DATA8001 Assignment

Credit_Risk_25_final-2.csv

Brian Higgins – R00239570

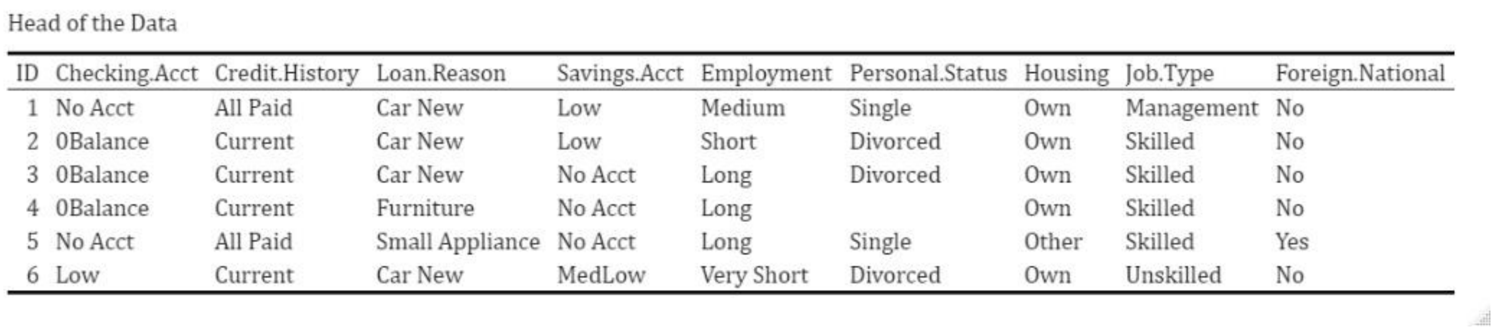
1. Introduction

1.1 Data Introduction.

The data is made up of Financial data with 809 entries over 14 columns. There are four Integer/numeric columns and ten character columns, the character columns are categorical types with a number of entries which are listed below. A description is included if it is needed.

- ID – Integer – ID for the row observation, which are customers.
- Checking Account – Character – What level of checking account does the customer have.
  - No Acct, 0Balance, Low, High
- Checking History – Character – Credit history of the customer.
  - All Paid, Current, Bank Paid, Delay, Critical
- Loan Reason – Character - Purpose of the loan
  - Car New, Furniture, Small Appliance, Education, Car Used, Business, Large Appliance, Repairs, Other, Retraining
- Saving Account – Character
  - Low, No Acct, MedLow, MedHigh, High
- Employment – Character – Employment status
  - Medium, Short, Long, Very Short, Unemployed, Retired
- Personal Status – Character
  - Single, Divorced, Married
- Housing – Character - Housing situation
  - Own, Other, Rent
- Foreign National – Integer –
  - Yes, No
- Months Since Checking Account opened – Integer– Value for length of time in months
- Residence Time in Current District – Integer – Value for the length of time in current District.
- Age – Numeric - Value in years
- Credit Standing – Character - Credit Rating

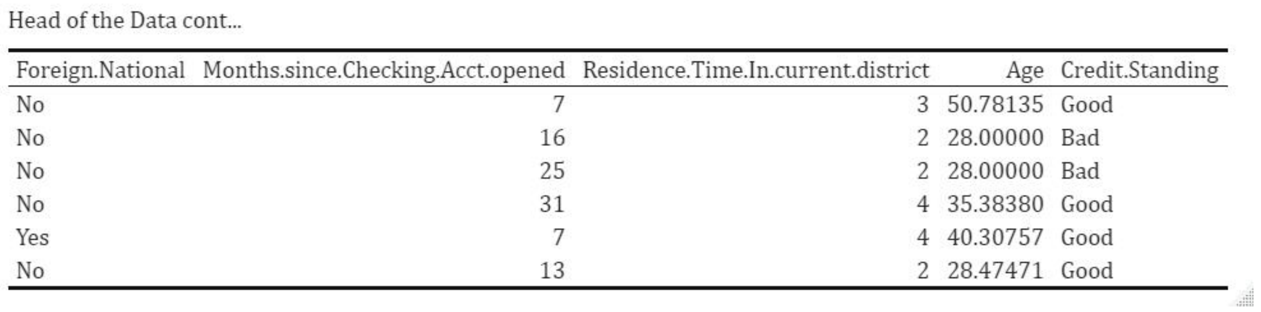
Below in Fig 1.1 and Fig 1.2 you can see an output of the top of the table, you can see the ID, then the various categorical columns, with the numeric columns and followed lastly by the target column.



Head of the Data

ID	Checking.Acct	Credit.History	Loan.Reason	Savings.Acct	Employment	Personal.Status	Housing	Job.Type	Foreign.National
1	No Acct	All Paid	Car New	Low	Medium	Single	Own	Management	No
2	0Balance	Current	Car New	Low	Short	Divorced	Own	Skilled	No
3	0Balance	Current	Car New	No Acct	Long	Divorced	Own	Skilled	No
4	0Balance	Current	Furniture	No Acct	Long		Own	Skilled	No
5	No Acct	All Paid	Small Appliance	No Acct	Long	Single	Other	Skilled	Yes
6	Low	Current	Car New	MedLow	Very Short	Divorced	Own	Unskilled	No

Fig 1.1 Head of the data



Head of the Data cont...

Foreign.National	Months.since.Checking.Acct.opened	Residence.Time.In.current.district	Age	Credit.Standing
No	7	3	50.78135	Good
No	16	2	28.00000	Bad
No	25	2	28.00000	Bad
No	31	4	35.38380	Good
Yes	7	4	40.30757	Good
No	13	2	28.47471	Good

Fig 1.2 Head of the data cont

1.2 Report Outline

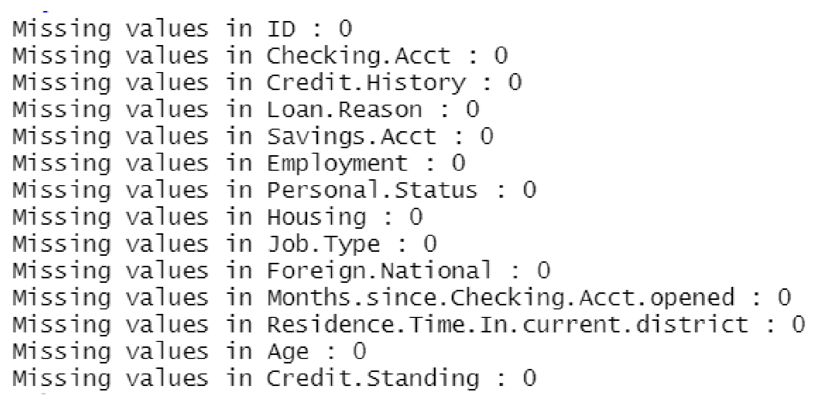
This project aims to take the target column, which we have been told is the Credit Standing column. This column has two values, “Good” and “Bad” that tell us the Credit Standing or Credit Rating of customers who have accounts. We will then use various models to look to see how accurately we can predict whether a Customer will have a Good or Bad rating. We will use Decision Trees and Random forests along with various Hyperparameters to fine-tune our models.

