, Own, Other
Job Type	Character	Type of job the customer has. Type: Management, Skilled, Unemployed, Unskilled
Foreign National	Character	Customer is foreign national or not. Type: No, Yes
Months since Checking Acct opened	Number	Duration of account opened
Residence time in current district	Number	Duration of stay in current place (Years)
Age	Number	Age of the customer in Years
Credit Standing	Character	Target Variable. Credit risk classification Type: Good, Bad

Table 1: Data type of variables and its description

First, the data was loaded into a data frame and all character type variables were converted into factor type for exploratory data analysis and model building. This was done using *as.data.frame()* in R. The variable name for *Months since checking acct opened* was converted to *Acct Opened Months* and *Residence time in current district* to *Residence Time* using *names()* for convenience. (See Appendix A for raw data)

3. SUMMARY STATISTICS

Summary statistics were created using *summary()* and *skim().* (See Appendix B for detailed summary).

Data Summary	
Name	Credit_risk_data
Number of rows	790
Number of columns	14
Number of factor variables	10
Number of numeric variables	4

Table 2.1: Overall summary of data

22 missing values were found in the data. Following variables have missing values.

Variables	No Of Missing Values
Employment	11
Personal Status	6
Housing	5

Table 2.2: Number of missing values in variables

3.1 Statistics of Numerical Variables

1) Age

Min	1st Quad	Median	Mean	3rd Quad	Max
18	26	32	34.73	41	99

Table 3.1: summary statistics of Age

Mean age of customers seems to be slightly greater than median age. This indicates that the data is slightly skewed to the right. Standard deviation of Age is 11.57. Also, maximum age of the customer is 99, which might be an outlier, as this customer's reason for taking the loan is New Car. So, this age might be a human entry error. The standard error of Age is calculated to be 0.41. Since both skewness and kurtosis is greater than | 2 times standard error |. Hence not normally distributed. Most customers have age between 20-45

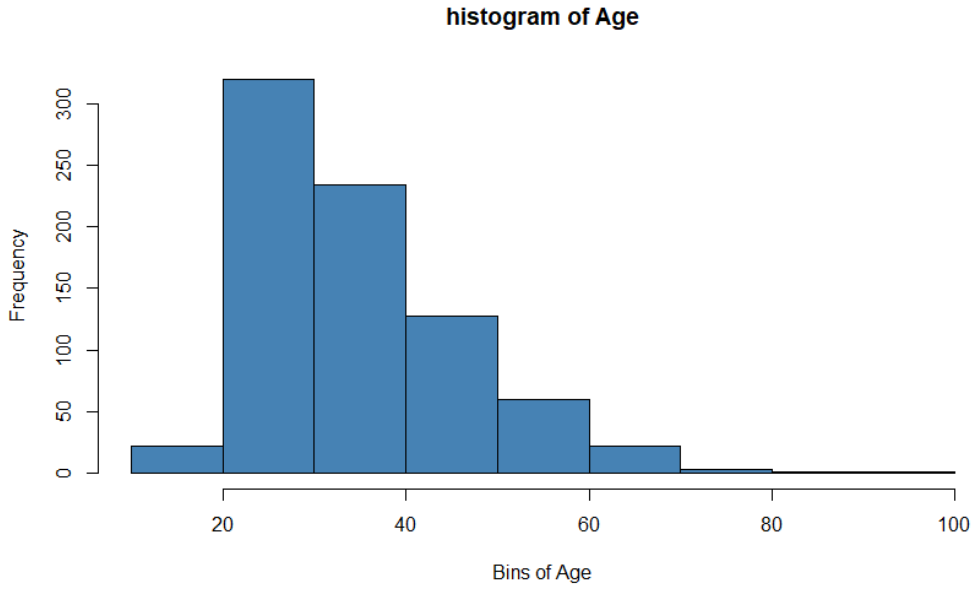


Figure 1.1: Distribution of Age

2) Residence Time (In current district)

Min	1st Quad	Median	Mean	3rd Quad	Max
-2	2	3	2.871	4	10

Table 3.2: summary statistics of Residents time in current district

Mean and Median of residence time is very close to each other, which indicates normal distribution. Minimum value is negative which is clearly an error as residence time cannot be negative. Standard deviation is 1.16. Skewness is 0.22. It is close to zero, indicating no major skewness to the data. Kurtosis is 1.903, which shows higher peak than normal in the distribution. 99% of customers have lived in the current district between 0-4 years.

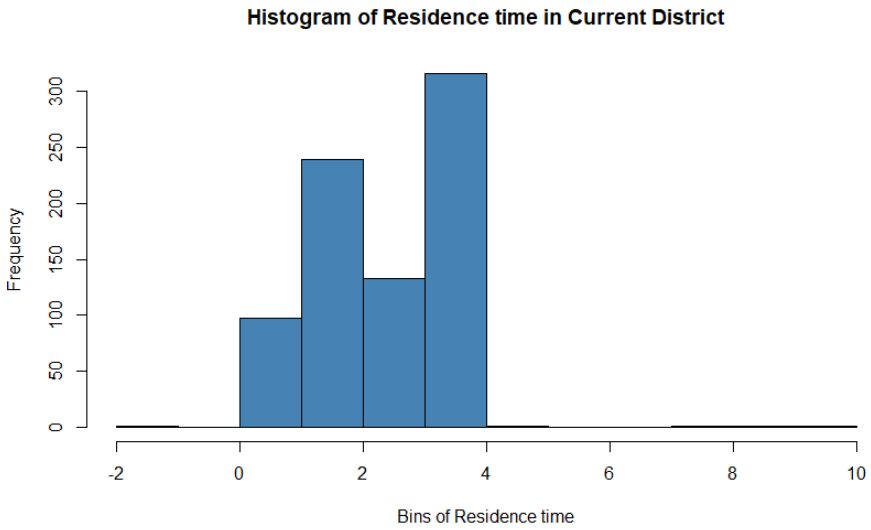


Figure 1.2: Distribution of Residence time in current district

3) Acct Opened Months (Checking Acct)

Min	1st Quad	Median	Mean	3rd Quad	Max
5	13	19	23.15	29	120

Table 3.3: summary statistics of months since Checking Account opened

Mean is slightly greater than median, showing right skewness. Standard deviation is 12.78. Skewness is 1.37, indicating right skewness to the distribution. Kurtosis is 4.16 which much higher than zero and hence a very high peak than normal in the distribution. More than 95% of customers have opened their checking account between 0-40 months. An outlier of 120 months is observed from the stem plot and histogram

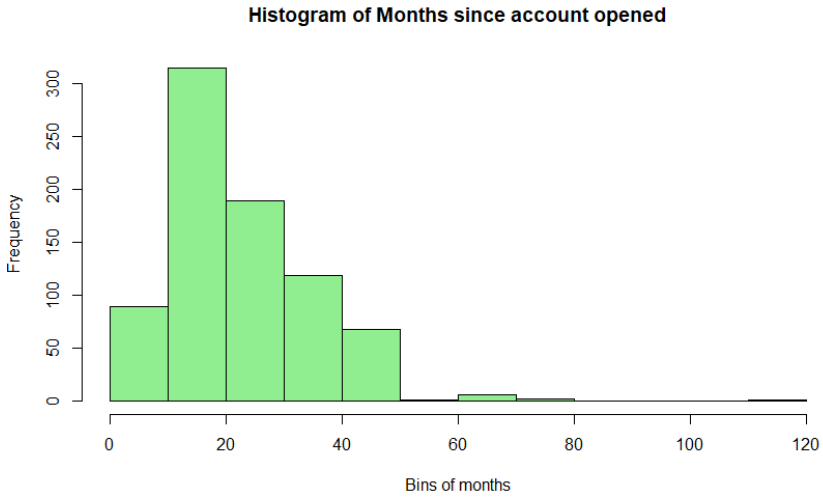


Figure 1.3: Distribution of Months since Checking Account opened

4.EXPLORATORY DATA ANALYSIS

4.1 Univariate Analysis

Univariate analysis of all variables was carried out using contingency tables in R. Some of the important observations are given below

1) **Credit Standing/Target Variable**

BAD	GOOD
0.4101266	0.5898734

Table 4.1: Proportion of customers in credit standing

About 60% of data have good credit standing. Hence it is a slightly imbalanced data. This bias can lead to poor predictive performance of the classification model.

2) **Checking Acct**

0Balance	High	Low	No Acct
0.33670886	0.05696203	0.25822785	0.34810127

Table 4.2: Proportion of customers with checking acct

Very few customers have high checking acct. Most of them have 0 Balance or No Acct.

3) **Saving Acct**

High	Low	MedHigh	MedLow	No Acct
0.0367	0.6430	0.0632	0.1012	0.1556

Table 4.3: Proportion of customers with savings acct

More than 60% of customers have Low checking Acct.

4) **Housing**

Other	Own	Rent
0.1210	0.6777	0.2012

Table 4.4: Proportion of customers in Housing

Almost 70% of customers have their own house.

5) **Job Type**

Management	Skilled	Unemployed	Unskilled
0.1291	0.6417	0.02405	0.2050

Table 4.5: Proportion of customers with Job Type

64% of customers are skilled and 12 % are in management. Only very few customers are unemployed, which is a good indicator.

6) Foreign National

No	Yes
0.3215	0.6784

Table 4.6: Proportion of customers in that are foreign national

About 68% of customers available in the data are foreign national.

4.2 Bivariate Analysis

A bivariate analysis of all predictors with the target variable was checked to understand which variable can have more influence in the prediction. Also, analysis between predictor variables were also carried out to check on correlation and interactivity between predictor variables. Some of the important analysis is given below:

Checking Acct and Credit Standing

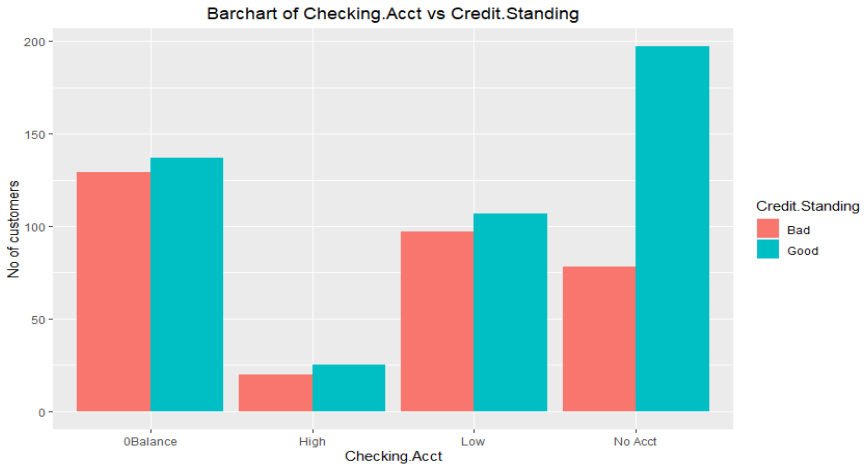


Figure 2.1: Bar graph of checking acct with credit standing

Large number of accounts belong to No Acct category and more than 50% of them have good credit standing. All other categories have equal number of good and bad credit standings.

Credit History and Credit Standing

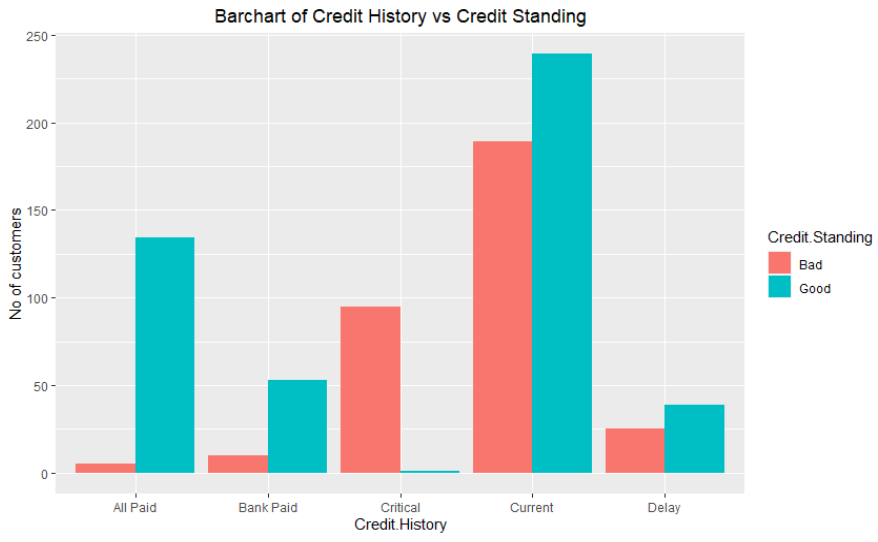


Figure 2.2: Bar graph of credit history with credit standing

From the chart it is observed that the type of credit history of a customer can easily influence the credit standing. **All subcategories in credit history except current can clearly classify the target variable**. All paid & bank paid credit history mostly have good credit standing. Critical accounts have bad credit standing in almost all cases. CHI-SQUARE test was conducted to establish significance between variables. (See Appendix C)

Employment and Credit Standing

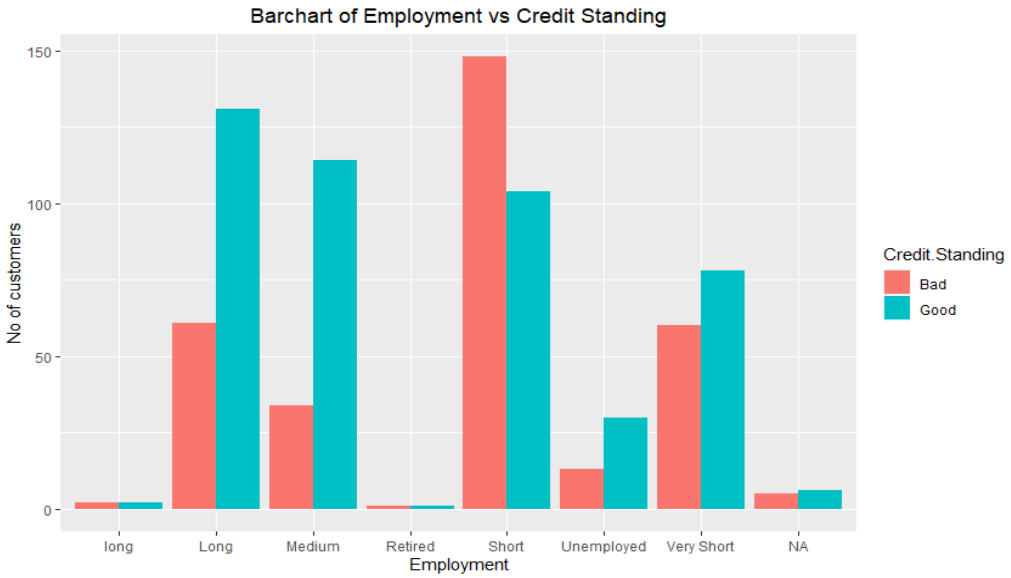


Figure 2.3: Bar graph of employment with credit standing

Short term employment has very high number of bad credits standing classes whereas long and medium type employment seems to have high number of good credit standing classes.

Housing and Credit Standing

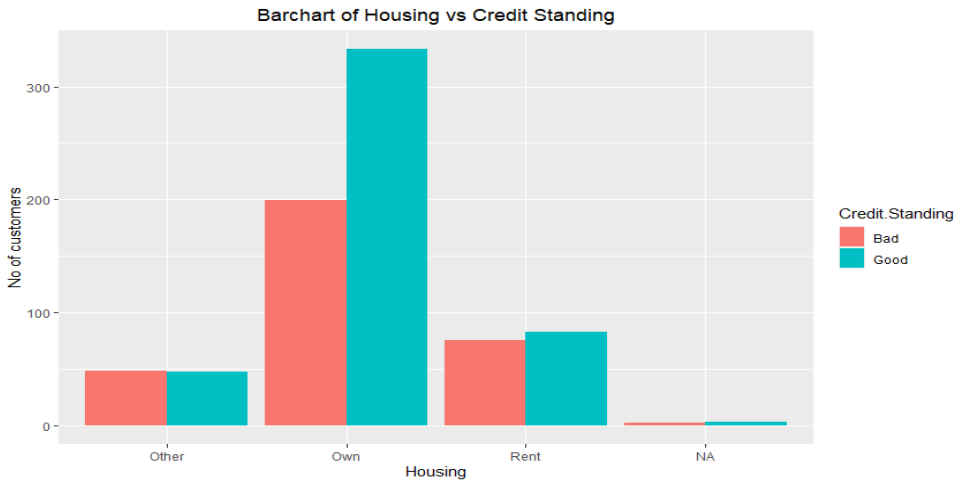


Figure 2.4: Bar graph of Housing with credit standing

Most customers have their own houses and they a lot of them have good credit standing. NA values are also observed.

See more the results of Bi Variate analysis conducted in Appendix D

4.3 Tri-variate Analysis

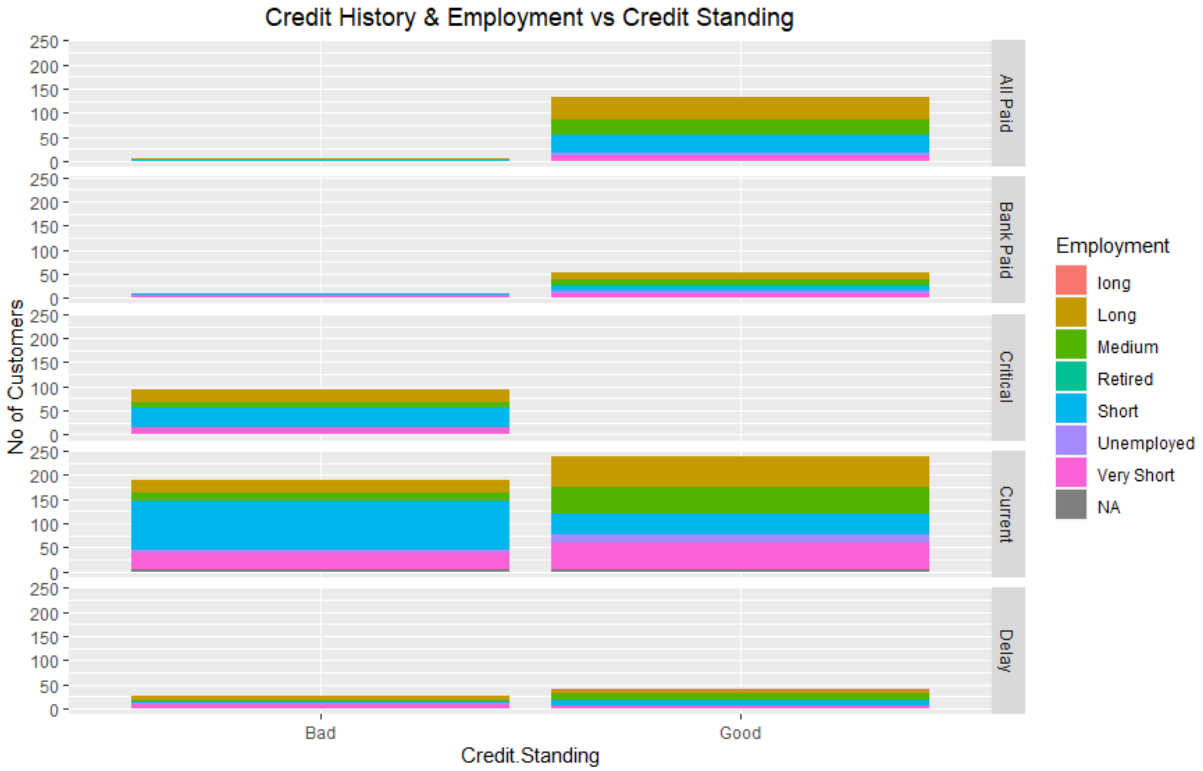


Figure 3.1: tri variate analysis of credit history employment and credit standing

Plot of credit history and employment vs the target variable is shown in the above figure 3.1. It can be seen that customers having bad credit standing mostly belong to critical, current and delay credit history categories. Also, these customers mostly have short or very short type employment. Customers having employment status as Long belong to paid and current credit history categories and they generally have good credit standing.

This clearly shows that people with short term employment and critical or delayed credit history tend to have a bad credit standing.

Figure 3.2: tri variate analysis of credit history job type and credit standing

From the above figure 3.2, it is observed that majority of customers with all paid credit history are skilled and have good credit standing. This makes sense as they might have the capability to pay back. At the same time a lot of skilled customers have critical credit history and bad credit standing. This indicates that some of the customers took loans showing their skill but were unable to pay back in time.

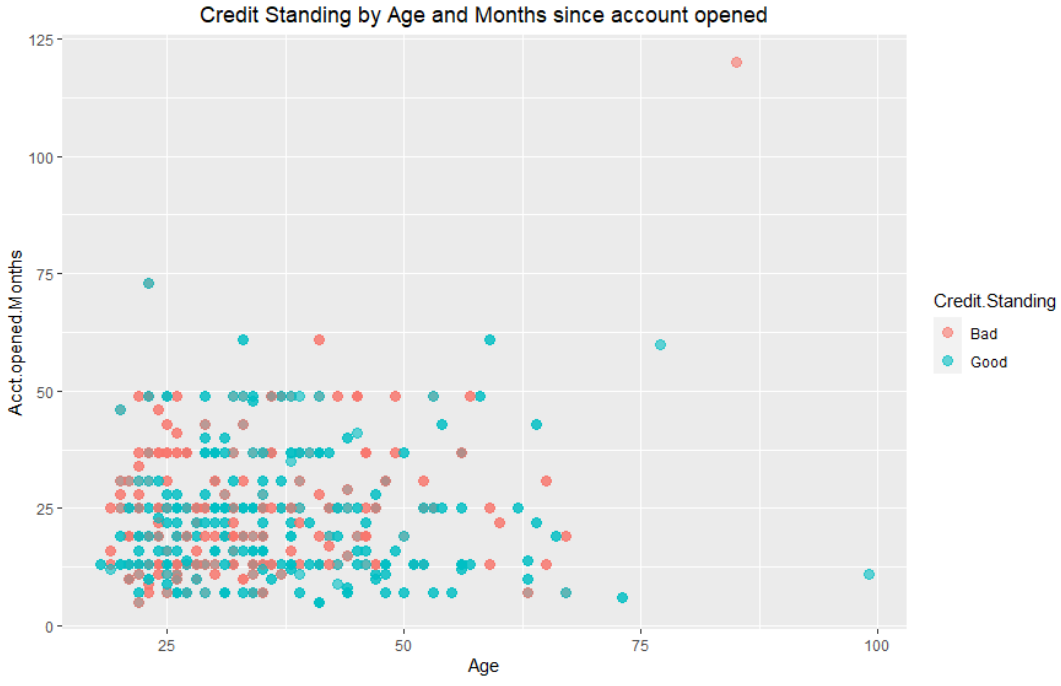


Figure 3.3: tri variate analysis of age, acct opened months and credit standing

It can be seen that most of the customers are below the age of 50 and have opened their accounts in the past 4 years. Also, it is interesting to note that many customers below the age of 25 have a bad credit standing.

4.4 Missing & Invalid Values

The necessity to clean the missing and invalid values in a data set is the most crucial point in the data science process. These values can influence the machine learning model significantly and can lead to poor results or predictions that are far away from the actual truth. Hence it is necessary to either remove these values or impute them. That is, replacing them with a mean or mode of the corresponding attribute.

Following steps were performed as part of data cleaning

- A negative value was observed in the attribute Residence Time. As time cannot be negative, it needs to be removed or replaced. Most of the customers have a residence time of 4 years. Also, as the data is skewed, mean will be highly influenced. Hence the negative value has been imputed with the most frequent;y occurring value or mode of the residence time. This is implemented in R using a user defined function called *getmode.credit().* This function has been implemented using built-in functions in R such as *unique(),match() , tabulate().*
- Age attribute has some outlier values which can influence the analysis. A customer of age 99 and has a loan reason of new car could be an outlier as the chances of that occurrence is very less. As the data is skewed the age value of 99 is imputed with the middle most occurring value or median of age.
- Missing values were checked in the data using *is.na().*A total of 22 missing (NA) values were found in the raw data from the attributes- employment, housing and personal status. As no associations could be found for imputation and s the number of missing values is less significant compared to the whole data set, these missing values were removed from the data set using *drop_na().*
- The ID column was removed from the data set as it does not give any meaningful information and it does not have any association with the rest of the variables.
- There are also 2 categories in Employment long & Long. This must have been a human entry error and they must be the customers with same status.

The dimension of the data set was checked after data cleaning using dim() and head(). The data has 769 records and 13 attributes.

5.SPLITTING THE DATA

After all the pre-processing is completed, training the model becomes the first step in making good predictions. Also, it is necessary to prevent the model from overfitting. That is, the model learns too much about the training data such that it performs well, but when an unseen data comes, it fails to make good predictions. Hence the ratio of splitting the data has to be carefully considered based on the use case. But as a general practice, 70-75% of data is selected as training data set randomly. Randomisation of selection is important so that the model does not pick up any unwanted errors or patterns from the data while training.

Sample() in R is used for this selection of training data and a size of 0.75 is mentioned as a parameter in this function to get 75% of data . All the data that is not present in the training data is taken as test data (25%)

A *seed()* was set with a unique number before the selection of training data so that the same random numbers are generated each time while the code runs and the same training and testing data is selected for model building to give constant results.

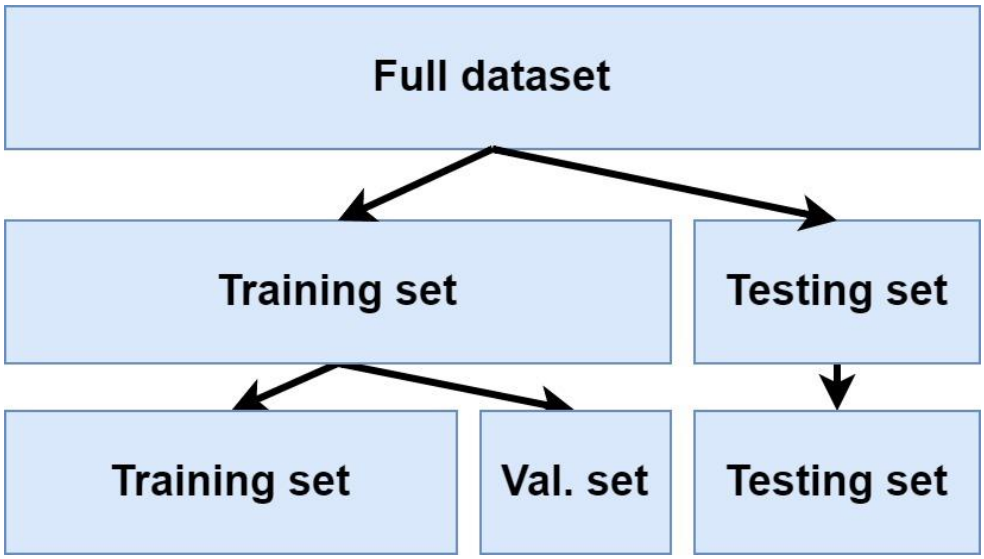


Figure 5.1: splitting data set into train test and validation sets

The figure 5.1 shows the general methodology of training and testing the model. A validation dataset is sometimes used to make sure that the training data gives the right results.

6.MODEL BUILDING

As it is required to categorize the target variable into two categories using the predictor variables, classification algorithms become the most suitable one for predictions. Classification is a process in which a set of predictor variables are used to predict the class of a target variable. Numerous classification models are available such as Logistic Regression, Naïve Bayes, K-Nearest Neighbours, Decision Tree, Random Forest, Support Vector Machine etc. Each algorithm uses a different approach and has its own advantages and disadvantages. One of the most commonly used algorithms is Decision Tree algorithm

6.1 Decision Tree Algorithm

In a tree-based method, the predictor space is segmented into a number of simple regions. This segmenting is done by a set of splitting rules and criterions. As the process of decision making based on rules to segment the predictor space can be summarized and visualized in the form of an inverted tree starting from root to branches to leaves, such methods are known as decision-tree methods. The decision trees use a top-down approach to the data set.

Advantages and Disadvantages

Tree based methods are often simple, can be visualized and can be explained easily. Also, decision tree algorithms require very less data preparation and does not get effected by missing values. It handles nonlinearity seamlessly. However, it is not robust and a small change in the

data can cause a large change in the structure of the tree. It is prone to overfitting of training data. Also, sometimes the decision tree calculations can get very complex and can be relatively expensive compared to other algorithms as complexity and time increases.

Types of Decision Trees

Decision tree can be applied in both classification and regression scenarios. As the problem statement is to predict a qualitative response rather than a numerical response, classification tree is used to build the model

6.2 Methodology

A classification decision tree begins with a root node, and it branches into different nodes based on the questions or conditions presented to the tree. The internal nodes present within the tree can be thought of as various conditions to split into branches or decisions. Each branch is a subsection of the entire tree and it. Every subsection is independent on its own. The edge of a tree-based model is the outcome of the questions, and these outcomes are represented by leaf nodes. Each leaf node actually represents the target class distribution.

Following are the steps that is used in making the top-down induction of decision trees:

1. Decide dataset and target label
2. Try different splits on attributes
3. Finalize the split based on the best value of the metric
4. Split again based on new attribute
5. Stop when criteria are met.

There are several metrics that are calculated at each node to make decisions in a tree. Entropy, Gini index and information gain are the most commonly used metrices.

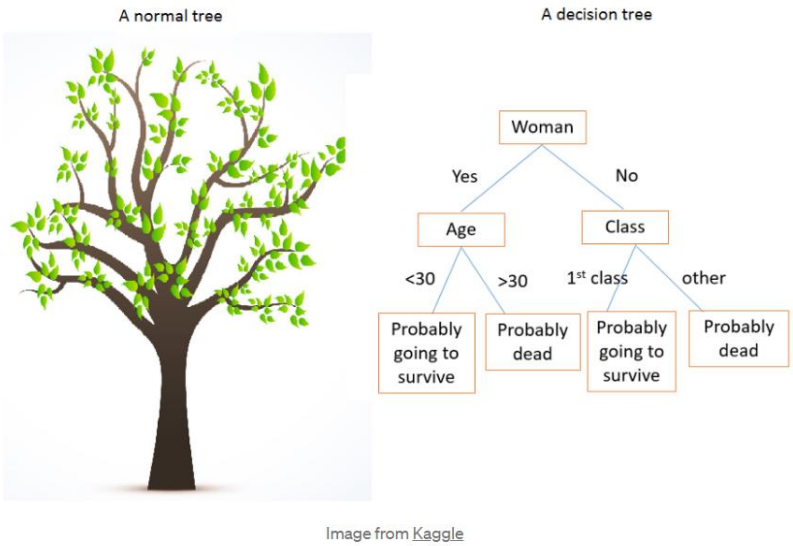


Figure 6.1: A comparison of a physical tree and decision tree using a sample data set from kaggle

The above figure shows how the survival of a woman based on her age and class of travel in the titanic data set is calculated by a classification decision tree algorithm and its similarity with a physical tree.

6.4 Entropy and Information Gain

Entropy is the measure of randomness in a system. In machine learning scenario, it is the measure of sum of uncertainty of a given data. In decision tree algorithm, an entropy value describes the degree of randomness of a particular node. On a node, entropy of each predictor variable is calculated by determining how purely it can classify the target variable. In essence it is the sum of probability of classifying the target variable by each subcategory in an attribute. The higher the entropy, the higher will be the randomness in a dataset. And as entropy increases the classification becomes more impure. Usually when building a decision tree model, a lower entropy would be more preferred as it can make decisions more clearly.

Example- If attribute 1 can classify credit standing (target variable) of customers in a 50-50 ratio, the decision making, and prediction of target class based on this attribute will be very difficult for the decision tree as the entropy and randomness is very high. Whereas if the attribute 2 can classify the target class more purely then decision making would get easier.

So, at every node we calculate the entropy of each attribute and the attribute with the lowest entropy becomes the splitting criterion. Information gain is a metric that is used for measuring the reduction in entropy after the data set is split on an attribute. The attribute that gives highest entropy difference in a node has the highest information gain. The formula for entropy is given as below:

$$E = -\sum_{i=1}^{N} p_i log_2 p_i$$

Figure 6.2: Formula for Entropy

Where P_i is the uncertainty of a category and index (i) refers to the number of possible categories. Logarithm of a decimal number is negative. To nullify it there is a negative symbol in the equation.

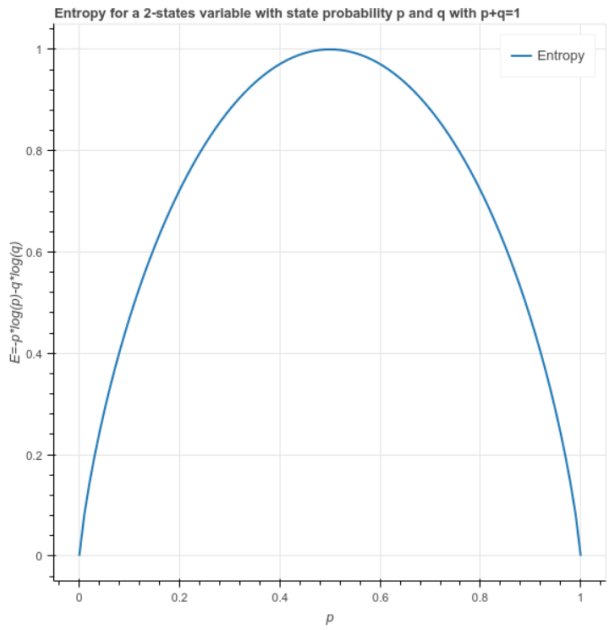


Figure 6.3: Graphical representation of entropy function with respect to probability values

The above graph shows how entropy increases as probability of prediction of a class goes to 0.5 (maximum uncertainty) and how entropy decreases as probability approaches to 0 or 1 (minimum uncertainty).

7.FINDING ROOT NODE SPLIT USING ENTROPY

Using only categorical training data

Root node is the initial node that starts the classification process in a tree. Following are the steps to find the root node using entropy method:

1) Calculate the entropy of the target variable for the training data
2) Calculate the entropy of each predictor variable in classifying the target variable
3) Obtain the information gain of each predictor variable by subtracting its entropy with the target variable's entropy.
4) The variable with the most information gain or least entropy will be the root node.

Finding entropy of target variable

Out of 576 records in the training data, the Credit standing variable has the following class distribution.

Good	Bad
336	240

Good	Bad
0.583	0.416

Table 7.1 and 7.2: Number and proportion of customers in target labels

Entropy is calculated as below:

Entropy (E) = (-P(Good) * $\log_2$ P(Good)) + (-P(Bad) * $\log_2$ P(Bad))

= (-0.583 * $\log_2$ (0.583)) + (-0.416 * $\log_2$ (0.416)) = **0.979868**

In R, this is calculated with the help of contingency tables by using table (), prop.table() and sum() to get the result

Finding entropy of each independent variable

An example of checking Acct is used below to show the calculation of entropy. Checking Acct has 4 categories- 0Balance, High, Low, No Acct and its classification of target variable credit standing as Good and Bad is as follows:

	Good	**Bad**
0Balance	98	104
High	13	14
Low	74	65
No Acct	151	57

	Good	**Bad**
0Balance	0.4851	0.5148
High	0.4814	0.5185
Low	0.5323	0.4676
No Acct	0.7259	0.2740

Table 7.3 and 7.4: Number and proportion of customers in checking acct w.r.t target labels

Generally, the entropy at a level is not simply the sum of entropy at each leaf, rather it is the weighted sum of entropy in each leaf. The weights are given so that the contribution to entropy from each class is according to the data points contained in them. In the example of checking acct, No Acct class should have more weightage as it has a greater number of data points. The weight of No Acct class is (151+57)/576, where 576 is total number of records in the training data.