2. Exploratory Data Analysis

2.1 Missing / Duplicate Data

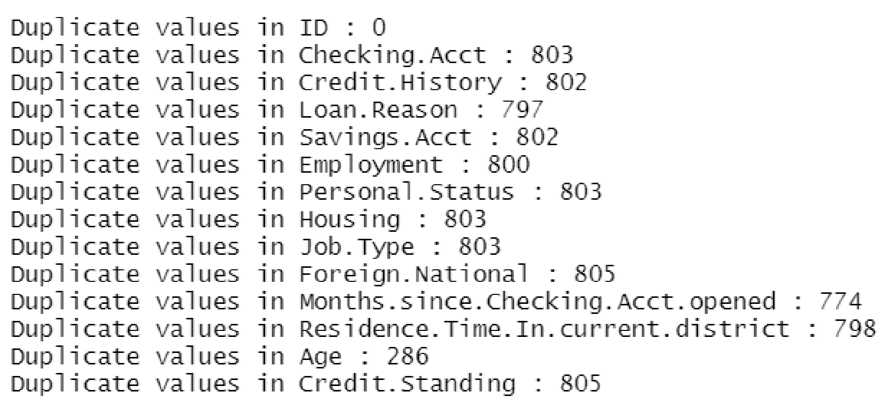
As you can see below in Fig 2.1.1 at first glance there appears to be no Missing Data within the Columns. However, on investigating the head of the data we see there are some columns have empty values.

In fig 2.1.2 there are a large number of duplicate values within the data, this is not surprising given that most columns are categories with classes. The ID has no duplicates which are important so we know no entry is duplicated.



```
Missing values in ID : 0
Missing values in Checking.Acct : 0
Missing values in Credit.History : 0
Missing values in Loan.Reason : 0
Missing values in Savings.Acct : 0
Missing values in Employment : 0
Missing values in Personal.Status : 0
Missing values in Housing : 0
Missing values in Job.Type : 0
Missing values in Foreign.National : 0
Missing values in Months.since.Checking.Acct.opened : 0
Missing values in Residence.Time.In.current.district : 0
Missing values in Age : 0
Missing values in Credit.Standing : 0
```

Fig 2.1.1 Missing Values



```
Duplicate values in ID : 0
Duplicate values in Checking.Acct : 803
Duplicate values in Credit.History : 802
Duplicate values in Loan.Reason : 797
Duplicate values in Savings.Acct : 802
Duplicate values in Employment : 800
Duplicate values in Personal.Status : 803
Duplicate values in Housing : 803
Duplicate values in Job.Type : 803
Duplicate values in Foreign.National : 805
Duplicate values in Months.since.Checking.Acct.opened : 774
Duplicate values in Residence.Time.In.current.district : 798
Duplicate values in Age : 286
Duplicate values in Credit.Standing : 805
```

Fig 2.1.2 Missing Values

2.2 First look at columns and anything interesting to note

In Fig 2.2.1 we have a look at four predictor columns that will likely have an effect on the Target Variable, Credit Standing.

- Some columns have “ “ instead of missing Na values. We will change this to “ No Information” to describe what it is.
- Credit History has a large number of Customers who have current loans. Also, 143 have “All Paid”, this should be useful for training the model.
- Savings account has 518 customers with “Low” savings and also 128 that have “No Account”. From a biased view, these will also have a negative effect on a Credit Rating.
- For Employment nearly one-third have “Long” and “Medium” employment which should be Good for a Credit Rating.
- Job type shows a high amount of “Management” and “Skilled” Customers which is also likely to have a Good effect on Credit Rating.

The above observations are biased from the business point of view but our model will not have these biases so it will be interesting to see if it finds these predictors important.

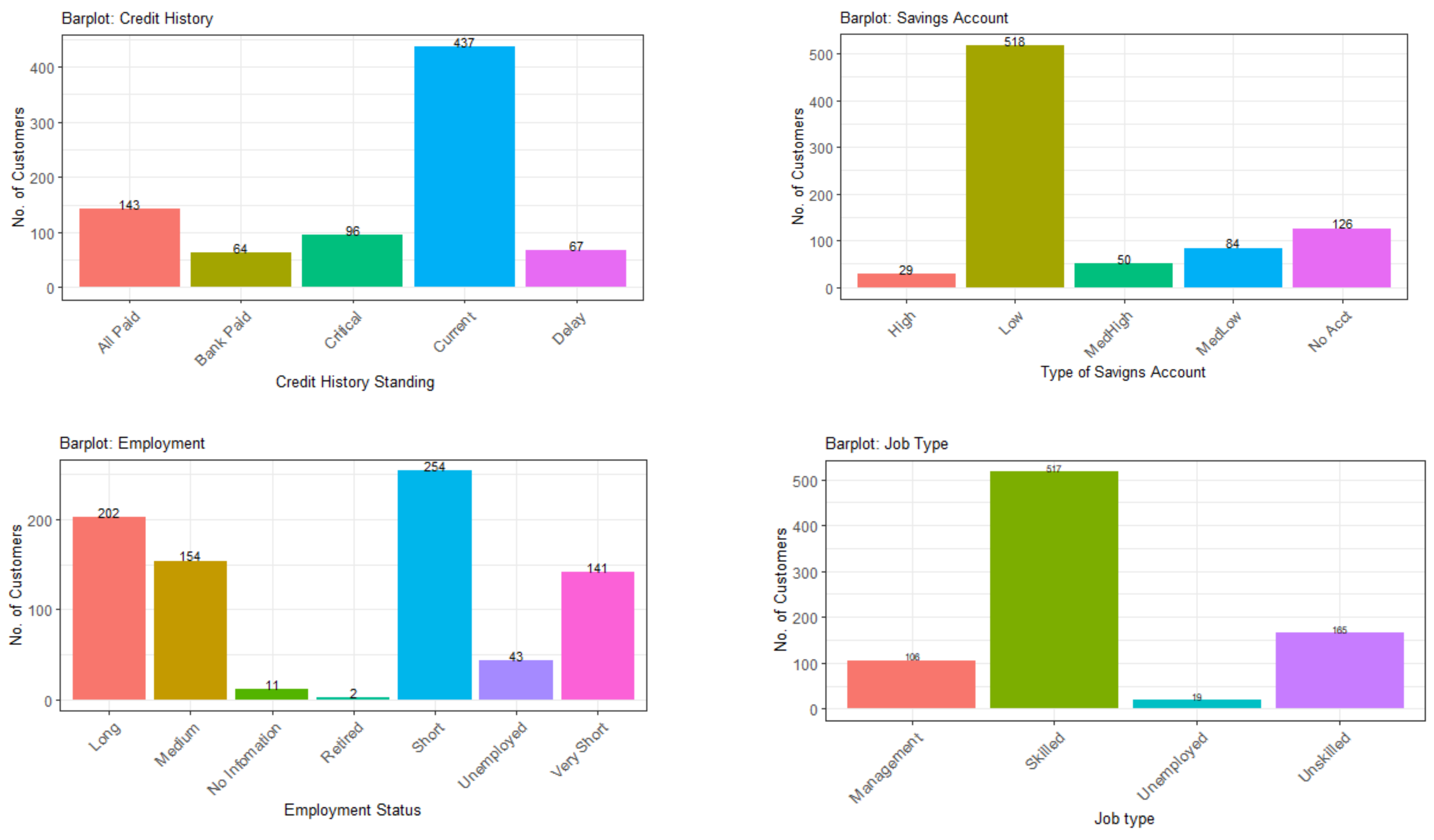


Fig 2.2.1 Bar plot of the likely important predictor columns

In Fig 2.2.2 we can see the numeric columns, all the data is right-skewed with some outliers giving the data tails.