Entropy (E, X) = Entropy (Credit Standing, Checking Acct)

= P (0Balance) * E (98,104) + P (High) * E (13,14) +

P (Low) * E (74,65) + P (No Acct) * E (151,57)

= 202/576*[(-0.4851* $\log_2$ (0. 4851)) + (-0.5148 * $\log_2$ (0.5148))] +

27/576*[(-0.4814* $\log_2$ (0. 4814)) + (-0.5185* $\log_2$ (0. 5185))] +

139/576*[(-0.5323* $\log_2$ (0. 5323)) + (-0.4676* $\log_2$ (0. 4676))] +

208/576*[(-0.7259* $\log_2$ (0. 7259)) + (-0.2740 * $\log_2$ (0. 2740))]

= **0.9438223**

Information Gain = Entropy (E) - Entropy (E, X) = 0.979868 - 0.9438223 = **0.0360457**

0.0360457 is the information gain obtained when we look at the entropy of Checking Acct with the target variable. Similarly, the same process is conducted for all the attributes to find the one with the minimum entropy or maximum information gain.

Credit_entropy_node()

In R, entropy calculation is implemented by using a user defined function that uses contingency tables (*table ()*, *prop.table()*) to calculate probability values for each class of attribute and *sum() and rowSums()* for summation. The element wise operations of each element in contingency tables makes the process easier in R. The entropy values of all the attributes are obtained using an iterative approach. That is, calling the user defined function for each attribute and calculating entropy. These entropy values are inserted into a vector. The minimum value of entropy present in the vector corresponds to the attribute having minimum entropy.

After the implementation of this function, the attribute **Credit History** was found out to be the one with the minimum entropy of **0.71681.** Entropy of each attribute was subtracted from the total entropy of target variable (credit standing). This gives information gain.

Information Gain (G) = Entropy(E) - Entropy(E, X)

Where X is each attribute. The information gain was calculated to be 0.979868- 0.71681 = **0.26306.**

Hence **credit history is the root node.** This supports the exploratory data analysis conducted in the earlier stages as credit history was the attribute that clearly classifies the target variable more purely.

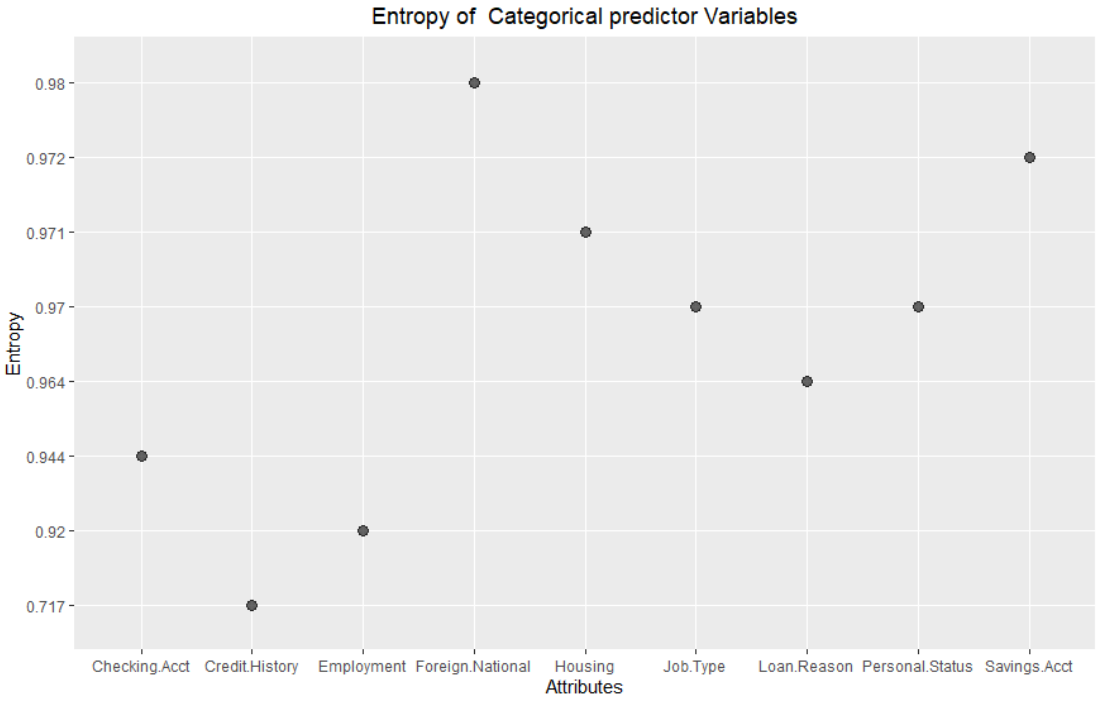


Figure 7.1: Entropy values of categorical variables in the initial data set

The figure 7.1 clearly shows that credit history followed by employment and checking acct has the least entropy.

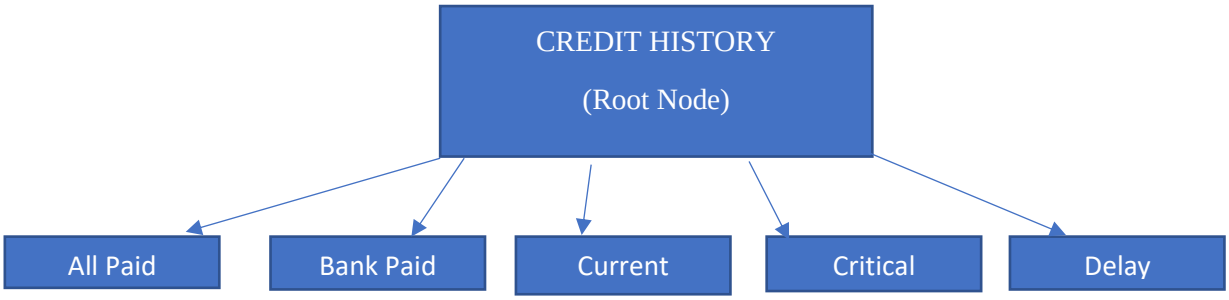


Figure 7.2: Branching of credit history into different possibilities

8.IMPLEMENTATION OF BINARY SPLIT

A decision tree can be split into too many possibilities in each node based on the number of classes in each predictor variable. This often leads to a very complex decision tree model that is hard to interpret the results and also takes more time in implementation. A binary decision tree is a simpler model that classifies each attribute present in a node only into a binary decision. Each decision leads to one of the two possibilities and there cannot be more than two possibilities. Each decision leads to another binary decision and finally it leads to a prediction in the leaf node.

If the attributes present in the dataset have more than two classes, it is necessary to combine some of them in a logical way so that there will only be two classes in each attribute.

The goal is to categorise or combine classes in each attribute such that the resultant attribute has less entropy. That is, the new classes should classify the target variable more purely with less entropy. Therefore, randomness in the data will be reduced. At the same time the categorization should be such that there should not be a significant imbalance in the number of data points between the new classes in the attribute.

The following categorical variables were combined into binary class from multiclass. The process has been done using a series of logical statements inside *mutate()* from *dplyr* package.

The decision of combination was made by looking at the purity of splitting the target variable (Credit Standing) by each class in a category and the total number of data points in each category. This was done in R using contingency tables by printing *table()* and *prop.table()* of each feature with target variable.

1) **Checking Acct**
   Customers with No Account have more than 70% of good credit (low entropy) standing and the number of customers with no acct is also high. Hence such customers alone are in one category so that the algorithm can classify target variable more purely. Customers with 0Balance, Low &High Balance have fairly equal number of good and bad credit standing. So, they have been combines as given below:
   - No Acct → No Acct
   - 0Balance, High, Low → Yes Acct
2) **Credit History**
   Customers with All paid, Bank Paid and Current credit history have no payments due and most of them have good credit standing class. Hence, they are categorized into one

class as paid. Customers with Delay and Critical credit history are risky accounts as they have credits to be paid at other banks or defaulted the payments or delayed it in the past. Hence, such customers have very high Bad credit standing class in the target indicating very low entropy. Hence, they are combined into another category as Not Paid. Splitting customers into paid and not paid categories will result in better classification by the model as randomness is reduced.

- All Paid, Bank Paid, Current → Paid
- Critical, Delay → Not Paid

3) Loan Reason

There are 10 categories in Loan Reason. Except education, all other categories have more than 50% of good credit standing. Business, Large Appliance, Other, Repairs, Retraining have very high percentage of good credit standing. Also, such loan reason belongs mostly to some kind of a business. Hence, they have been grouped to one category. Rest of the loan reasons are mostly for household. So, they have been grouped into second category.

- Car New, Car Used, Education Furniture, Small Appliance → Household
- Business, Large Appliance, Other, Repairs, Retraining → Business

4) Savings Acct

Most of the customers have Low Savings Acct. Customers with MedHigh and MedLow savings account have high proportion of good credit standings (low entropy) and customers belonging to Low and No Acct classes have almost equal proportion of good and bad credit standing (higher entropy). Customers with High Savings acct also has 56% good credit standing and hence they have been combined with the medium category.

- No Acct, Low → Low.NoSavAcct
- MedLow, MedHigh, High → Med.High.SavAcct

5) Employment

Customers with short and very short employment have high proportion of bad credit standing while customers with medium and long-term employment have high proportion of good credit standing. Hence it is easier to separate short and very short into one category and long and medium into another category having low entropy. Customers with unemployed status also have very good proportion of good credit standing. Hence, they are added to the category with long term. Also, there are two customers who are retired. That record has also been added to the long category. There are also 2 categories long & Long. This must have been a human entry error and they must be the customers with same status.

- Short, Very Short → Short
- Long, long, Medium, Retired, Unemployed → Long.Unemployed

6) Personal Status

Single Customers have more proportion of good credit standing (less entropy) than married or divorced customers. Hence, they have been classified as:

- Single → Single
- Married, Divorced → Married.Divorced

7) **Housing**

Most customers have their own housing. Customers having their own housing have higher proportion of good credit standing and hence making them a separate category would yield in a lower entropy and better classification by the model. Customers having housing status as Rent & Other have almost equal number of good and bad credit standing classes and hence, they are combined into a separate category.

- Own → Own
- Rent, Other → Rent.Other

8) **Job Type**

Most of the customers come under skilled category. Customers having the job type of management, skilled and unemployed are categorized into one as they have higher proportion of good credit standing and can classify target variable more purely. Customers with unskilled job type have been classified into another category as the number of customers with good and bad credit standing is not that different.

- Management, Skilled, Unemployed → Skilled. Unemployed
- Unskilled → Unskilled

9) **Foreign National**

There were only two categories in this attribute and hence further combinations in sot required for binary split.

After implementing the binary split of categorical variables, they were converted into factor form in R so that each attribute has only 2 levels. The entropy was again calculated for all the binary split categorical variables in the training data using the same user defined function **credit_entropy_node**

After the implementation of this function, the attribute **Credit History** was found out to be again the one with the minimum entropy of **0.88576.** But this time the entropy of credit history has increased compared to the one before the binary split indicating that the splitting criterion performed may not be the optimum one. The information gain of each attribute was calculated by subtracting its entropy from the entropy of target variable in the training data. The information gain was calculated to be 0.979868- 0.88576 = **0.09411**. The information gain has reduced as compared to the one before the binary split. This shows that the information retrieved from the model has been reduced.

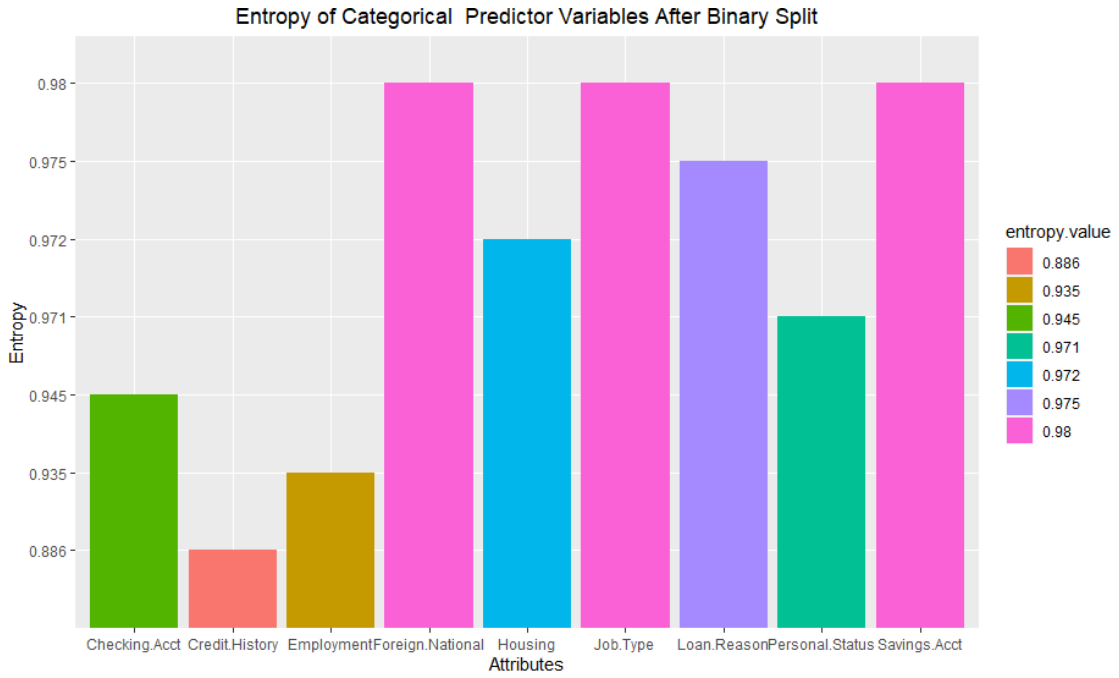


Figure 8.1: Entropy values of categorical variables after binary split shown as bar chart

It can be seen from the above figure that the entropy of all attributes has increased to an extend after the binary split after the categorization of attributes into binary. The current method is implemented considering purity of the split of the target variable and the logical aspects of each class in the attribute. More proven algorithm needs to be implemented in finding the optimum binary combination of each attribute.

9.INCLUSION OF NUMERIC VARIABLES

So far, the entropy calculation was implemented only to numeric variables in the training data. But there are also 3 numeric variables which needs to be included in the classification process. As numeric variables include several distinct continuous values and not classes, it is important to convert them into binary classes before using them in classification decision tree algorithm.

In R, this process is implemented using cut(). Cut() is used to divide the range of values present in a continuous into intervals. Hence binning of numeric variables into categories can be implemented and labels can be specified for these bins to make them in a factor form.

Implementation of Algorithm to get the least entropy

1) A user defined function is created which takes 2 arguments. 1.The value at which the binning of numeric variable has to occur. 2. The column number of the attribute in the data set
2) Calculate the maximum value of the numeric variable for iteration.
3) Initialize two empty vectors to capture entropy values and the value at which binning has occurred
4) Implement a loop starting from initial value to the maximum value of a numerical variable with a step of our choice. This is implemented using seq() in R
5) Call the user defined function for each value of the iteration.

6) The function will use passed arguments to bin the numerical variable.
7) This binned numerical variable is given into the entropy calculations that use probability values with the target variable with contingency tables
8) The entropy values returned at each cut is stored in the vector initialized.

1) Age

Age ranges from 18-85 and it needs to be binned. After the implementation of the aforementioned algorithm, following were the result.

	Age	Entropy Value
1	18	0.97852
2	20	0.97980
3	22	0.97936
4	24	0.97214
5	26	0.97691
6	28	0.96940
7	30	0.97348
8	32	0.97533
9	34	0.97830
10	36	0.97688
11	38	0.97886
12	40	0.97883
13	42	0.97790

Figure 9.1: Entropy of Age at different bins

It can be seen from figure 9.1 that age 28 has the least entropy and hence that has to be criteria for binning.

- Age less than or equal to 28 → Young
- Age greater than 28 → Adult

2) Months since Acct opened

By implementing the algorithm, the minimum entropy of 0.96871 was obtained at 12 months. Therefore, Binning was done as

- Months less than or equal to 12 → less than one year
- Months greater than 12 → More than one year

3) Residence Time (in current district)

After running the algorithm for all the residence time, the minimum entropy of 0.97717 was found out to be for the value 1. So, categorization was done as

- Less than or equal to 1 → Less than 1 year
- Greater than 1 year → Greater than 1 year

Finally, entropy was again calculated for all the binary split variables (categorical & numeric) in the training data using the same user defined function **credit_entropy_node**

After the implementation of this function, the attribute **Credit History** was found out to be again the one with the minimum entropy of **0.88576, followed by employment.** The information gain of each attribute was calculated by subtracting its entropy from the entropy of target variable in the training data. The information gain was calculated to be 0.979868-0.88576 = **0.09411**

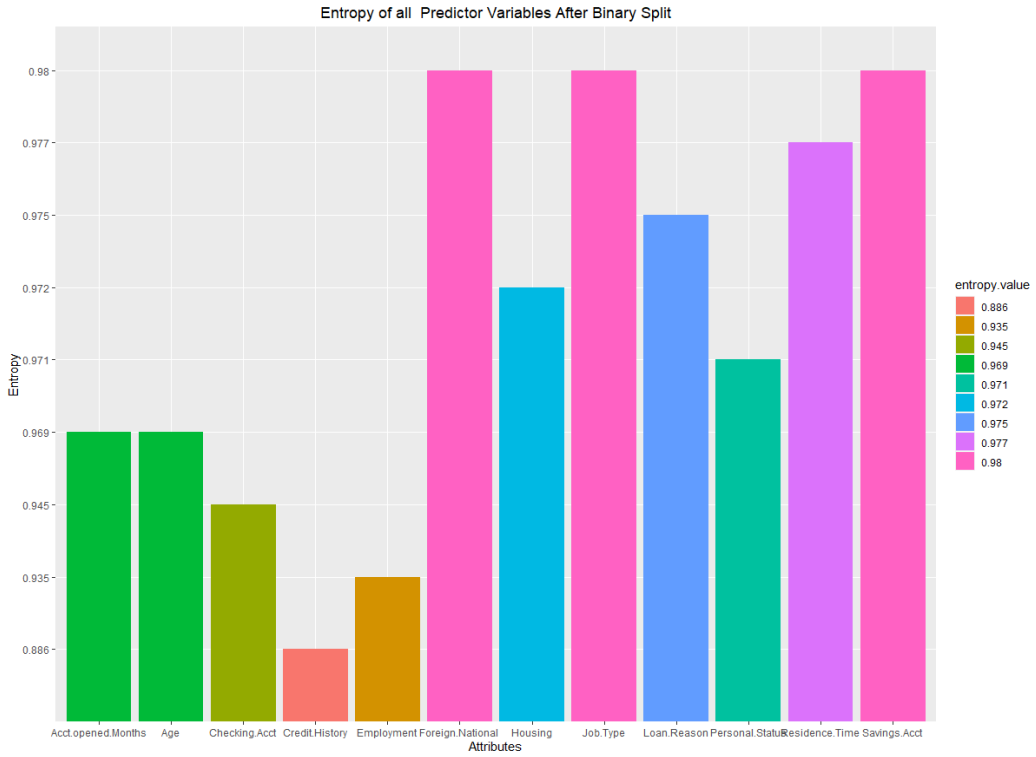


Figure 9.2: Entropy values of all variables after binary split shown as bar chart

The entropy of Age and Acct opened months was reduced to a far extend due to the implementation of algorithm. But still the most important features for classification were found to be credit history, employment and checking acct.

Tree after Binary split

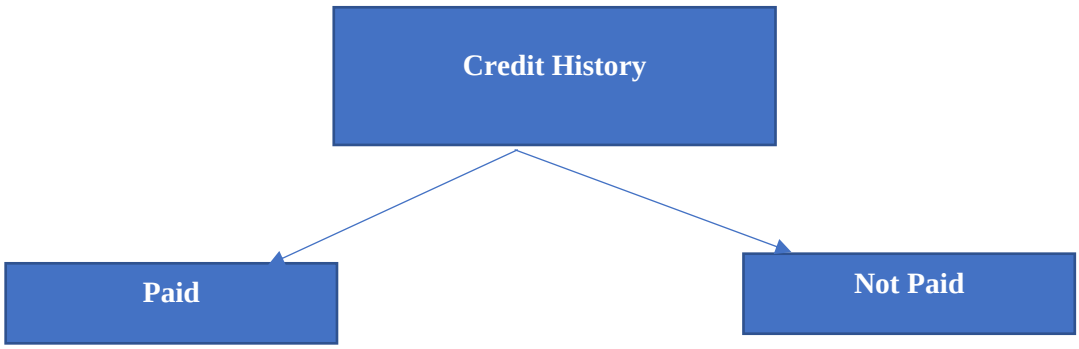


Figure 9.3: Branching of root node into two branches

10.EVALUATING SECOND LEVEL SPLIT

The root node (Credit History) found in the earlier stage has to be split into 2 categories- Paid and Not Paid by the binary classification decision tree. After this, entropy for all attributes in each category is calculated separately to get the next node for split. To implement this using R following steps are followed.

Create subtree T1

1) IF Credit History==’Paid’

   THEN create a new data frame containing only ‘PAID’ category

2) Remove the homogeneous credit history column of ‘PAID’ from the data frame
3) Calculate the entropy before the split using entropy formula for the target variable
4) Calculate entropy for all the predictor variables iteratively using entropy formula implemented in the previously developed user defined function credit_entropy_node()
5) Capture the return value of entropy from the function for all attributes in a vector.
6) Find the minimum entropy and the corresponding attribute name. This attribute will become the node for the next level split
7) Subtract the minimum entropy found from the total entropy calculated in stpe3.
8) Entropy before the split- Entropy after the split gives the information gain.

Create subtree T2

1) IF Credit History==’Not Paid’

   THEN create a new data frame containing only ‘Not Paid’ category

2) Remove the homogeneous credit history column of ‘Not Paid’ from the data frame
3) Calculate the entropy before the split using entropy formula for the target variable
4) Calculate entropy for all the predictor variables iteratively using entropy formula implemented in the previously developed user defined function credit_entropy_node()

5) Capture the return value of entropy from the function for all attributes in a vector.
6) Find the minimum entropy and the corresponding attribute name. This attribute will become the node for the next level split
7) Subtract the minimum entropy found from the total entropy calculated in stpe3.
8) Entropy before the split- Entropy after the split gives the information gain.

On implementing Tree T1 on to the left side of root node, there were 455 data points. The entropy before the split was found to be 0.9100256. The attribute **Employment** was found out to be the one with the minimum entropy of **0.84513.** The information gain of each attribute was calculated by subtracting its entropy from the entropy of target variable before the split. The information gain was found to be the maximum for Employment

Information Gain = 0. 9100256- 0. 84513 = **0. 0649**

Hence Employment is the next level of split on left side of the root node.

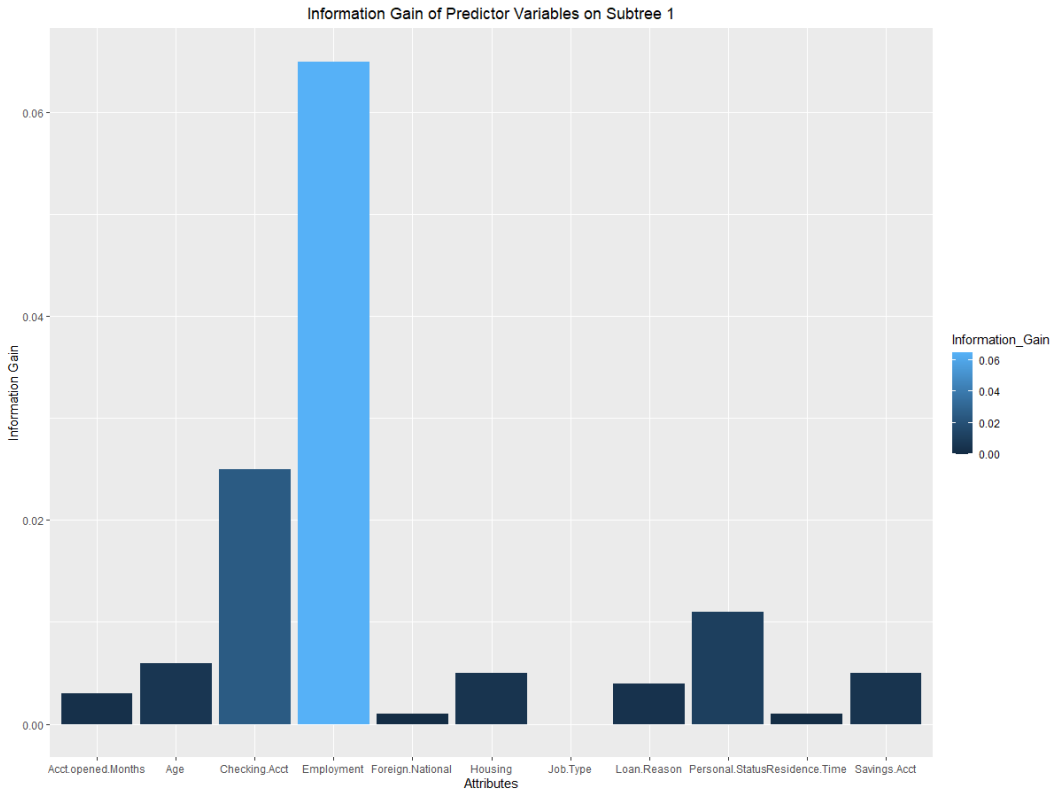


Figure 10.1: Information gain obtained at child node 1 – credit history paid category

On implementing Tree T2 on to the right side of root node, there were 121 data points. The entropy before the split was found to be 0.7944901. The attribute **Checking Acct** was found out to be the one with the minimum entropy of **0.70634.** The information gain of each attribute was calculated by subtracting its entropy from the entropy of target variable before the split. The information gain was found to be the maximum for Checking Acct.

Information Gain = 0. 7944901- 0.70634= **0. 08815**

Hence Checking Acct is the next level of split on right side of the root node.

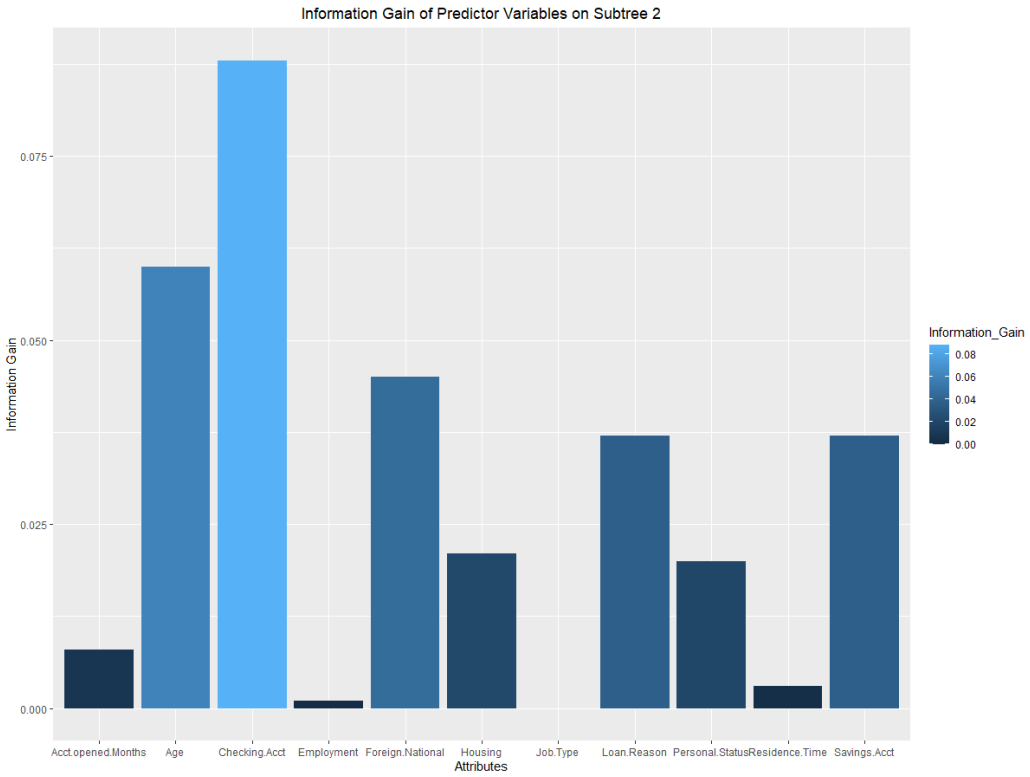


Figure 10.2: Information gain obtained at child node 2 – credit history not paid category

Decision Tree after Second Split

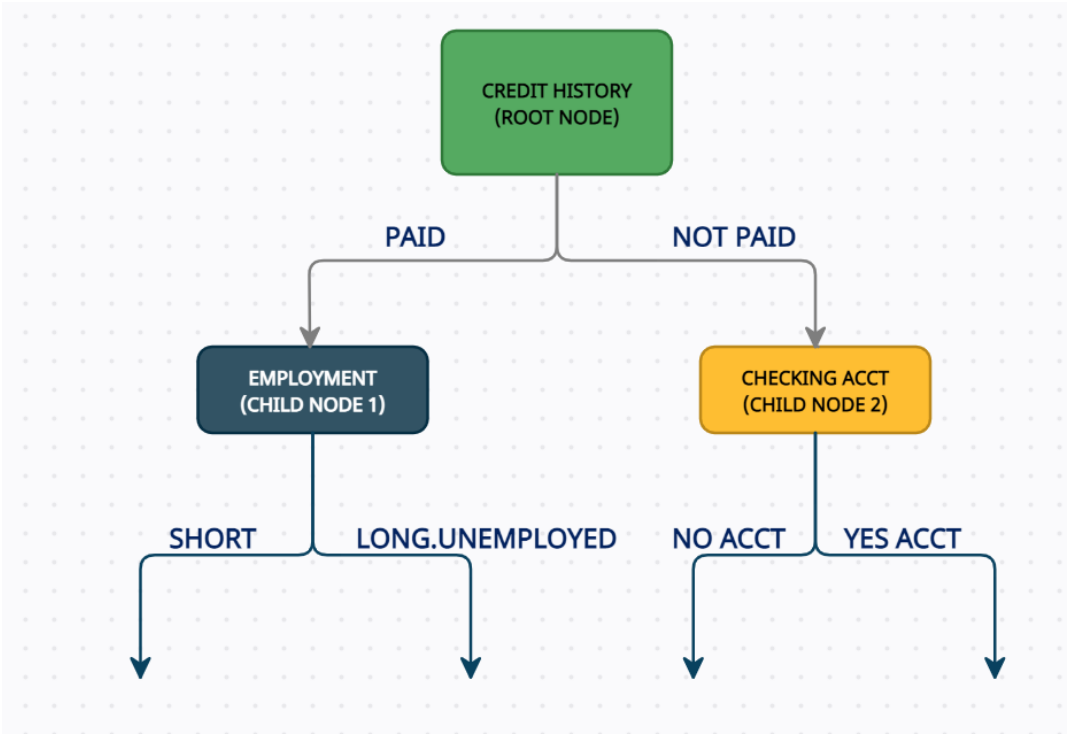


Figure 10.3: Branching of decision tree by entropy calculation

11.IMPLEMENTATION OF DECISION TREE USING PACKAGE TREE

Decision tree is about testing an attribute and branching the cases based on the result of the test.

- Each internal node corresponds to a test
- Each branch corresponds to the result of a test
- Each leaf node assigns a classification

11.1 General Algorithm and Implementation

1) Choose an attribute from the dataset
2) Calculate the significance of attribute in splitting the data
3) Split data based on the value of best attribute. Best attribute can be the one with more predictiveness, less impurity, low entropy etc
4) Go to step 1- Go to each branch and repeat for each attribute.

There are several built in packages in R for the implementation of decision tree. Some of the commonly used functions include ctree (), dtree (), rpart () etc. Here, decision tree is implemented using tree(). The tree is implemented by using a binary recursive partitioning. This is done by using the left-hand-side response variable given in the formula expression inside the function. When a classification tree is implemented, this response variable should be a factor. The number of splits for the tree is chosen from the terms given in the right-hand side of the formula. The right-hand side can have both numeric and factor variables separated by + symbol. The splitting criteria is chosen by the package such that it maximises the reduction in impurity and this process is repeated. The growth of the tree is limited to 32 levels by the tree () package in case of a factor predictor variables or a classification tree. The commonly used arguments are:

- Formula – The expression to implement decision tree in tree (). The left-hand side is the response variable, and the right-hand side is the predictor variables separated by '+' symbol. To exclude a predictor variable from the model '-' symbol can be used. It is also a common practice to use '.' symbol to include all predictor variables in the formula. Symbol '~' is used to separate predictor and response variables in the model.
- Data – The data frame to be used to implement the model and formula
- Subset- Subset of the data frame to be used in model. E.g., The training subset of the data
- Model- If a data frame is given as input to this argument, formula and data arguments are ignored and this argument will be used define the model.
- Split- Splitting criterion for decision can be given here
- Control- The list returned by the parameter tree.control is used here. Tree.control can have arguments like mincut, minsize to determine the minimum size of data points required for each split.

To implement decision tree model, first the data set after EDA was split into train and test data sets. Although missing values would be handled by decision tree, the EDA completed dataset is given as input to the tree and the missing values are omitted. 75% of data was taken as the training set and the rest 25% of the data became the test set for the model. A seed value was set while implementing sample() in R to get the same train and test data in each run. The sample() function randomly picks up data from the data frame according to the size given. Here it is 0.75 for 75% of train data. All the 576 records were given as the training data to the model.

tree.credit = tree(Credit.Standing~., data=credit_train_tree)

A model tree.credit was implemented with credit standing as response variable and all the other attributes as predictor variables. Training data was given as the input to the model.

Summary of the model outputted using summary (tree. credit) gave following result:

1) Important variables used in the model - "Credit.History","Employment" "Residence.Time", "Loan.Reason", "Savings.Acct", "Age"
2) The number of terminal nodes or leaf nodes that classify credit standing as target was found to be 10
3) Residual mean deviance which is the sum of square of residuals divided by the number of terminal nodes were found to be 0.8343. Here since it is a binary classification tree, it is the binomial deviance. The lower the deviance the better the fit of the tree to the training data.
4) Misclassification error rate was found to be 0.2118. This refers to the number of observations wrongly classified by the model by the total number of observations. Since this is a supervised machine learning model, actual response variables are obtained from the data and the proportion of observations present in the training data predicted to fall in a different class than they actually did can be calculated. Hence, the training error rate is found out to be 21% which is not too good or bad.

Printing the model gave the following result:

The root node and the most important attribute was found to be credit history. Critical class of credit history was found to be classifying Bad credit standing with 98.6% accuracy. This belonged to the left side split of root node. This node can be considered as a pure node as the accuracy of predicting the correct category of target class is close to 100%. The right side was composed of the other categories of credit history attribute, and they were able to classify Good class of response variable with 66.6% accuracy.

In the next split, Current and Delay categories of credit history was split into the left-hand side with 55.8% accuracy for Good and in the branch followed that, Employment 'Short' was used to split Bad classes with 63.9% accuracy. In the right-hand side all paid and bank paid categories went into further branch with 78% accuracy for Good class. Savings Acct became the child node after this. This process continued to get a binary classification decision tree.