- Months Since Checking Account Opened is in months but the other two numeric columns are in years. So we will change this to years. Otherwise, 1 month and 1 year would have the same weight. There is an outlier of 10 years but there are customers in the district for ten years so we will keep it. Just an outlier to be aware of.
- For residence time in the current distinct there is a minus 2 value. You cannot live in a place -2,so we will change this to 0.
- Age is in a float number type with 5 decimal points. While age can be a continuous variable we do not need it to be, but for our use, we will change it into a whole number. E.x. 43.34232 to 43

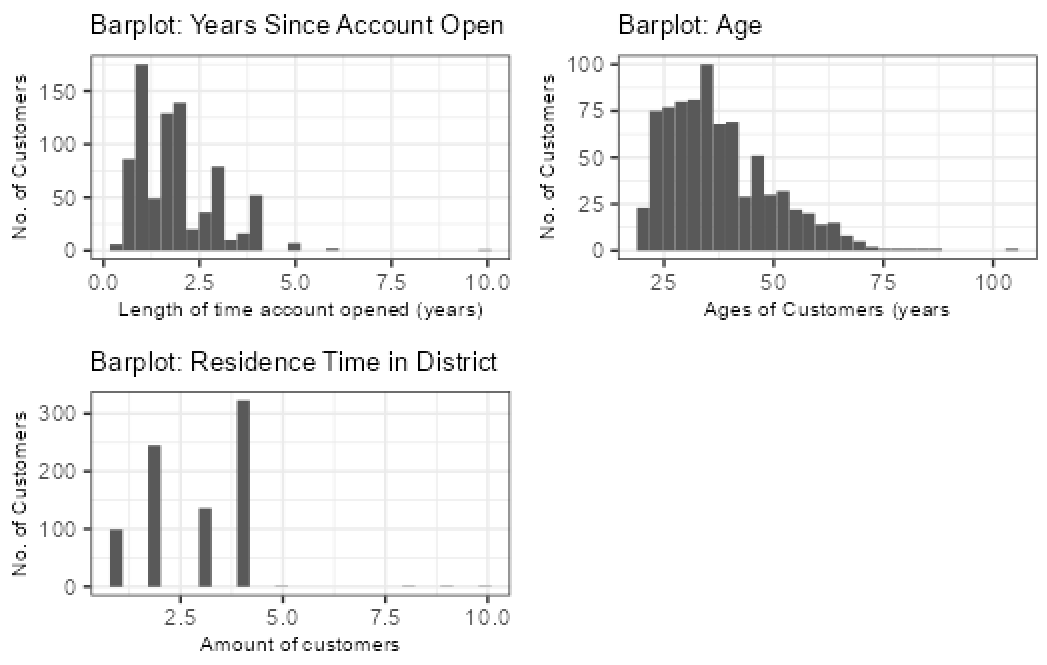


Fig 2.2.2 Barplot of contusions variables

2.3 Multivariate Analysis

Since this is an early analysis we do not know which predictors will be important for the model. Fig 2.3.1 shows a heat map of the Correlation between each of the Numeric columns. I wanted to look to see if there was a relationship between the n Years Since the account opened and Residence time in District especially. However, we can see that there is not much correlation between them. There is a weak positive correlation between Age and Residence which does make sense as you would imagine the older you are the longer you live in the same place.

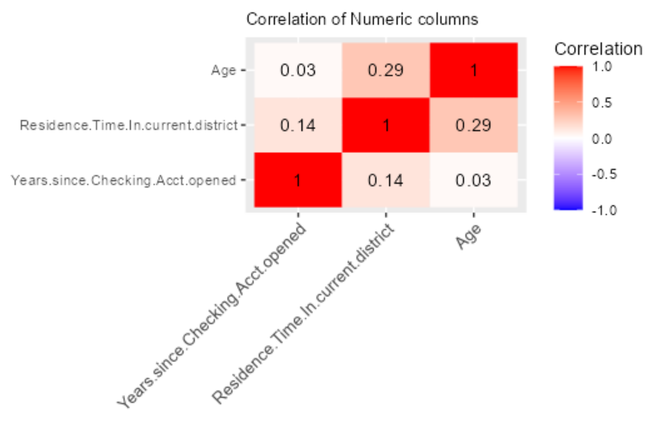
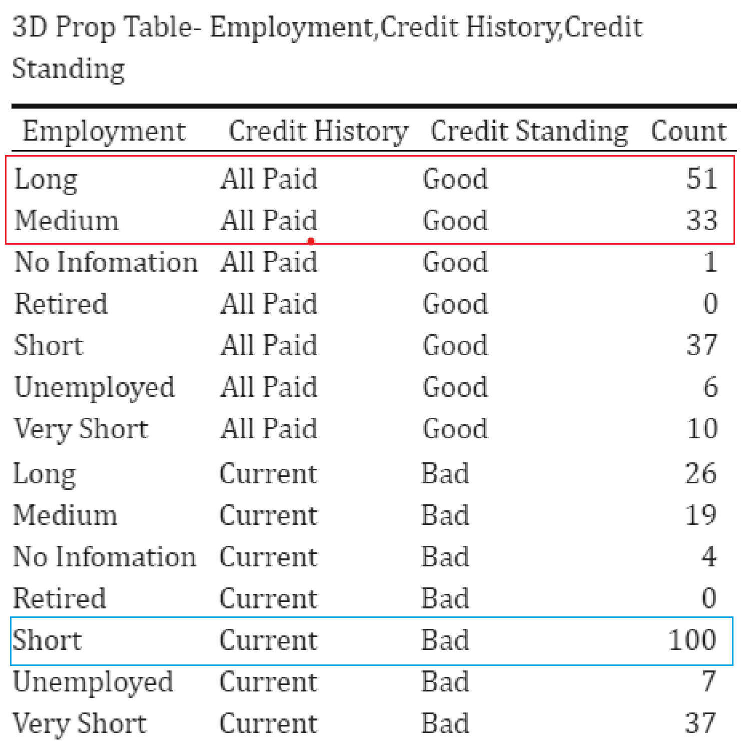


Fig 2.3.1 Multivariate Heatmap for numeric columns

Fig 2.3.2 shows a sample part of a 3D prop table of Employment, Credit History and Credit Standing with a count of the number of records. In the red box, we can see over 80 customers who have Long and Medium Employment lengths with Credit History that are paid have a good Credit Standings. We can see in blue that an even larger number of customers, 100 with Short Employment and a Current Credit History (meaning they still have ongoing loans) have a Bad Credit Standing, perhaps they have missed some payments due to the nature of their short employment. From a financial services biased view this does make sense for both, will the model see the same without the bias?