One of the most important feature of decision tree package is that they can be graphically plotted and hence easily explainable. Plot () displayed the structure of the tree and text() was used to display the labels at each nodes. Misclass.tree() of the model gives the number of misclassifications happened at each node of the decision tree

The figure 11.2 below shows the resultant decision tree from the training data

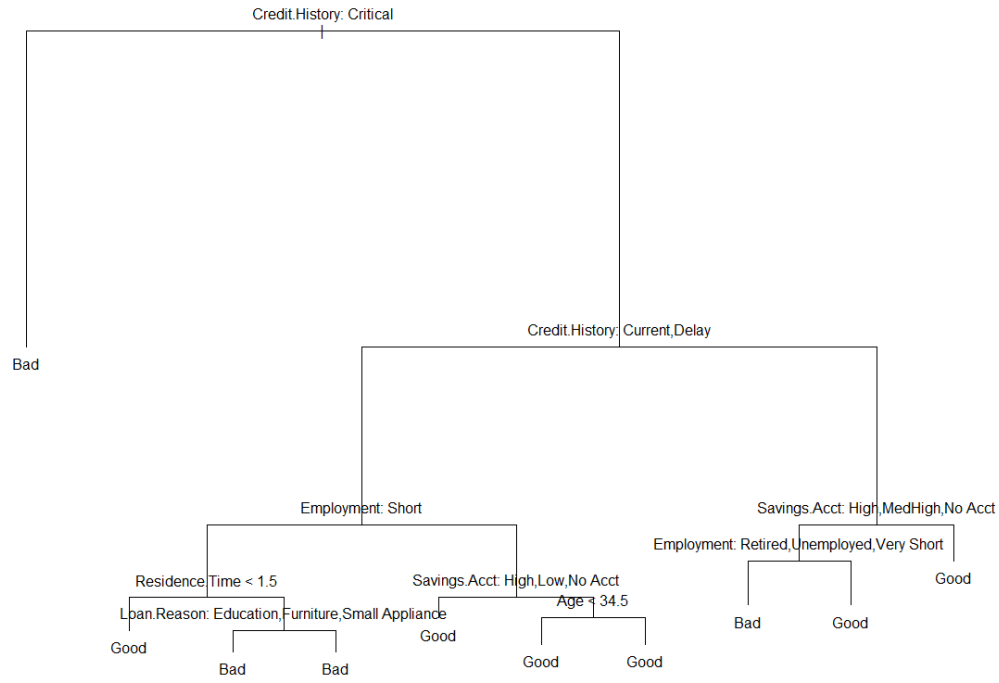


Figure 11.1: Decision tree with 10 terminal nodes

11.2 Model Prediction and Evaluation

In order to evaluate the performance of a decision tree, it is necessary that we estimate the test error of the model. Hence, after we build the decision tree using the training data it is required to estimate the performance of the model using the test data. The test data is the unseen data by the model and the performance of the trained model on this unseen data gives the actual performance metrics. For this purpose, predict() is used. In R, the trained model and testing data is given as arguments to the predict(). Type=’class’ is another argument that allows R to return predicted classes of the response variable into a factor variable.

tree.pred = predict(tree.credit,credit_test_tree, type="class")

This factor tree.pred containing the predictions is compared with the original classes of response variable (credit standing) to get the accuracy of predictions. That is, for how many observations the model gave the correct classes compared to the actual labels. The accuracy of tree was found to be 77%.

Caret package was used to produce the confusion matrix in R. Confusion matrix is a table that is used to evaluate the performance of a classification model. On printing confusion matrix, we get the following result

	Reference / Actual	
Prediction	**Bad (Positive)**	**Good (Negative)**
Bad (Positive)	46 (TP)	11 (FP)
Good (Negative)	32 (FN)	104 (TN)

Table 11.1: confusion matrix of decision tree model

The reference is the actual class labels from the test data and prediction is the predicted labels.

In any classification model there would be a positive class and a negative class. Here positive class is defined as 'Bad' credit standing and negative class is defined as 'Good' credit standing.

True positive (TP) → Truly predicted as positive class. That is both predicted and actual classes were positive class. Here 'Bad' class

True negative (TN) → Truly predicted as negative class. That is both predicted and actual classes were negative class. Here 'Good' class

False positive (FP) → Falsely predicted as positive class. That is the actual labels in training data was negative class (Good), but the model predictions were positive class (Bad)

False negative (FN) → Falsely predicted as negative class. That is the actual labels were positive class (Bad), but the model predicted labels were negative class (Good)

Accuracy of the model is given by $\frac{TP+TN}{TP+TN+FP+FN} = \frac{46+104}{46+104+11+32} = 77.72\%$

Accuracy metric depicts how many were correctly predicted by the model out of the total predictions. Here accuracy of 77.72% is a fairly good accuracy. This is much better than the baseline accuracy rate of 58.6% in the data set.

Recall/Sensitivity= $\frac{TP}{TP+FN} = \frac{46}{46+32} = 58.9\%$

Recall tells us how many actual positive classes were there in the entire testing data and out of that how many were predicted correctly. That is, how many Bad classes were predicted correctly out of all the Bad classes. This model does not give a good recall.

Precision=$\frac{TP}{TP+FP} = \frac{46}{46+11} = 80.7\%$

Precision tells how much good the model is in predictions. Precision talks only about predicted classes.

There is a trade-off between precision and recall metrics in machine learning model. When precision increases recall decreases. And if cut offs were changes to increase recall, precision goes down.

So, the amount of precision and recall required depends on the use case of the model. To understand this, we need to look at the number of false positive and false negative cases in the model predictions and which is more dangerous to the use case.

Here **Credit Standing classification, False Negatives are more dangerous.** Because in false negative, the actual label was positive or Bad class. But the predicted labels by the model were good class. That is, a person with Bad credit standing was predicted as good credit standing by the model. If a loan was issued to him, that will most probably result in a loss for the company. Hence, **FN has to be reduced and recall has to be higher.** The model here has a low recall, and it has to be improved. To do this we need to reduce the cut off. Sampling techniques can be used for this.

11.3 CROSS VALIDATION

Cross validation is a technique to measure the effectiveness of the model created. It is a re-sampling technique of the data to evaluate the performance of the model. Decision tree algorithm is very prone to overfitting. That is, the model will learn too much from the training data and picks up noise. This in turn results in poor performance with train or unseen data. Cross validation helps to avoid this overfitting by re-sampling the training and testing data again and again and thus reduces the testing error. There are different techniques for this. But one of the most common is K- Folds cross validation

K-Folds Cross validation

This technique is very easy to understand because every data point in our data set can come in both training and test set by re-sampling. This method becomes very important when we have less data points to train and test. It follows the following steps:

1) Split the entire raw data set randomly into K sets or folds. The value of K should not be too high or too low
2) Train the required model using k-1 data sets and test on the remaining fold. For instance, if the data set is divided into 6 folds, The first 5 folds become the training data and 6^{th} fold becomes the test data in first case. In the second case, 1^{st} fold becomes the test data and remaining becomes the train set
3) This process is continued until every fold becomes the test.
4) Finally, the average of results obtained in all the folds is calculated. This will be used to assess the model performance.

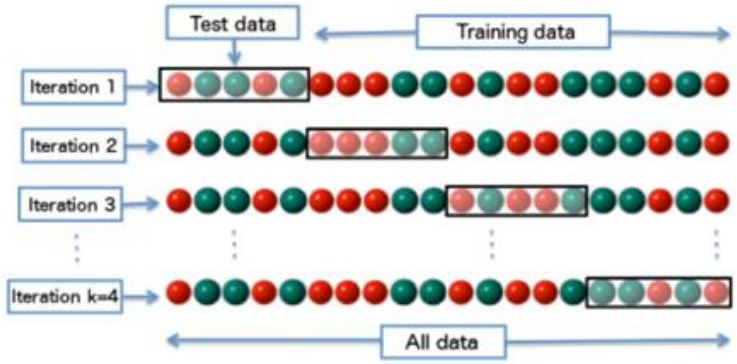


Figure 11.2: 4-fold cross validation. Image source Kaggle

11.4 PRUNING A DECISION TREE

It is necessary that a stopping criterion to be specified in a decision tree model. If this is not done the decision tree will fit all samples in the training data and it will lead to overfitting. Also, the complexity of the tree also gets higher. Pruning is a technique that is used to overcome the problem of overfitting. Pruning in its essence is to put some conditions on tree growth like the maximum depth of the tree, minimum number of data points for each leaf, minimum leaves for a split etc so that the tree does not grow in size. Pruning removes weakest rules and simplifies decision trees. Also, it makes it more adaptable to the data and reduces test error

Pruning can be done before and after the construction of decision tree. They are called pre-pruning and post-pruning. In pre-pruning each subtree growth is evaluated and stopped based on some measure. Example the information gain obtained from that node must be greater than threshold information gain or the depth of the tree must be less than or equal to the maximum depth defined

In post-pruning the full tree is constructed initially and later pruned in a bottom-up way. Measures such as Gini impurity or information gain is checked from the bottom decision nodes and the subtree with the least information gain is omitted or pruned. There are many pruning algorithms like reduced error pruning, cost complexity pruning, pessimistic pruning etc.

Cost complexity pruning works by calculating a score for each subtree. This score is comprised of residual sum of squares (RSS), hyper parameter *a*nd number of terminal nodes in a subtree (T). Optimum value of the tuning parameter (*a)* is found using cross validation. This tuning parameter controls the complexity of the subtree and the fit to the training data. First the score of all the subtrees is calculated and the one with the lowest score is taken. But again, this depends on *a* and cross validation. The process is repeated for different values of *a* and on an average the one that gives the lowest score is the best value of *a*. Finally, the one corresponds to the final value of *a* will be the pruned decision tree.

For implementing cross validation and pruning in R, cv.tree() is used to conduct K fold cross validation.

cv.credit <- cv.tree(tree.credit,K=10, FUN=prune.misclass)

This function accepts decision tree model and the number of folds for cross fold validation.

Prune.misclass parameter indicates that classification error rate has to be used to implement cross validation and cost complexity pruning. This will recursively snip off least important splits to get the pruned tree.

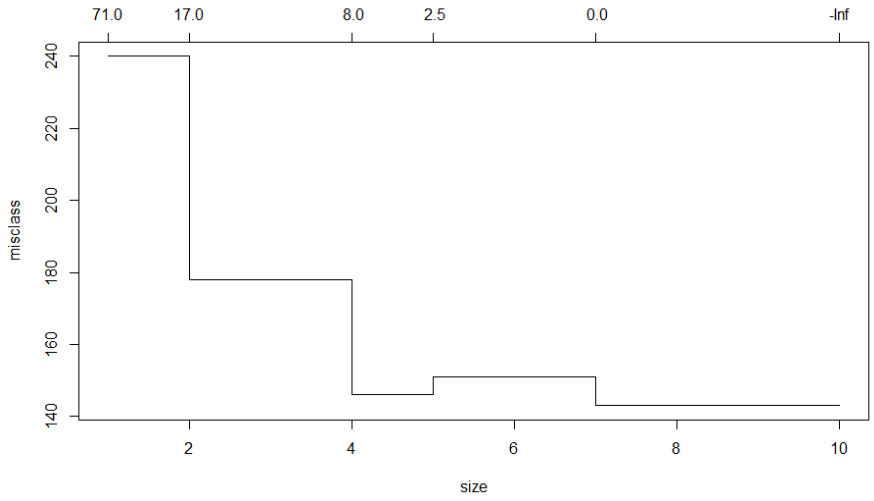


Figure 11.3: tree size (number of terminal nodes) vs misclassification

On plotting the pruned tree object, It is observed that misclassification is least for a tree with size 4. That is the number of terminal nodes would be 4.

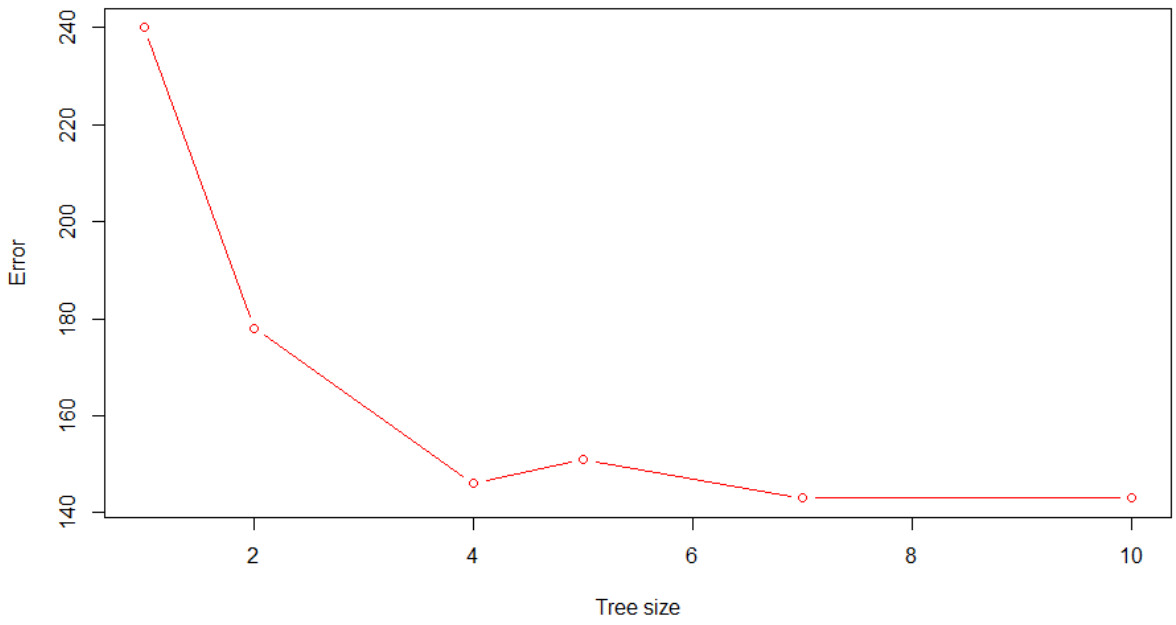


Figure 11.4: Decision tree size vs Deviance

On plotting deviance vs size of the tree, it can be seen that the error is least for the tree size of 4 and 8. But a tree size of 4 has much less complexity and hence it is the best size.

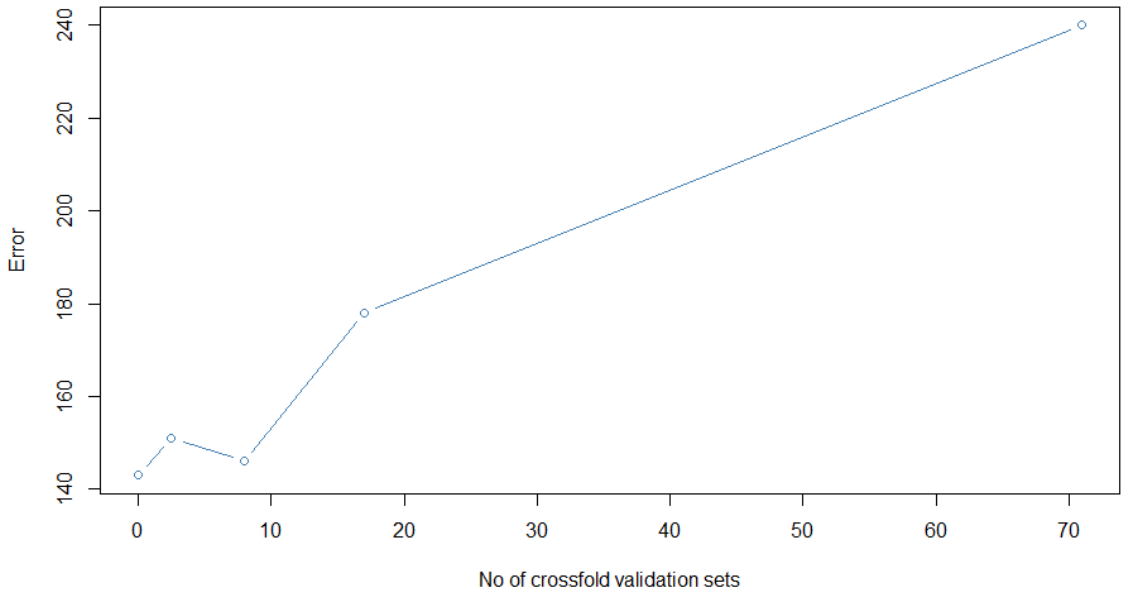


Figure 11.5: Number of cross validations sets vs Deviance

On plotting deviance vs number of cross validation sets (K), It can be seen that minimum error is for K=10. Therefore, for an optimum cross validation fold of 10, a tree size of 4 is the best pruned tree with least error.

After this pruning of the original tree is done with best size 4 using prune.misclass(). Om plotting this pruned tree, we get a shorter tree with only 4 leaf nodes.

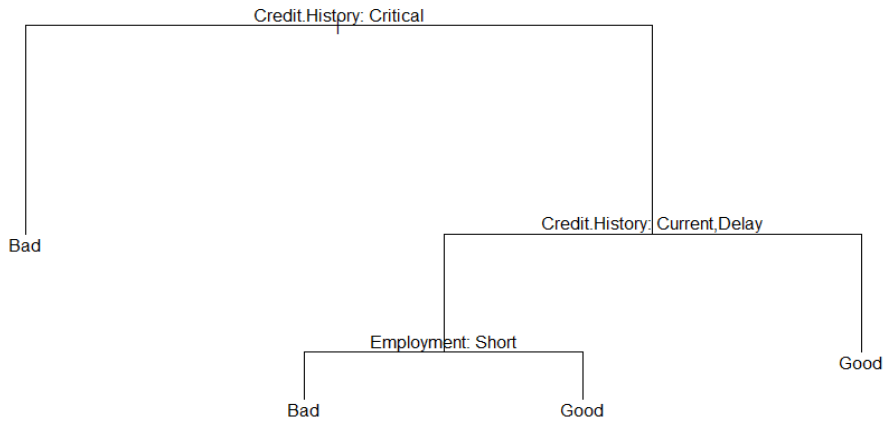


Figure 11.6: Pruned Decision tree with 4 leaf nodes

It can be seen that the new tree is very easy to interpret and uses only two significant attribute Credit History and Employment.

Now, prediction of this new tree is done using predict(). It gives the following confusion matrix.

	Reference / Actual	
Prediction	**Bad (Positive)**	**Good (Negative)**
Bad (Positive)	47 (TP)	12 (FP)
Good (Negative)	31 (FN)	103 (TN)

Table 11.2: confusion matrix of pruned decision tree model

The accuracy is still 77.7% after pruning. It has not increased but neither it has decreased. Nut the significance is that the same accuracy was obtained with a simpler tree. Hence the complexity of the tree has been reduced.

False Negative has reduced slightly by one observation and thus increasing the recall. But still, it is not a significant improvement. Hence better models such as random forest has to be used to improve the results.

11.5 Comparison of package tree with the entropy model tree

The tree generated from the package tree() and the tree created manually using entropy function have the same root node credit history. But tree function has taken critical category in credit history as a leaf node as it has almost 100% purity. The entropy function tree had delay category also in it and hence it was not fully pure to consider it as a leaf node. The second split in entropy tree was employment and checking account on left and right branches. The tree() also created the node as employment after credit history. This also indicates that the combination used in entropy tree to categorize customers with short employment as Bad was in fact correct because the package tree also did the same. Hence entropy and information gain calculation done is accurate. But the tree() further creates savings acct as the new node and branches into employment based on savings acct. But the entropy calculation showed very high entropy and low information gain on savings acct. Further analysis needs to be conducted on how savings acct became an important feature for decision tree using tree ().