3D Prop Table- Employment,Credit History,Credit Standing

Employment	Credit History	Credit Standing	Count
Long	All Paid	Good	51
Medium	All Paid	Good	33
No Infomation	All Paid	Good	1
Retired	All Paid	Good	0
Short	All Paid	Good	37
Unemployed	All Paid	Good	6
Very Short	All Paid	Good	10
Long	Current	Bad	26
Medium	Current	Bad	19
No Infomation	Current	Bad	4
Retired	Current	Bad	0
Short	Current	Bad	100
Unemployed	Current	Bad	7
Very Short	Current	Bad	37

Fig 2.3.2 3D Prop Table for Employment, Credit History and Credit standing.

2.4 Data Manipulation

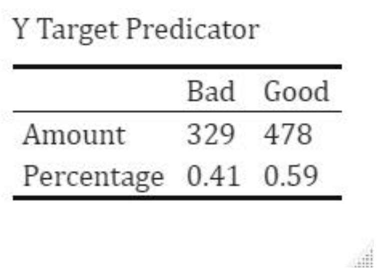
We performed some operations on the data which are listed below:

1) Drop the Id column as this is not useful for our modelling.
2) Fix blank space data in some categorical data columns:
    - Employment column, changed empty space to "No Information".
    - Personal Status column, changed empty space to "No Information".
    - Housing column, changed empty space to "No Information"
3) Change all character column types to as.factor types, which will be needed for modelling.
4) For Residence in Current Distinct take the smallest, a minus 2 value and change it to zero. You can't live in a place -2 time.
5) Round Age and drop the float values. Change to an int type
6) Change Months.Since.Acct.Opened to Years to match the units in the other two numeric columns.

3. Main Findings

3.1 Target Variable

We have been told the target variable is the Credit.Standing column, this is whether a person has a "Good" or "Bad" Credit rating. We will use this as the Y target variable when we create the models below. We can see that there are 329 Bad ratings which are 41% of the data and 478 Good ratings which are 59% of the data in Fig 3.1.1 and the Barplot of 3.1.2 shows this visually.



Y Target Predicator

	Bad	Good
Amount	329	478
Percentage	0.41	0.59

Fig 3.1.1 Y Target Predicator

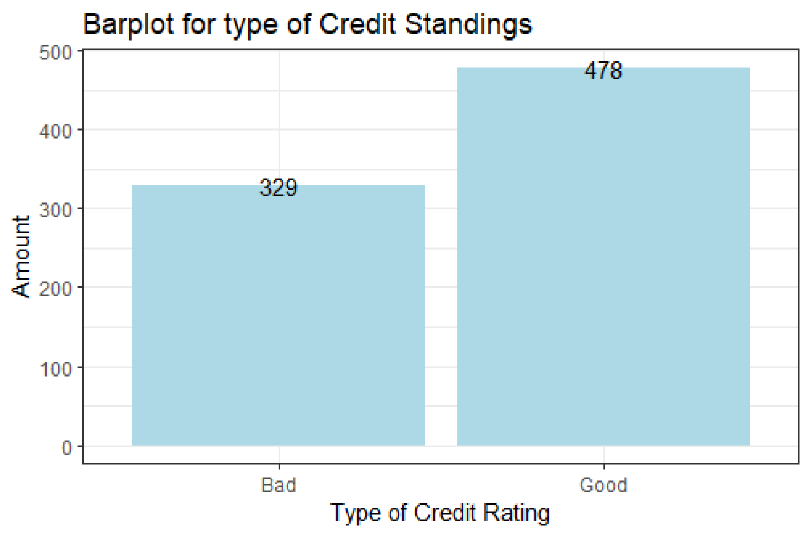


Fig 3.1.2 Barplot for type of Credit Standings

3.2 Train and Test data

The data has been divided into two random subsets on the seed value of the last three digits of my student ID (579), into data.train and data.test, with 75% of the data in the Train Data and the remainder of the data in the Test data. "data.train" has 607 observations and the "data.test" dataset has 202 observations, both over 13 columns, 12 predicator columns and 1 target variable (ID has been dropped as stated above as it will not be important for modelling).

Parts "c to g" of the assignment guidelines will be conducted on this "data.train" as this has been specified in the report in order to do binary splits on each of the predictor variables. "data.test" will be used to test the results of a Decision Tree on this manually split binary data.

I have also created "data.train.org" and "data.test.org" which is the original train and test data without the binary splits. In parts "h to j" of the guidelines we will use these unaltered data to train and test our Decision Trees and Random Forrest models in more detail. The reason for this is, if we train our models just on the train.data which we have converted into manual binary splits we will be limiting the model's abilities to find the best results. This will be discussed in more detail later.

3.3. Entropy Overview

For the first part of our project, we will be looking at the categorical and numeric columns and working out the best binary splits of each. The Categorical predictors have between two and ten different classes in each and the numeric predictors have between 7 and 57 values. There are different metrics used in Decision Trees like Gini Index and Entropy/Information gain. We will be using Entropy and Information Gain to look for the best splits for both category and numeric predictors.

3.3.1 What is Entropy and Information Gain

Entropy

Entropy is the randomness or disorder from the data that is being processed and is used as a metric to measure the level of impurity or level of unpredictability in data. Fig 3.3.1 shows three systems with different levels of impurity. The three systems in Fig 3.3.1 we first see a system that has an equal amount of red and green dots so the system is very impure. The next system has mostly green dots but a few red and is less impure and then the last system has all green which shows a pure system. Given the chance ,a Decision Tree has a greedy approach and would try to reach a state where each terminal node at the bottom of the tree is completely pure. However, this makes trees very large and does not lead to the best models.

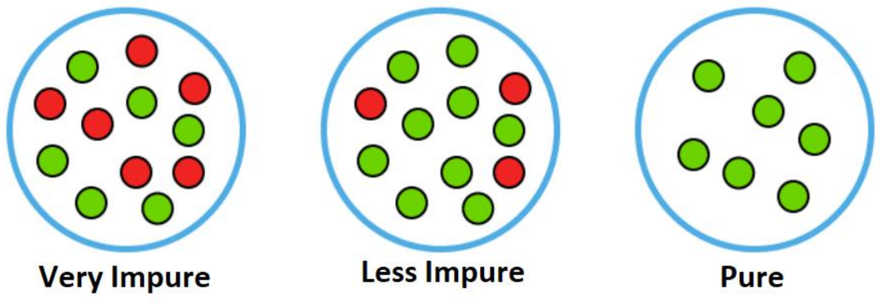


Fig 3.3.1 Impurity in systems

Entropy is calculated with the formula in Fig 3.3.2 and is a range between 0 and 1. With 0 or 1 being pure classes and the range in between being a level of impurity. “Pi” is the probability of a class “I” in our data. If there are just two classes “I” can be either negative or positive with a + or a – sign.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Fig 3.3.2 Entropy Formula

Example: If we had two class in 100 data points and 60 were positive and 40 were negative. The positive class would be 6/10 and the negative class would be 4/10. Using the above formula with Fig 3.3.3 we would get an entropy value of 0.78. This entropy value shows a high level of disorder or a low level of impurity in the system. With values for each class/predictor, we can see which one has the most disorder.

$$-\frac{4}{10} \times \log_2\left(\frac{4}{10}\right) - \frac{6}{10} x \log_2\left(\frac{6}{10}\right) \approx 0 \cdot 78$$

Fig 3.3.3. Formula

Information Gain

We now have Entropy which shows us the disorder of the system, we use Information Gain as a metric to look at how this is reduced in the target predictor given. In Fig 3.3.4 we can see the formula we use to take the entropy of Y given X from just the entropy of Y. Using the Information gain we can then decide which node to be the root node and start our tree on.