12.RANDOM FOREST

Even though decision trees are easy to interpret, they are prone to overfitting and are not flexible. Random forest model combines the simplicity of decision tree and at the same time it gives flexibility too. This results in the improvement of accuracy of the model. Random forest consists of large number of individual decision trees and works as an ensemble technique, that is, as a group of trees. Random forest can be used for both classification and regression techniques. In a classification setting, each individual tree in a random forest outputs a response class prediction and the class with the majority vote out of all the trees becomes the model's output. Each tree in a random forest classifier will be good in evaluating some aspect of the data set. When all the trees combine, a better result would be obtained.

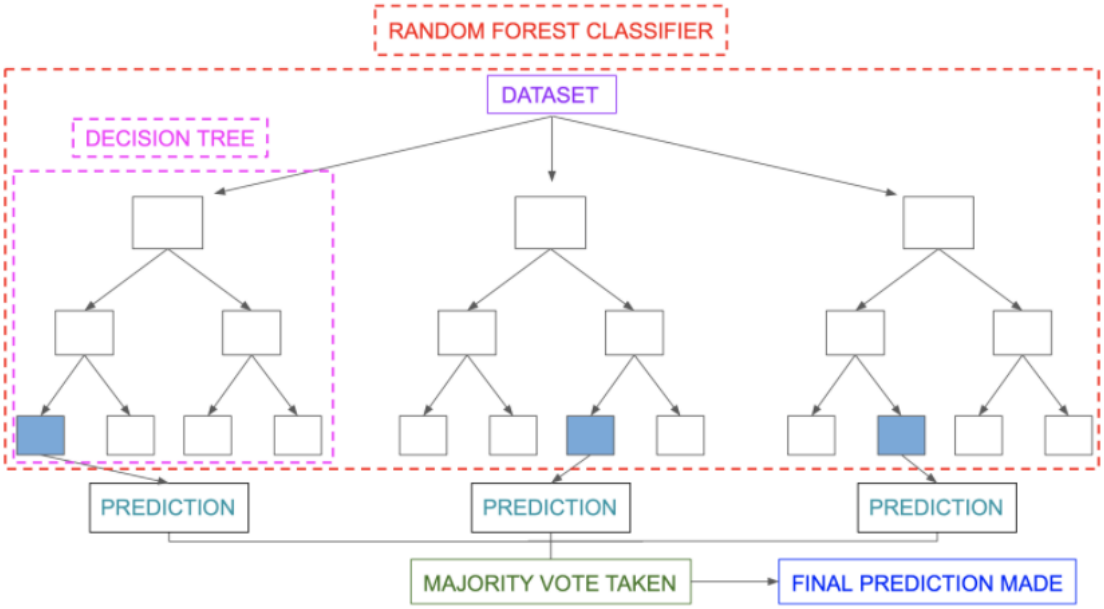


Figure 12.1: Random Forest model containing decision trees. Image source Kaggle

The reason why a random forest works really well because, all the trees inside the random forest is not too correlated with each other. If there was correlation, then error rate would increase, and it will decrease the performance of the model. This non correlation is achieved through the process of bagging.

12.1 Bagging (Bootstrapping + Aggregation)

Bagging is a procedure used to reduce the variance of a statistical learning method. In general, averaging a set of observations reduces variance. But since models usually have only one set of training data, this is not possible. Hence bootstrapping is used which is a technique of taking repeated samples from the same training data set. Bootstrapping is sampling with replacement. It is not the subset of training data that is used to bootstrap but rather the same size of data is fed into each tree, but each dataset would be randomly sampled with from the original data containing duplicate data points. Hence each tree is built on an independent bootstrap data set, and they do not become too correlated with each other. In a regression technique the average of all the predictions of trees is taken which is known as aggregation. But in a classification

scenario, the class predicted by each of the N trees is recorded and the overall prediction is the most commonly occurring class among N predictions.

ID	Checking Acct	Credit History	Employment	Credit Standing
1	No Acct	Paid	Long	Good
2	Yes Acct	Not Paid	Short	Bad
3	No Acct	Not Paid	Short	Good
4	Yes Acct	Not Paid	Short	Good
5	No Acct	Not Paid	Long	Bad

ID	Checking Acct	Credit History	Employment	Credit Standing
1	No Acct	Paid	Long	Good
2	Yes Acct	Not Paid	Short	Bad
3	No Acct	Not Paid	Short	Good
2	Yes Acct	Not Paid	Short	Bad
3	No Acct	Not Paid	Short	Good

ID	Checking Acct	Credit History	Employment	Credit Standing
3	No Acct	Not Paid	Short	Good
4	Yes Acct	Not Paid	Short	Good
5	No Acct	Not Paid	Long	Bad
4	Yes Acct	Not Paid	Short	Good
1	No Acct	Paid	Long	Good

ID	Checking Acct	Credit History	Employment	Credit Standing
1	No Acct	Paid	Long	Good
3	No Acct	Not Paid	Short	Good
4	Yes Acct	Not Paid	Short	Good
5	No Acct	Not Paid	Long	Bad
5	No Acct	Not Paid	Long	Bad

Figure 12.2: Bootstrapping of a sample data set. Left hand side shows the original data. Right hand side shows 3 bootstrapped samples obtained from original data

Random forest being a bagging and ensemble technique has lots of advantages. It is much more robust than decision trees and does not get affected by noise. It reduces overfitting problem of decision trees and improve accuracy. Random forest can be used for both classification and regression problems. Missing values are handled automatically by random forest by determining which values are similar to the missing data and using proximity matrix. It is robust to outliers and very stable. Disadvantages of random forest is that it is quite complex as it uses a large number of trees, and it takes longer time to train as compared to decision trees.

12.2 Implementation

To implement random forest model, first the randomForest() was imported and the data set after EDA was split into train and test data sets. Although missing could be imputed here using rfImpute(), the EDA completed dataset is given as input to the randomForest () without any missing values. All the missing values were omitted earlier in EDA. 75% of data is taken as the training set and the rest 25% of the data became the test set for the model. A seed value was set while implementing sample() in R to get the same train and test data in each run. The sample() function randomly picks up data from the data frame according to the size given. Here it is 0.75 for 75% of train data. All the 576 records were given as the training data to the model.

Here, random forest is implemented using randomForest() in R. It implements Brieman's random forest algorithm. When a classification predictive model is implemented, the response variable should be a factor. The right-hand side of formula of the model can have both numeric and factor variables separated by + symbol. Data is fed into each tree by bootstrap technique. Each input row gets predicted at least a few times based on the number of trees specified and finally majority vote is taken to get the predictor class. The commonly used arguments are:

- Formula – The expression to implement decision tree in tree (). The left-hand side is the response variable, and the right-hand side is the predictor variables separated by '+'

symbol. To exclude a predictor variable from the model '-' symbol can be used. It is also a common practice to use '.' symbol to include all predictor variables from the data in the formula. Symbol '~' is used to separate predictor and response variables in the model.

- Data – The data frame to be used to implement the model and formula
- Subset- Subset of the data frame to be used in model. E.g., The training subset of the data
- ntree- The number of trees that should be grown. It should not be too small
- mtry- Number of variables randomly selected at each split of a tree in randomForest
- Importance- Capture importance of each predictor variables in the model
- do.trace – outputs the classification error for each class for each tree

credit.rf <- randomForest(Credit.Standing~., data=credit_train_tree, ntree = 1000, importance=TRUE, do.trace=TRUE)

A random forest model credit.rf was implemented with credit standing as response variable and all the other attributes as predictor variables. Training data was given as the input to the model. The number of trees to implement random forest was given as 1000. While running the model it shows classification errors in each class and OOB error rates for each tree. On printing the model, it showed following

```
randomForest(formula = Credit.Standing ~ ., data = credit_train_tree,      ntree = 1000,
importance = TRUE, do.trace = TRUE)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 3
```

Figure 12.3: Summary of random forest model

The default mtry value was found to be 3. The training OOB error rate was 25.35%. Some of the common terminologies in random forest are:

12.3 OOB (Out of Bag) Error

Each tree in a random forest model is using a different bootstrap sample for the original data. Due to this some samples are left out for the construction of each tree. This is known as out of bag samples. There would be some decision tree models that do not see a particular data point at all. These models are used to predict the class of this unseen data point and the majority class is taken as random forest output for that data point. This is compared with the original class label to see if its correct or not. The proportion of times that it was not equal to the true class label is averaged over all cases to get the OOB error rate. This is more like an unbiased estimate of testing data set error. Hence random forest does not need cross validation, or a separate test set to get test error estimates.

P (selecting a bootstrap sample out of n data points) =1/n

P (not selecting a bootstrap sample out of n data points) =1-1/n

P (not selecting a bootstrap sample out of n data points in n independent trials) $=(1-1/n)^n$

As n tends to infinity, probability of not selecting a bootstrap sample out of n data points is derived to be 0.36. That is for a sample of 100 data points, 36 data points will be out of bag points.

12.4 Variable Importance

Every tree in random forest has its own out of bag data. That is, the data that it is unseen. The accuracy of predictions for out of bag samples is calculated first. After that, the values in the attributes of out of bag samples are randomly shuffled. But at the same time all other variables are kept intact. After this, the decrease in prediction accuracy of shuffled data is calculated. The mean decrease in accuracy of all trees is taken. On shuffling, the importance is captured as how much removing that particular variable decreased the prediction accuracy. Variable importance can be calculated not only using mean decrease in accuracy but also using mean decrease in gini index. Here impurity criterion between nodes in a tree is used to capture variable importance

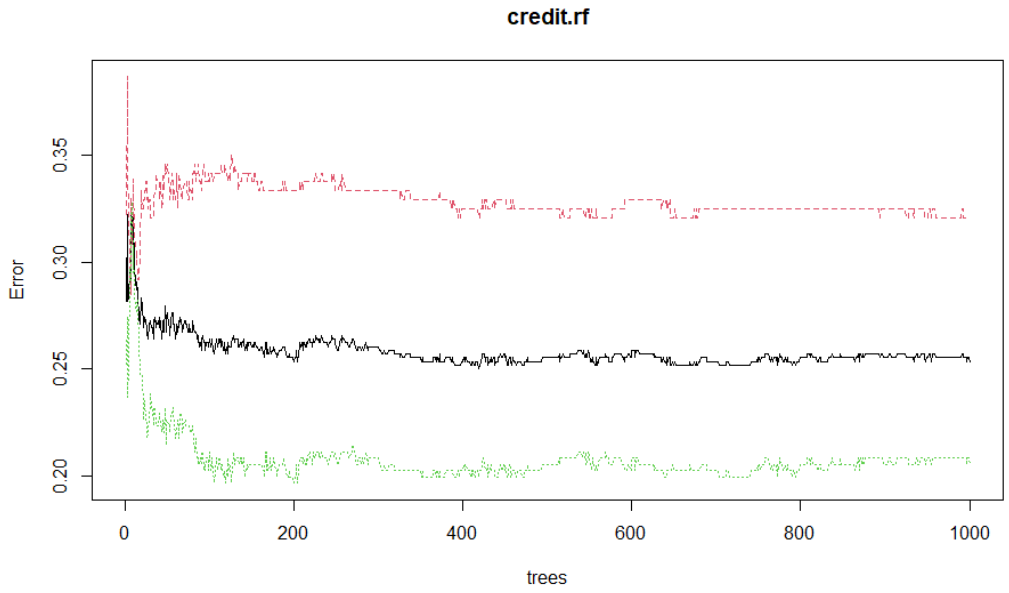


Figure 12.4: pattern of error with the increase in number of trees in the random forest model. Black- OOB Error, Red- error for Bad class, Green- error for Good class

Figure shows final OOB error rate of random forest model for 1000 trees is 25.17% after training. That is a misclassification of 25.17% on OOB samples. Also, the error rates for class Bad and Good were found to be more stabilized after 500 trees and the final error rates were 31.67% and 20.54%. Also, it can be seen that initial error rates were very high which would be the error for a single decision tree. But on implementing random forest as a collection of trees errors have minimized.

Variable Importance can be plotted using varImPlot() in R. It shows variable importance due to mean decrease in accuracy and mean decrease in Gini (impurity)

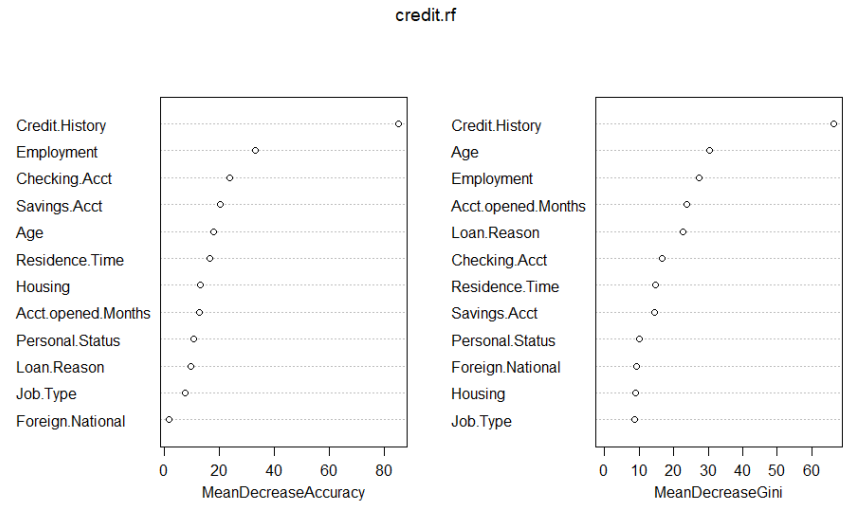


Figure 12.5: Variable importance plot in the random forest model. Left side using mean decrease in accuracy. Right side using mean decrease in Gini impurity

It can be seen that credit history, employment and checking acct are the most important variables according to mean decrease in accuracy which is similar to the one observed in tree(). When credit history was removed from OOB samples, accuracy decreased by 90% on an average. Rest of the other variables have less than 50% decrease in accuracy. Hence credit history is far more important in the model prediction than any other variables. But mean decrease in Gini shows Age has more importance after credit history. That is, these attributes contribute more to the homogeneity of leaf nodes. They can classify credit history more purely.

12.5 Evaluation

Prediction of results were done using predict(). Type was given as class to get predictions as classes and not probabilities

rf.pred = predict(credit.rf,credit_test_tree,type = "class")

Caret package was used to produce the confusion matrix in R. Confusion matrix is a table that is used to evaluate the performance of a classification model. On printing confusion matrix, we get the following result

	Reference / Actual	
Prediction	**Bad (Positive)**	**Good (Negative)**
Bad (Positive)	53 (TP)	17 (FP)
Good (Negative)	25 (FN)	98 (TN)

Table 12.1: confusion matrix of random forest model

Accuracy of the model is given by $\frac{TP+TN}{TP+TN+FP+FN} = \frac{53+98}{53+98+17+25} = 78.23\%$

Accuracy metric depicts how many were correctly predicted by the model out of the total predictions. Here accuracy of 78.23% is a fairly good accuracy. But still it is not a significant improvement from the decision tree model.

Recall/Sensitivity= $\frac{TP}{TP+FN} = \frac{53}{53+25} = 67.9\%$

Recall tells us how many actual positive classes were there in the entire testing data and out of that how many were predicted correctly. That is, how many Bad classes were predicted correctly out of all the Bad classes. This model gives a much better recall than decision tree. This because it was able to eliminate a lot of false negatives. Hence it won't predict much of the Bad credit standings as Good.

Precision=$\frac{TP}{TP+FP} = \frac{53}{53+17} = 75.7\ \%$

Precision tells how much good the model is in predictions. Precision talks only about predicted classes. Here precision has decreased by a few percentages as compared to decision tree model. This is due to the trade-off between precision and recall. As recall increases precision decreases.

12.6 Model Tuning

Model tuning is conducted by tweaking hyper parameters to improve the performance of the model. In random forest algorithm tuneRF() is used for this.

Mtry is the number of predictor variables that is used for splitting at each node in a tree. If mtry is 2, 2 attributes will be looked at for each split of the tree. In tueRF() a search for the optimum value of **mtry** with respect to the out of bag error estimate is conducted starting from the default value or the value specified in the parameter **mtryStart**. **StepFactor** determines the increase in mtry at each iteration. **ntreeTry** gives the number of trees that has to be used at each tuning step. **Improve** parameter gives the minimum improvement required in OOB error for the search for optimum mtry to continue.

The following tuning was done on the data with mtrystart of 2, stepFactor of 2 and ntreeTry of 1000

tuneRF(credit_train_tree[,-13],credit_train_tree[,13], mtryStart = 2,stepFactor = 2, ntreeTry = 1000, improve = 0.05)

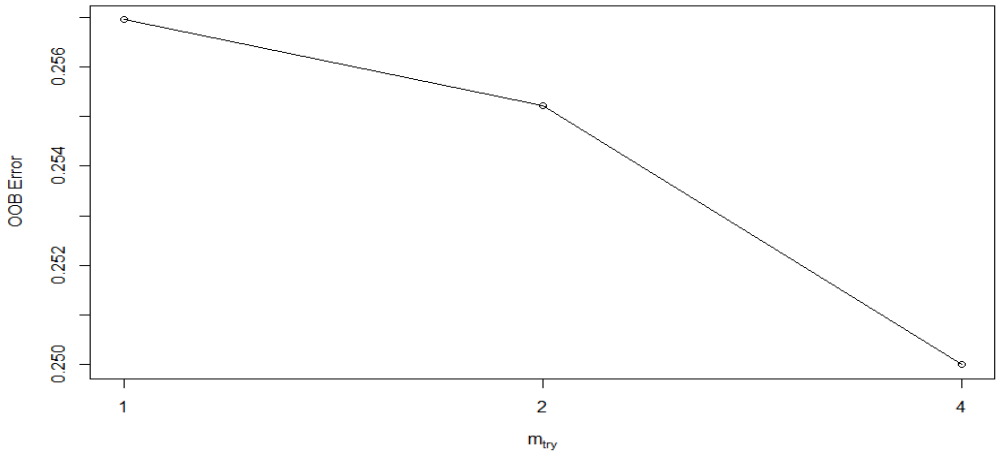


Figure 12.6: OOB error rate graph with mtry parameter

mtry	OOB Error
1	0.2569
2	0.2552
4	0.2500

Table 12.2: OOB error rate with each mtry

It can be seen that OOB error is minimum for mtry=4. That is 4 predictor variables have to be considered at each split of decision tree inside random forest.