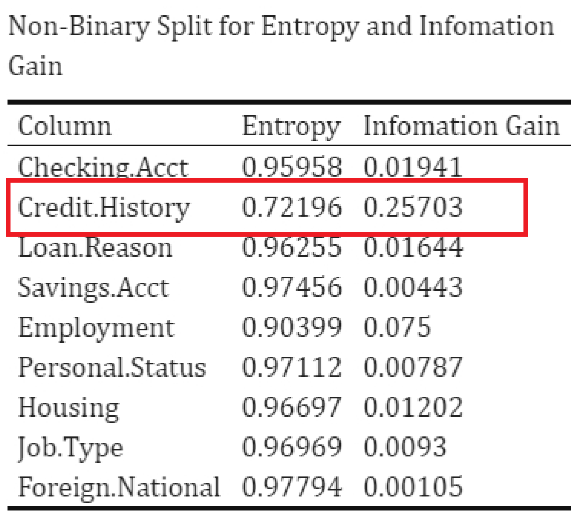
$$IG(Y, X) = E(Y) - E(Y|X)$$

Fig 3.3.3. Information Gain Formula

3.4 Entropy for Categorical Columns with Non-Binary Split

First, we will look at the entropy and information gain for the categorical columns but we will not be limited to any binary splits. A Decision Tree works in this way and it would take in each categorical and then decide on its metric (in our case Entropy/Information Gain) what predictor to use as the root node and then binary split on. Some of these predictors have a range of classes from 2 to 10, which must be brought down to a binary split. Note, there are other algorithms like ID3 that use more than binary splits.

Using the tabfun function from the lab tutorial as a base I created a for loop to go through each of the categorical predictors and get an Entropy and Information Gain which use the formulas above. You can see in Fig 3.4 which shows the Entropy and Information gain for each of the 9 categorical predictors. Credit History has the lowest Entropy and then the highest Information gain by quiet a margin. So this would be the Root Node for a Decision Tree and the first Node to split on.



Non-Binary Split for Entropy and Infomation Gain

Column	Entropy	Infomation Gain
Checking.Acct	0.95958	0.01941
Credit.History	0.72196	0.25703
Loan.Reason	0.96255	0.01644
Savings.Acct	0.97456	0.00443
Employment	0.90399	0.075
Personal.Status	0.97112	0.00787
Housing	0.96697	0.01202
Job.Type	0.96969	0.0093
Foreign.National	0.97794	0.00105

Fig 3.4 Entropy / Information Gain for Category Predicators

Since we are not restricted to binary splits, the next split could be Credit History again as we can see from the table no other predictor has quite the same values as Credit History. After that, the next best predictor column is for Checking Account.

3.5 Entropy for Categorical Columns with binary Split.

We will now look at taking the categorical columns as predictors and creating our own binary split using entropy. I looked at two ways to do this:

3.5.1 Method One:

Below in Fig 3.5.1, you can see where I created a for loop to get the entropy for each class in a predicator. You can see in this table the lowest entropy is for "No Account". I could now spilt on this but I have not taken into account how each combination might change the entropy values.

Checking Account
Class Entropy

	Entropy
0Balance	0.9980009
High	0.9993375
Low	0.9951724
No Acct	0.8860324

Fig 3.5.1 Checking Account class entropy

Fig 3.5.2 shows another example of getting the class entropy of Credit history for example. Here we see that "All Paid" and "Critical" have low entropies. (Of course, they are almost opposite on what we want and while not joining these two together would seem like the better result, we cannot add in our own bias). If you look at the Credit History Count we can see that there are different counts for each which is to be expected on how the entropy is calculated.

Credit History Entropy

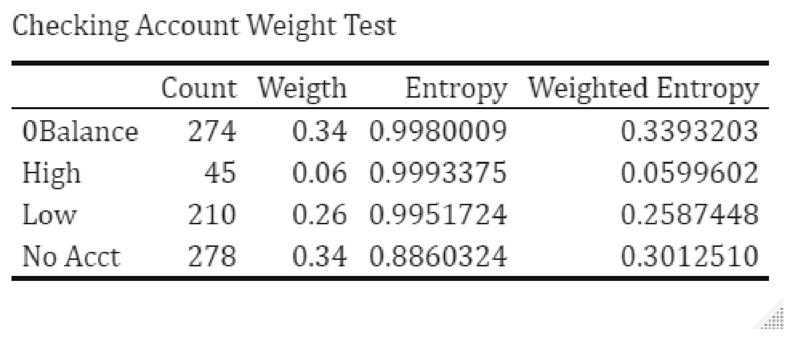
	Entropy
All Paid	0.2285380
Bank Paid	0.6880466
Critical	0.1044214
Current	0.9934271
Delay	0.9580420

Fig 3.5.2 Credit History class entropy

But with the above examples, we cannot just take one entropy value and then do a binary split on this. A decision tree would look at each of the classes to decide what to split on. Then it could still split on the same predicator again. We need to look at ways to combine splits and then later in another section we will talk about how the second node is chosen.

Dead End:

One area I was looking at (for binary splitting, second node later) was how I could use the weight of the values to create a better value for entropy. I ruled this out as the count and so the weight is already been taken into account in the Entropy formula. Because I multiplied the weight by the entropy I get a value that I don't believe makes sense, because now I would be looking at the higher value whereas entropy looks at the lowest value and as I mentioned the count has already been taken into account. I need another way, Method Two.


Checking Account Weight Test

	Count	Weigth	Entropy	Weighted Entropy
0Balance	274	0.34	0.9980009	0.3393203
High	45	0.06	0.9993375	0.0599602
Low	210	0.26	0.9951724	0.2587448
No Acct	278	0.34	0.8860324	0.3012510

3.5.2 Method Two: Improvement on Method One

For method one there is a disadvantage as I mentioned. I am only getting the entropy for each class but not taking into account how each entropy will be affected when I group them. This mean for the above Checking Account Example I am putting "No Acct" in one split and combining the other three classes, "OBalance", "High" and "Low" into the other side of the split. But thinking on this I might find that in reality a split between "High/Low" and "OBalance/No Acct" could be the best split for a binary split. In order to check this I would need to check each combination for entropy to see which is the lowest for a binary split.

I would need to take each combination of the 4 classes. And run the entropy again on all of these. Then see which binary split has the lowest entropy to allow me to manually split. So for example, I would need some splits like the below for Checking Account. And Load Reason was ten classes so a much longer list.

[OBalance] - [High, Low, NoAccount]	[OBalance, High] - [Low, NoAccount]
[High] - [0balance, Low, NoAccount]	[High, 0balance] - [Low, NoAccount]
[Low] - [0Balance, High, NoAccount]	[Low, 0Balance] - [High, NoAccount]
[NoAccount] - [0Balance, High, NoAccount]	[NoAccount, 0Balance] - [High, NoAccount]

3.5.3 Method Two: Pseudo Code idea

I tried to program this but was not able yet. I looked at for loop as well as some libraries (caret, itertools) to help but was not able to do it. Time restrictions and learning code limited the time I could spend on this.

I looked into the following way:

- For loop take each combination of levels:
    - Do entropy for each combination
    - Save results to a table and label
- Compare the results and take the entropy with the lowest value and their combination for a binary split

3.5.4. Binary Split for Category Results

I have taken Method One and chosen a manual spit for each of the categorical columns based on the returned lowest entropy for that class. I then use the function "fct_collapse" to collapse the data.train factors into new levels with only two factors. In a few cases with Loan Reason, I chose more than one class since there was a large amount, 10. You can see a list of the binary splits in Fig 3.5.4.1

PREDICATOR	**SPLIT 1**	**SPLIT 2**
CHECKING ACCT	"0Balance/Low/High"	"No Acct"
CREDIT HISTORY	"Bank Paid/Critical/Current/Delay"	"All Paid"
LOAN REASON	"Business/Car Used/Large Appliance/Other/Repairs/ Retraining/Small Appliance"	"Car New/Education/Furniture"
SAVINGS ACCT	"High/Low/MedHigh"	"MedLow"
EMPLOYMENT	"Long/No Information/Retired/ Sort/Unemployed/Very Short"	"Medium"
PERSONAL STATUS	"Divorced/Married/No Information"	"Single"
HOUSING	"Other/Own/Rent"	"No Information"
JOB TYPE	"Management/Skilled/Unskilled"	"Unemployed"

Fig 3.5.4.1 Binary Splits

Then I run the Entropy and Information Gain function again to see how my results compare to the non-binary splits from section 3.4. We can see in Fig 3.5.4.2 we have our new manual binary split table and now binary split table side by side. The red box shows the Credit History for both and we can see that the entropy value has gone up and the Information Gain value down. Is this that surprising? No, we have made the system more pure by reducing the amount of classes and disorder. We will see how this affects our models later.

Binary Split for Entropy and Infomation Gain

Column	Entropy	Infomation Gain
Checking.Acct	0.96	0.01899
Credit.History	0.86226	0.11673
Loan.Reason	0.97096	0.00803
Savings.Acct	0.97511	0.00388
Employment	0.9482	0.03079
Personal.Status	0.97617	0.00282
Housing	0.97842	0.00057
Job.Type	0.9711	0.00789
Foreign.National	0.97794	0.00105

Non-Binary Split for Entropy and Infomation Gain

Column	Entropy	Infomation Gain
Checking.Acct	0.95958	0.01941
Credit.History	0.72196	0.25703
Loan.Reason	0.96255	0.01644
Savings.Acct	0.97456	0.00443
Employment	0.90399	0.075
Personal.Status	0.97112	0.00787
Housing	0.96697	0.01202
Job.Type	0.96969	0.0093
Foreign.National	0.97794	0.00105

Fig 3.5.4.2 Compare Binary and Non-Binary splits

3.6 Entropy for the Numeric Values

Next, we get the entropy for the Numeric columns. There are three numeric columns that we can see from fig 3.6 that have the following values.

Number of Unique Values in each Numeric Prediator

	Count
Residence Time in current District	7
Years Since Account opened	31
Age	57

Fig 3.6. Number of unique values.

I investigated four methods to look for the best way to split the numeric columns into binary splits:

1) Pick three points at set points decided by the index
2) Separate by split by looking at the summary of the data and splitting by Q1, median and Q3
3) Take a random value
4) Brute Force Technique with computer power for all combinations.

Method One: Pick three points at set points decided by the index

In the labs, there was a way to look at splitting at a certain point by index. I looked into this but just by splitting on the index always gave me the same entropy. I did not think this would be the best approach but I was unsure why I was always getting the same entropy result but the labs were not. Even if just on the index, if I split on ¼ and ¾ I would have thought I would have gotten different values.

I added this to a To Do list to come back to later as it was the least important way I wanted to try and I was constrained by Time.

Method Two: Separate by split by looking at the summary of the data and splitting by Q1, median and Q3

A second method was to get a summary of a predictor and then to do a split on the 1st quartile, median and 3rd quartile. We will look at the results below. This a good approach as it takes values spread across the numeric columns based on the summary results.

Method Three: Take a random value

I saw an article online that gave the idea to split on a random value for a numeric predictor. I didn't save this reference so I lesson for the future. So I randomly took three values and did a split on those. Later I would like to make this into a function so I could check multiple runs and samples.

Method Four: Brute Force Technique with computer power – Not coded

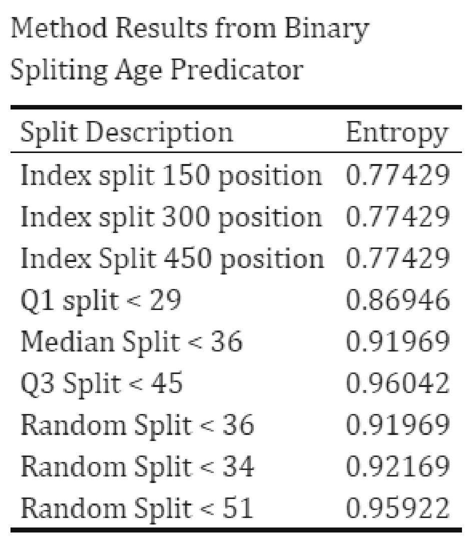
The last technique I had is very similar to the second idea for Categorical predictors and that was to get every split for all the unique values in the predictor. So If I had a numeric column with 1,2,3, I would split on [1] - [2,3], [2] - [1,3] and [3] - [1,2] and get the entropy for each. Our "Age" column has 57 unique values so this has the potential to take a bit of brute computer force but this is something we can do when the values are not too high.

I did not have time to do this yet. What also stopped me was also the article I read that said a random value can work just as well.

3.6.1 Example

Fig 3.6.1 shows the results of the first three methods. As I mentioned the index split is not working as planned and gives the same entropy value. While it is taking the index you would still have 25/100, 75/100 and 40/100,60/100 which should have given different results. On to do list to look at more at a later point.

The next three results "Q1 split", "Median Split and "Q3 split" use the second method. As we can see that the Q1 split has the lowest entropy and does better than the random three values. So this would be our split in this case.


Method Results from Binary Spliting Age Predicator

Split Description	Entropy
Index split 150 position	0.77429
Index split 300 position	0.77429
Index Split 450 position	0.77429
Q1 split < 29	0.86946
Median Split < 36	0.91969
Q3 Split < 45	0.96042
Random Split < 36	0.91969
Random Split < 34	0.92169
Random Split < 51	0.95922

Fig 3.6.1. Age Predicator

When I did the same process for Residence Time in Current District and Years Since the account opened, the random split got better results than method two: summary method. Of course, this could be a chance as well but that shows that the random approach was best two out of three times and would be worth further investigating.