Implementing random forest again but this time with mtry =4.

randomForest(Credit.Standing~.,data=credit_train_tree,mtry=4, importance=TRUE, do.trace=TRUE, ntree=1000)

The training accuracy was found to be the same before tuning 74.4%

Prediction on test data was conducted using predict()

predict(credit.rf,credit_test_tree,type = "class")

	Reference / Actual	
Prediction	**Bad (Positive)**	**Good (Negative)**
Bad (Positive)	54 (TP)	17 (FP)
Good (Negative)	24 (FN)	98 (TN)

Table 12.3: confusion matrix of random forest with best mtry

The accuracy after tuning was found out to be **78.75%** and it has improved slightly. Further improvement is possible by an exhaustive approach of tuning all hyper parameters and using boosting techniques.

The number of false negatives has reduced even further and has improved the recall to 69.2% which is again good for credit standing detection use case as stated before.

95% confidence interval for the accuracy was found out to be [72.3%,84.3%]. This tells that the accuracy would lie in this range for 95% of samples.

12.7 ROC-AUC Curve

Roc- Receiver operating characteristic and AUC- Area under the curve.

ROC curve shows true positive (y axis) vs false positive (x axis) rate for the model. If AUC is more the model is better.

To implement this, response variable was again predicted on tuned random forest model but this time using probability.

rf.pred2 <- predict(credit.rf,credit_test_tree, type = "prob")

Library pRoc was used to plot the characteristics by using roc() and auc()

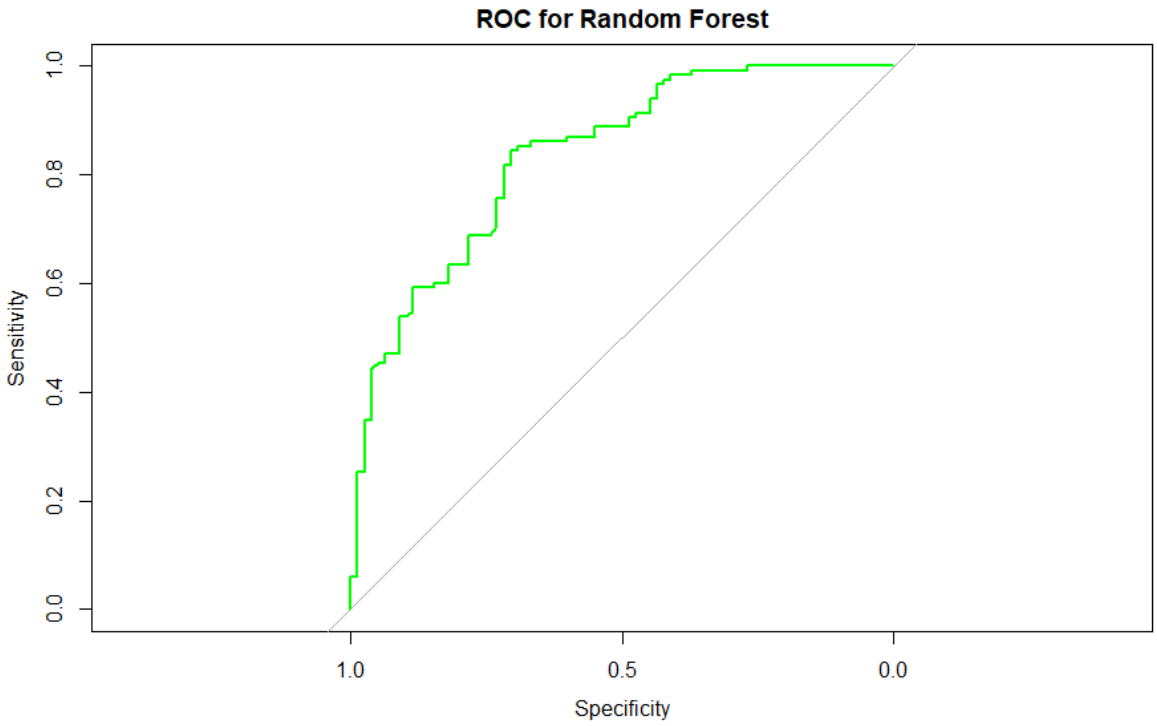


Figure 12.7: ROC-AUC curve of random forest model

The area under the curve was found out to be 83.3% which is good. If the area was 100% then the green line in the above figure would be a straight line on both axes.

12.8 Prediction On Scoring Data

Using the best model available it is possible to predict classes for a new set of unlabelled data. In the use case of credit standing, it is possible to give credit standing for new potential customers based on the model that was trained on existing data. However, the accuracy of the best model so far is 78.75% and the new predictions will also be based on this accuracy, and it may not be 100% correct.

To implement this, the new set of scoring data has to be loaded and data pre-processing steps has to be completed. Then, using the existing tuned random forest model (**credit.rf)** predictions are computed.

Credit.Standing. Score = predict(credit.rf,credit.risk.scoring,type = "class")

This newly predicted credit standing classes is appended to the scoring data set. This will give the financial institution the credit standing classes for new customers.

From the result it is observed that the model is giving accurate results, as customers with critical credit history has been classified as ‘Bad’ Credit standing.

13.GDPR

General data protection regulation (GDPR) is a new data protection law in European Union and EEA for data protection and to provide privacy for individuals. According to this law it limits the storage of personal data by organizations and regulates the transfer of personal data outside European Union and EEA. GDPR was introduced in 2016 and came into effect from 2018 onwards. This law gives individuals a control over their personal data and businesses cannot misuse them. GDPR principles are- Data processing must be according to law, and it should be transparent. Data must be processed only for legitimate purposes. Only required minimum data must be collected and all the personal data must be stored accurately and securely only for a limited time span. Proper security, integrity and confidentiality must be followed while processing data and the person or organization who handles this data must be accountable and responsible.

Often due to data privacy, personal and sensitive information is hidden or masked by the organization. The model evaluates Credit Standing of customers with GDPR in effect. That is, sensitive information such as Credit History, Personal Status and Foreign National status information is not available to the model for predictions.

13.1 Decision Tree Classification Model

To implement this, the above-mentioned attributes were removed from the data, and it was fed into the decision tree classification model. It was first trained with this data.

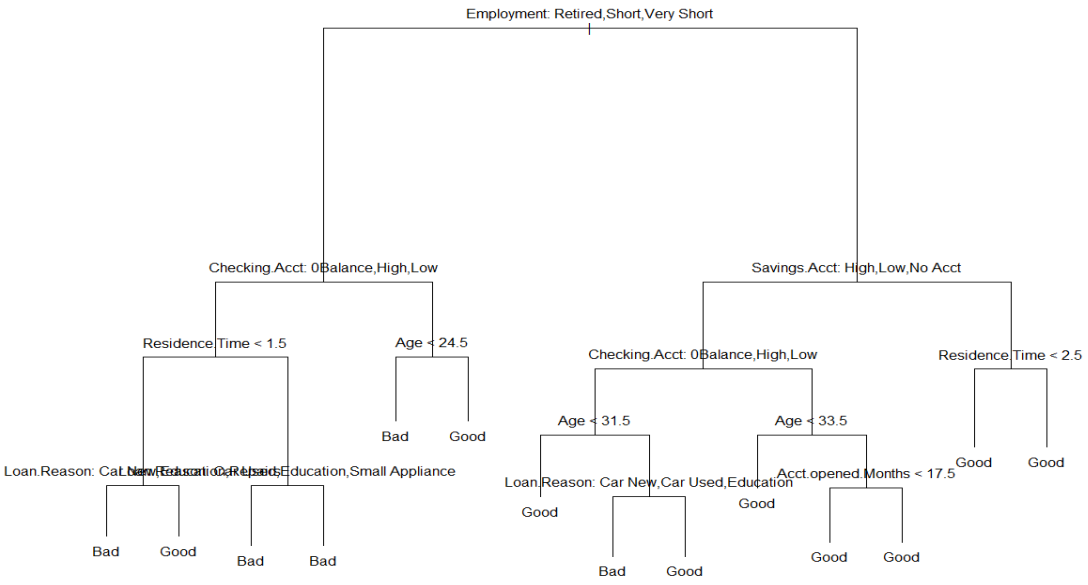


Figure 13.1: Decision tree after GDPR with 14 terminal nodes

In the absence of credit history, employment became the root node, as it was the second most important feature. The tree has changed vastly, and it has 14 terminal nodes now instead of 10 without GDPR. The child nodes have become Checking Acct and Savings Acct followed by residence time and age. This proves that decision tree has high instability, and it changes rapidly with the training data.

The predictions for test data give an accuracy of 64.7%. The accuracy has decreased considerably. This is mainly due to the loss of important predictor variables such as credit history. At the same time, it also shows how decision tree algorithm is prone to overfitting of training data.

As the complexity of the tree is very high pruning and cross validation has to be conducted to get a less complex tree or the one with better accuracy

cv.credit.gdpr <- cv.tree(tree.credit.gdpr,K=10, FUN=prune.misclass)

Cross validation of 10 folds is conducted with cost complexity pruning based on misclassification.

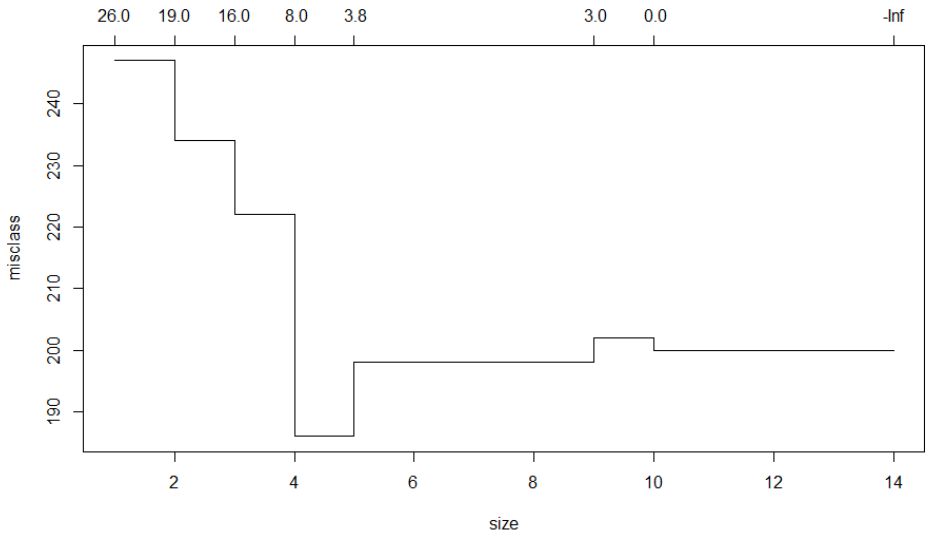


Figure 13.2: tree size vs misclassification

A tree size of 4 or 5 terminal nodes has the least misclassification

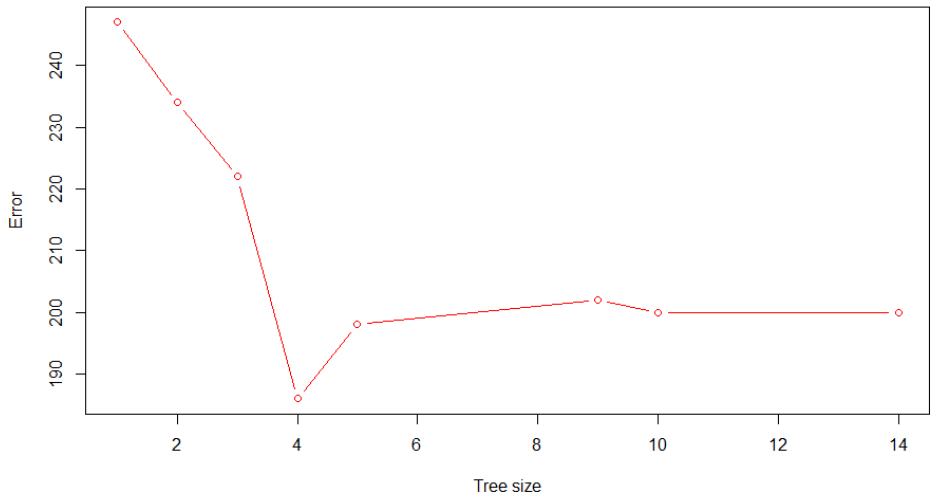


Figure 13.3: Decision tree size (number of terminal nodes) vs Deviance

prune.credit.gdpr<-prune.misclass(tree.credit.gdpr,best=5)

On creating a pruned decision tree with 5 terminal nodes, the complexity of tree has highly reduced from 14 to 5 terminal nodes.

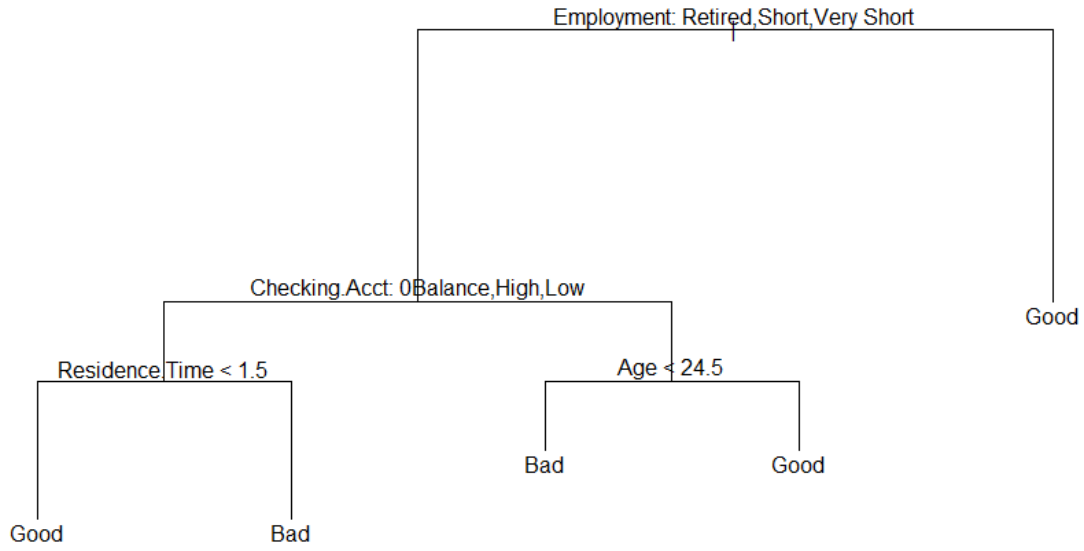


Figure 13.4: Pruned Decision tree with 5 terminal nodes

The new pruned tree only contains Employment, Checking Acct, Residence Time and Age to decide classifications of Credit Standing. This tree gives an accuracy of 63.7%. The accuracy has decreased slightly after pruning, but the simplicity of tree has increased a lot.

For further improvements random forest model is used on this data set.

13.2 Random Forest Model

To implement random forest model, the sensitive personal information's are masked and is fed into the random forest model

randomForest(formula = Credit.Standing ~ ., data = credit_train_tree_gdpr, ntree = 1000, importance = TRUE, do.trace = TRUE)

The GDPR implemented data set was given in the argument 'data' for training. All the available attributes were used for modelling the response variable credit standing. The number of trees used in random forest were 1000. The model was asked to track the important variables. Also, it was asked to trace the out of bag error prediction classes.

The number of variables tried at each split was still 3. But OOB error after training has increased to 29.17% when compared to the model without GDPR implementation.

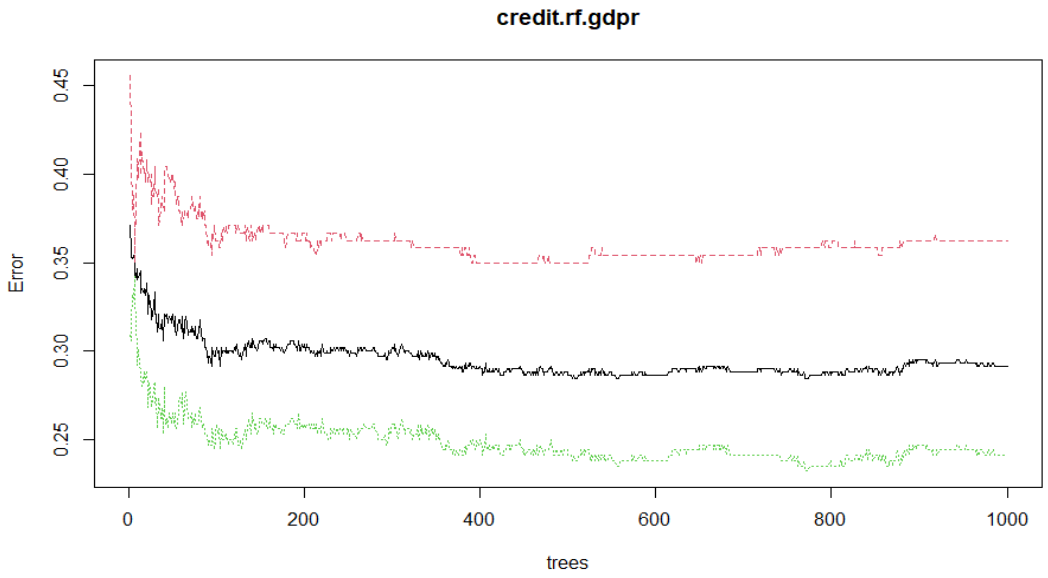


Figure 13.5: Pattern of error with the increase in number of trees in the random forest model. Black- OOB Error, Red- error for Bad class, Green- error for Good class

On plotting the model, it is observable that final OOB error is 29.17%. That is a misclassification of 29.17% on OOB samples. An increase of 4% when compared to the implementation of same model without GDPR. Also, the error rates for class Bad and Good were found to be more stabilized after 500 trees and the final error rates were 36.25% and 24.11%. These errors have also increased by 4%. It can be seen that initial error rates were very high which would be the error for a single decision tree. But on implementing random forest as a collection of trees errors have minimized.