3.6.2 Binary Split for Numeric Results

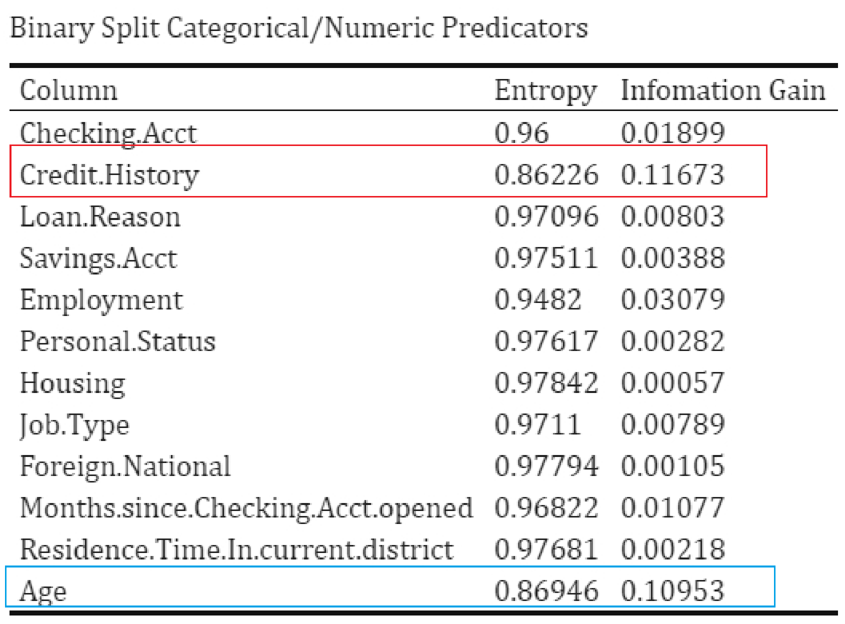
For the three numeric columns in Fig 3.6.2, the following splits were used:

PREDICATOR	**SPLIT**
AGE	Split on Under 29 years and Over 30 years
RESIDENCE TIME IN CURRENT DISTRICT	Split on under 12 months and Over 13 months
MONTHS SINCE ACCOUNT OPEN.	Split on Under 13 Months and Over 14 months

Fig 3.6.2 Numeric splits.

3.6.3 Entropy and Information Gain on Binary Splits.

Now we will run the entropy and Information Gain on the combined binary split for the Categorical and Numeric Predictors. In fig 3.6.3 we can see the lowest entropy in the red box is still for Credit history and the second best predictor is Age in the blue box.



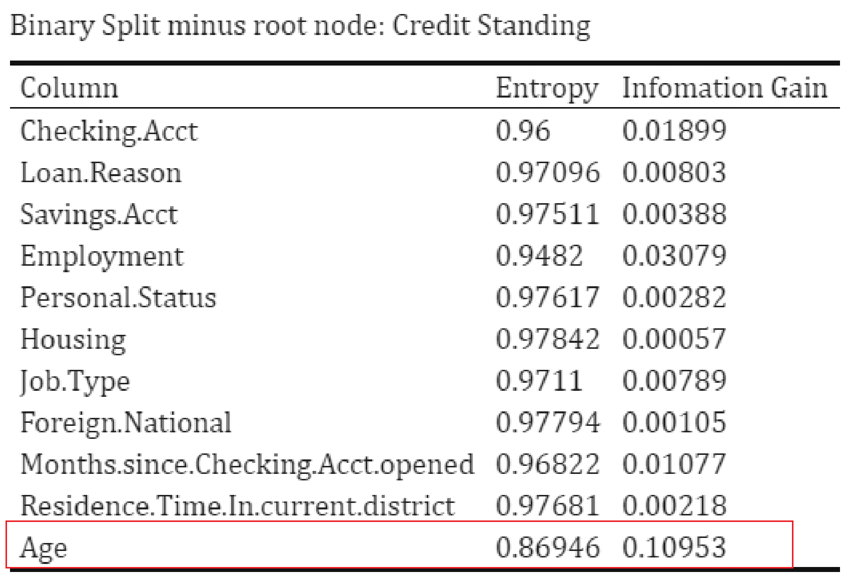
Binary Split Categorical/Numeric Predicators

Column	Entropy	Infomation Gain
Checking.Acct	0.96	0.01899
Credit.History	0.86226	0.11673
Loan.Reason	0.97096	0.00803
Savings.Acct	0.97511	0.00388
Employment	0.9482	0.03079
Personal.Status	0.97617	0.00282
Housing	0.97842	0.00057
Job.Type	0.9711	0.00789
Foreign.National	0.97794	0.00105
Months.since.Checking.Acct.opened	0.96822	0.01077
Residence.Time.In.current.district	0.97681	0.00218
Age	0.86946	0.10953

Fig 3.6.3 Binary split for all predicators

3.6.4 Next Node after the root node with binary split

The next node is then decided after the root node has been removed. In this case we can see below that the Credit Standing predicator is gone and now Age becomes the next node to be split on. However This is only on a binary split, what if we could use Credit History again?



Binary Split minus root node: Credit Standing

Column	Entropy	Infomation Gain
Checking.Acct	0.96	0.01899
Loan.Reason	0.97096	0.00803
Savings.Acct	0.97511	0.00388
Employment	0.9482	0.03079
Personal.Status	0.97617	0.00282
Housing	0.97842	0.00057
Job.Type	0.9711	0.00789
Foreign.National	0.97794	0.00105
Months.since.Checking.Acct.opened	0.96822	0.01077
Residence.Time.In.current.district	0.97681	0.00218
Age	0.86946	0.10953

Fig 3.6.4 Binary split for all predicators

3.7 Next Node after the root node with a non-binary split.

In a binary split once the predictor has been used it is no longer available to be used again. But in a non-binary split, if we split into two classes, the predicator still has other classes that can still be available for further splits. This is one of the advantages of non-binary splits. We have seen that Credit History is the best predictor to split on by a big margin. In Fig 3.7 we can see each of the classes again. If we split the "Critical" and "Current" classes and now removed these classes from consideration. When the entropy again for this predictor it could be highly likely that the entropy for this predictor would still be better than the next best predictor, Age. So we would split on this Predicator again. We will see an example of this later.

Credit History Entropy

	Entropy
All Paid	0.2285380
Bank Paid	0.6880466
Critical	0.1044214
Current	0.9934271
Delay	0.9580420

Fig 3.7 Credit History class entropy

3.8 Code for 3.7

I cannot do the code for section 3.7 yet, but it would be a continuation of the methods I have mentioned before where we look for each of the combinations of classes as a first step. This also made me realise just how complex a Decision Tree is. That it was not just taking into account a binary split but also splits in each of the classes in a predicator.

If we get the entropy of each combination of classes we would then compare each of the splits for the predictors and pick the root node with the predictor that has the lowest entropy. We would then look at which class inside that predicator had the lowest entropy and remove those classes from that predictor so that on the second split if the entropy for that predictor was still the lowest out of all the predictors we could split again.

It may be a new predictor or it could also be the same predictor again but for different results on the combinations of each class, minus the root node removed from consideration.

3.7 Measure Performance

The next step is to run some Decision Trees on the original Data and on the Binary split data to see how the models compare. Before we run our models for Decision Trees and Random Forests we will talk about how we will measure performance. We will do this by using a confusion matrix to measure our model’s performance. Fig 3.7.1 shows the layout of a confusion matrix (Ref 3.6.1). The rows show the actual target and each column shows a prediction for the target. We are looking for the total of True Negative and True Positive divided by the total of everything else.