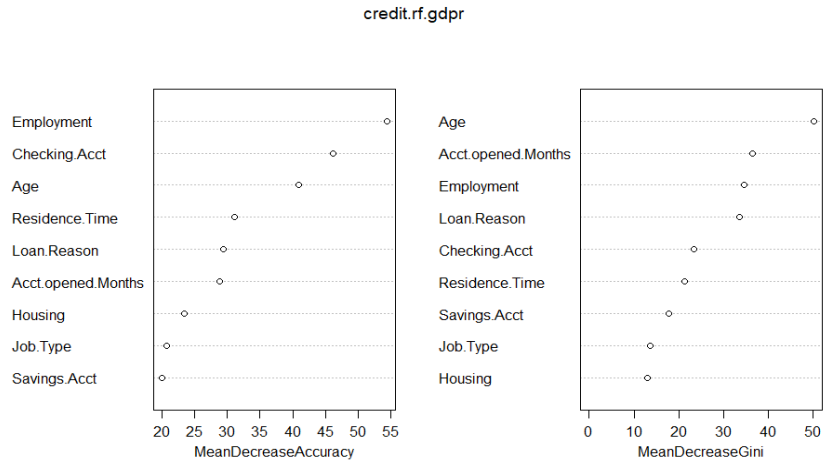


Figure 13.6: Variable importance plot in the random forest model. Left side using mean decrease in accuracy. Right side using mean decrease in Gini impurity

It can be seen that employment, checking acct, age and residence time are the most important variables according to mean decrease in accuracy which is similar to the one observed in tree(). When employment was removed from OOB samples, accuracy decreased by 55% on an average. But mean decrease in Gini shows Age and Acct opened Months have more importance than most of the other variables. That is, these attributes contribute more to the homogeneity of leaf nodes.

Prediction of results were done using predict().

predict(credit.rf.gdpr,credit_test_tree_gdpr, type = "class")

The accuracy on testing the model is **75.13%.** This is a major improvement as compared to the tree's accuracy of 64.7%

Tuning the model

TuneRF() is again used here for tuning the model.

The following tuning was done on the data with mtrystart of 2, stepFactor of 2 and ntreeTry of 500

tuneRF(credit_train_tree_gdpr[,-10],credit_train_tree_gdpr[,10], subset=credit_train_tree_gdpr, mtryStart = 2,stepFactor = 2, ntreeTry = 500)

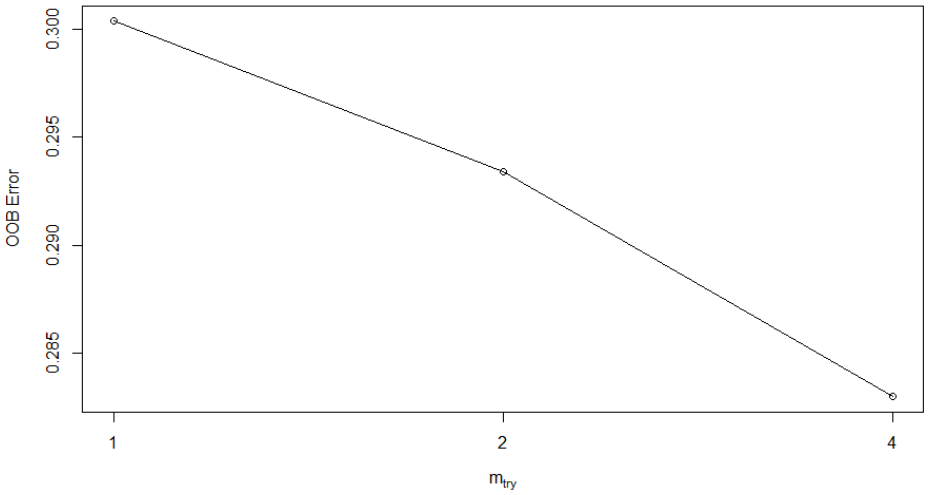


Figure 13.7: OOB error rate graph with mtry parameter

mtry	OOB Error
1	0.3003
2	0.2934
4	0.2829

Table 13.1: OOB error rate with each mtry

It can be seen that OOB error is minimum for mtry=4. That is 4 predictor variables have to be considered at each split of decision tree inside random forest.

Implementing random forest again but this time with mtry =4

randomForest(Credit.Standing~.,data=credit_train_tree_gdpr,mtry=4, importance=TRUE, do.trace=TRUE)

The accuracy after tuning was found out to be 75.1%. The accuracy has remained the same after tuning the model. Further improvement is possible by an exhaustive approach of tuning all hyper parameters and using boosting techniques.

Results

Overall, it can be concluded that the accuracy of models has decreased after implementing GDPR, especially for decision tree algorithm. This shows that decision tree is highly dependent on training data. Random forest is more robust, and its accuracy has decreased very less as compared to a tree. The decrease in accuracy is due to the exclusion of important variables like credit history in model prediction of credit standing.

14.Detecting Suspicious Pattern

Either due to the inaccuracy of computer system or due to the negligence or lack of knowledge of humans it is possible to have wrong classes given in the response variable. Such a situation causes lots of issues in terms of data quality and model predictability. Since a model train on this incorrect data, the prediction accuracy on test data would be completely thrown off leading to business losses. Also, it will be hard for the model to unlearn what it has learned, unless the cleaning of the data set is done, and the model is trained again which is very costly. Hence such inaccuracies have to be avoided at all costs. At the same time, it is important to develop a strategy so that such suspicious patterns can be found out.

The strategy followed is:

1) Use the decision tree model that got trained on 75% of the data.
2) Predict using test data. But this time the test data will contain the entire data set so that the response labels from the model is available for all the data points. The model has an accuracy of 78.5%.
3) Predicted response variables are made into a data frame.
4) A new empty vector is created to capture nonmatching ID values of data points.
5) The ID column, entire data set containing actual response variable and prediction values created as a data frame in step3 are combined into one new data frame
6) Iterative go through each data point in this new data frame and compare actual response variable and predicted response variable.
7) If the comparison in step 6 is not matching, append the ID values of such records into the vector created in step 4.
8) Capture all the attributes of nonmatching ID values and observe for any suspicious pattern

On close observation of the pattern of unmatched data points, it was seen that data points from 314-326 (ID values) is nonmatching. That is, the original response variable (credit standing) was BAD, but the model predicted GOOD credit standing.

On further investigating this suspicious pattern it is seen that

- All the records in this pattern have credit history as current. That is, they have paid back all dues till now but still the original response variable credit standing was ‘Bad’.
- Most of them have loan reason as car used, car new, furniture and small appliance. Generally, customers with these loan reasons have good credit standing but the response variable given was as bad.

- Most of the customers in this range have their own housing and they have been given a bad credit standing. It was seen in the earlier analysis that people with their own housing mostly have good credit standing.
- A lot of the customers in this suspicious pattern also have medium and long employment. Such customers usually have good credit standing, but here it is recorded as bad.

All these observations lead to the conclusion that these suspicious consecutive data points were incorrectly labelled at a particular time. Hence these needs to be changed, further investigated on a case-by-case basis and the model has to be retrained again after the correction. Several other non-matching values are seen in 300-350 ID values as well.

15. CONCLUSION

This report went in detail to capture the process of exploratory data analysis, building, predicting and evaluating classification models to predict credit standing classes of customers in a financial institution. Firstly, summary statistics of numerical variables were done. Later, univariate, bi-variate and tri-variate analysis of attributes were conducted. This analysis showed certain attributes like credit history and employment can classify target labels more purely. Then splitting of data into train and test was done.

After the train test split, entropy and information gain calculation, which is the base for decision tree classification was conducted on training data. The same process was repeated for categorical variables with and without binary split and after including binned numerical variables. All the process gave credit history variable as the one which gives maximum information gain. Hence credit history was selected as the root node for the decision tree. The data was split into two based on credit history to find next level split. It was found out that employment and Checking acct have the least entropy and they can be selected as child nodes for second level splits. This entropy calculated tree could be further improved. Binning of numerical variables into binary was done by checking the optimum value that gives least entropy. But binning of categorical variable into binary was not done by considering all the possible combinations of each attribute. This could be further improved and optimized by better algorithms.

After the manual implementation of entropy based decision tree, tree() was used to generate the decision tree. The tree() also showed similar behaviour as entropy method with credit history and employment as initial level splits. The tree had 10 terminal nodes and produced an accuracy of 77.7%. Pruning and cross validation was conducted on this. After pruning the complexity of the tree got reduced to 4 terminal nodes but with the same accuracy.

Next, Random forest model was implemented using the package randomForest(). The model gave a better accuracy than decision tree with 78.27%. An improvement in recall was also seen. Tuning of hyper parameters was conducted and the best mtry was obtained as 4 with least OOB error. After tuning the accuracy of the model improved to 78.75%. Further improvements to this model are possible by boosting techniques and better hyper parameter tuning.

On implementing GDPR, some of the important attributes had to be removed. This plummeted the accuracy of decision tree model to 64.7% but the accuracy of random forest model only reduced to 75.13%. This shows that random forest is a more robust model and is not prone to overfitting to training data easily.

Finally, a strategy was developed by using decision tree model to find out a pattern of incorrectly entered labels in the data set.

16. REFERENCES

- An Introduction to Statistical learning with applications in R by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani.
- Manning R in Action second edition Data analysis and graphics with R by Robert I. Kabacoff.
- https://www.rdocumentation.org/packages/tree/versions/1.0-41/topics/prune.tree
- https://www.educative.io/edpresso/what-is-decision-tree-pruning-and-how-is-it-done
- https://cran.r-project.org/web/packages/tree/tree.pdf
- https://medium.com/analytics-vidhya/binary-decision-trees-351caef46ec4
- https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6#:~:text=The%20Root%20Node%3A%20Is%20the,nodes%20where%20predictions%20are%20made.
- https://www.saedsayad.com/decision_tree.htm
- http://www.sthda.com/english/articles/32-r-graphics-essentials/129-visualizing-multivariate-categorical-data/
- https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6
- https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/
- https://towardsdatascience.com/understanding-random-forest-58381e0602d2
- https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- https://www.displayr.com/how-is-variable-importance-calculated-for-a-random-forest/
- https://www.analyticsvidhya.com/blog/2020/12/out-of-bag-oob-score-in-the-random-forest-algorithm/
- https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest
- https://www.rdocumentation.org/packages/iRF/versions/2.0.0/topics/tuneRF
- https://www.kaggle.com/milesh1/receiver-operating-characteristic-roc-curve-in-r
- https://r-coder.com/cut-r/
- https://www.rdocumentation.org/packages/ggplot2/versions/3.3.5
- https://gsp.humboldt.edu/olm/R/05_04_CART.html
- https://www.kaggle.com/search?q=decision+tree+tag%3A%22decision+tree%22
- https://www.kaggle.com/prashant111/random-forest-classifier-tutorial

17. APPENDIX

Appendix A – Raw Data

	Checking.Acct	Credit.History	Loan.Reason	Savings.Acct	Employment	Personal.Status	Housing	Job.Type	Foreign.National	Acct.opened.Months	Residence.Time	Age	Credit.Standing
1	No Acct	All Paid	Car New	Low	Medium	Single	Own	Management	No	7	3	44	Good
2	0Balance	Current	Car New	Low	Short	Divorced	Own	Skilled	No	16	2	28	Bad
3	0Balance	Current	Car New	No Acct	Long	Divorced	Own	Skilled	No	25	2	28	Bad
4	No Acct	All Paid	Small Appliance	No Acct	Long	Single	Other	Skilled	Yes	7	4	35	Good

Appendix B - Summary of Data

```
 Checking.Acct   Credit.History         Loan.Reason   Savings.Acct      Employment  Personal.Status  Housing          Job.Type
0Balance:260   All Paid :134    Car New        :190   High    : 28   long      :  4   Divorced:270    Other: 95   Management:101
High    : 43   Bank Paid: 61    Small Appliance:181   Low     :497   Long      :187   Married : 70    Own  :520   Skilled   :492
Low     :199   Critical : 94    Furniture      :156   MedHigh: 48   Medium    :146   Single  :429    Rent :154   Unemployed: 18
No Acct :267   Current  :416    Business       : 82   MedLow : 79   Retired   :  2                               Unskilled :158
               Delay    : 64    Car Used       : 73   No Acct:117   Short     :250
                                Education      : 45                 Unemployed: 43
                                (Other)        : 42                 Very Short:137
Foreign.National Acct.opened.Months Residence.Time       Age       Credit.Standing
No :245          Min.   :  5.0      Min.   : 1.00   Min.   :18.0   Bad :318
Yes:524          1st Qu.: 13.0      1st Qu.: 2.00   1st Qu.:26.0   Good:451
                 Median : 19.0      Median : 3.00   Median :32.0
                 Mean   : 23.2      Mean   : 2.87   Mean   :34.7
                 3rd Qu.: 29.0      3rd Qu.: 4.00   3rd Qu.:41.0
                 Max.   :120.0      Max.   :10.00   Max.   :85.0
```

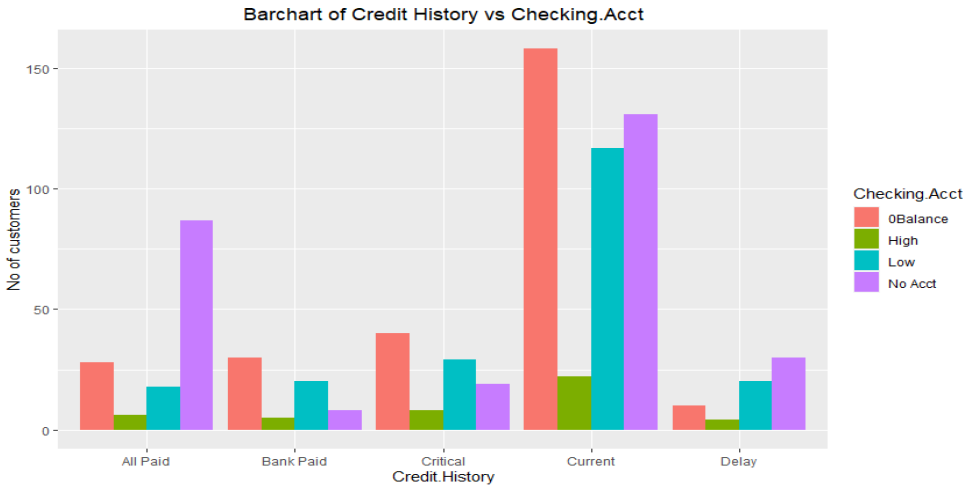
```
   skim_variable    n_missing complete_rate ordered n_unique top_counts
 * <chr>                <int>         <dbl> <lgl>      <int> <chr>
 1 Checking.Acct            0             1 FALSE          4 No : 267, 0Ba: 260, Low: 199, Hig: 43
 2 Credit.History           0             1 FALSE          5 Cur: 416, All: 134, Cri: 94, Del: 64
 3 Loan.Reason              0             1 FALSE         10 Car: 190, Sma: 181, Fur: 156, Bus: 82
 4 Savings.Acct             0             1 FALSE          5 Low: 497, No : 117, Med: 79, Med: 48
 5 Employment               0             1 FALSE          7 Sho: 250, Lon: 187, Med: 146, Ver: 137
 6 Personal.Status          0             1 FALSE          3 Sin: 429, Div: 270, Mar: 70
 7 Housing                  0             1 FALSE          3 Own: 520, Ren: 154, Oth: 95
 8 Job.Type                 0             1 FALSE          4 Ski: 492, Uns: 158, Man: 101, Une: 18
 9 Foreign.National         0             1 FALSE          2 Yes: 524, No: 245
10 Credit.Standing          0             1 FALSE          2 Goo: 451, Bad: 318

-- Variable type: numeric -----------------------------------------------------------------------------------------
# A tibble: 3 x 11
  skim_variable      n_missing complete_rate  mean    sd    p0   p25   p50   p75  p100 hist
* <chr>                  <int>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
1 Acct.opened.Months         0             1 23.2  12.7      5    13    19    29    120 ▇▃▂▁▁
2 Residence.Time             0             1  2.87  1.15     1     2     3     4     10 ▇▇▂▁▁
3 Age                        0             1 34.7  11.4     18    26    32    41    85 ▇▇▃▂▁
```

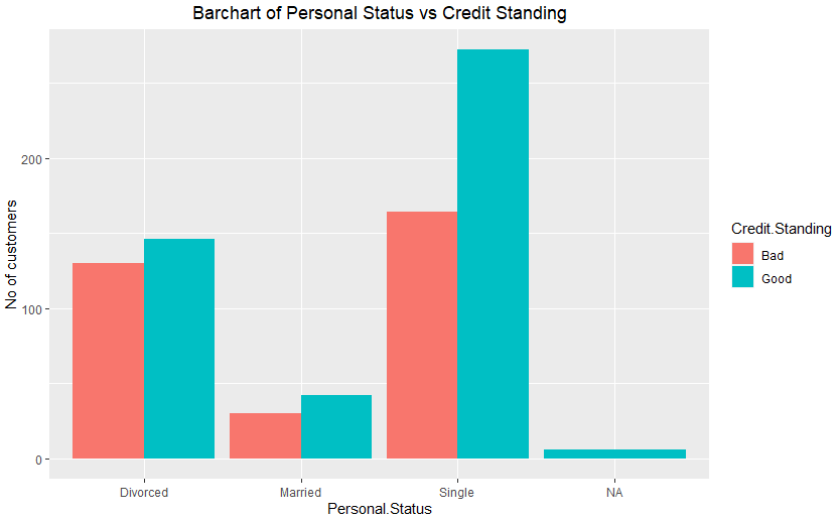
Appendix C – Chi Square Test

Conducting **CHI- SQUARE** test to establish the significance of relationship:
Null Hypothesis: variables credit history and credit standing have no relation.
Alternative Hypothesis: Variables credit history and credit standing have relation between them
Significance level alpha = 0.05. Using *chisq.test()* in R, the p-value was found to be << 0.05 and chi squared values was 231.99 which is much higher than the cut-off for the null hypothesis acceptance region. Hence, we reject null hypothesis. Hence, variables credit history and credit standing have significant relationship between them is proved.

Appendix D – Bi Variate Analysis



Most of the 0 balance checking accounts have current category in credit history. All paid credit history is mostly for customers having No Acct in checking Acct.



Divorced customers are equal in both good and bad credit standing classes. Very large number of single customers have good credit standing. NA values area also observed.

From the bivariate analysis, it is conclusive that **Credit History and Employment** can classify the target variable credit standing more clearly than other variables. These variables will have more influence on the decision made by the model