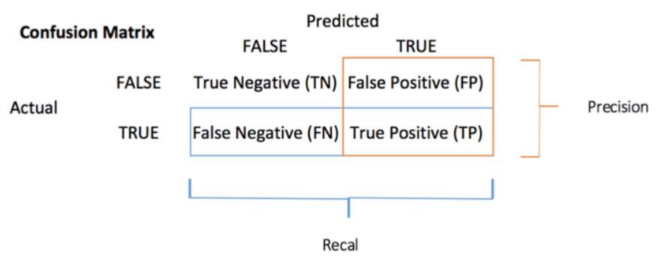


Fig 3.7.1 Confusion Matrix.

The rows show the actual target and each column shows a prediction for the target. We are looking for the total of True Negative and True Positive divided by the total of everything else. Fig 3.7.2 shows some notation of how this looks

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Fig 3.7.2 Accuracy Formula

In some models, you will see this this formula used to calculate the Accuracy but also a function call ConfusionMatrix that saves the manual input which was taken from the Caret library.

3.8 Decision Tree

Decision trees are one of the most popular Machine Learning models used due to their low cost, speed, and ability to deal with noise and deal with missing data. They are also the basis for Extreme Gradient Boosting and Random Forests. It gets its name as it uses a branching method to get all the possible outcomes. You can use different algorithms to do this with CART being the most popular as it uses a binary split of nodes, but there are others like ID3 that allows more than binary splits.

Fig 3.8.1.1 (ref3.8.1.1) shows what a Decision tree looks like and also how it gets its name as it looks like the roots of a tree, or some people say tree branches. You have a root node at the top and then that splits down to further branches until it ends in root nodes. There are different ways to work out splits including Gini index and Entropy/Information Gain.

Decision trees act like how a person thinks and so are easy to understand, are fast, don’t require as much data cleaning and are very useful for decision-based problems. Some disadvantages are that they can overfit if they are too large as they are “greedy” and would get every branch down to pure classes if given the chance and for lots of classes they can get more complex.

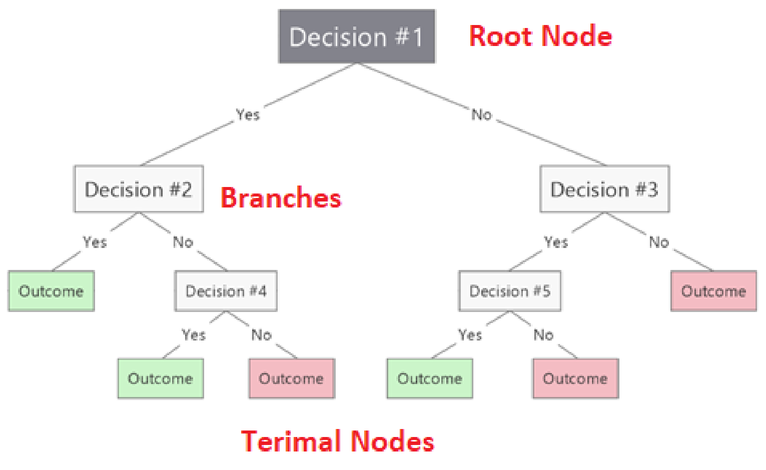


Fig 3.8.1.1 Decision Tree Structure

Hyperparameters

Decision trees can be greedy and will search for the best results for the current node and not necessarily find the best result globally. If no parameters are set then the tree will continue to grow until all the leaves have the same class. This is wasteful so we can use hyperparameters to apply stopping criteria to help tune the tree for the best results. We will look at four of these parameters which you can see in Fig 3.8.1.2 (Ref 3.8.1.2).

In our code, we will use the Tree Library as well as rpart to look at different results. There are other libraries like MLR.

- **MinSplit**: - sets the number of entries in a node before it split. If this minimum is not met, it will not split. In Fig 3.8.1.2 you can see when the minsplit value is increased from 4 to 5, the branches are reduced and on the right side, we no longer have a split to nodes 1 and 2.
- **Maxdepth**: sets the depth a tree will split to. In the diagram, you can see the left has a depth of 1 and the right has a depth of 3. As mentioned a tree will continue until all classes are pure if given the chance.
- **Cp**: Stands for Complexity Parameter which uses a cp value so that if the value is less than the set value the tree will not split further. If this split does not improve the complexity of the tree, then do not split. In the right you can see the value is below 0.1 and so the tree does not split to the next branch.
- **Minbucket**: If splitting would give fewer entries then the node will not be split. So the minbucket value of 5 spots splits down to 4,1,2 nodes.

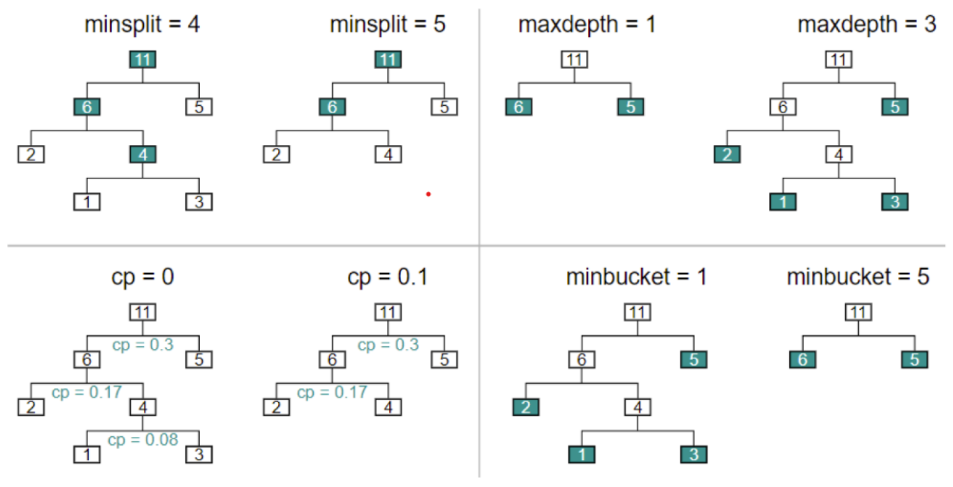


Fig 3.8.1.2: Hyperparameters

Cross Fold valuation and Pruning

Cross-fold validation is a way to check the validation of our models by using groups on the dataset. Our data is divided into K-folds (a parameter we can set), which then sets the number of groups our data is divided into train and validation groups. If the k value is set to 5, then the data is divided into 5 groups and one becomes the validation set. Then this is ran 5 times so each group gets the chance to be the validation group. It's like a pre-test test but does not use the actual test data.

Pruning deletes unnecessary nodes in order to get the best tree. Since a Decision tree is greedy, a tree with too many nodes will be in danger of overfitting but also a small tree may not capture all the important nodes for the best tree. Pruning lets you increase the size of a tree, and then delete unnecessary nodes without losing accuracy. Pruning is done with Cost Complexity Pruning (mentioned above) and Reduced Error Pruning.

3.8.1 Decision Trees on Binary Split data

We have now taken the original data.train and changed all the predictors into binary splits. Below you can see in Fig 3.8.1.3 we have run a Decision Tree Model and we can see the Decision Tree has run with the Credit History Predictor as the root node. We have used a default Decision Tree and it has used 5 Predictors [Credit History, Age, Employment, Job Type, Residence.Time.in.current.District] and ended with 7 terminal nodes.

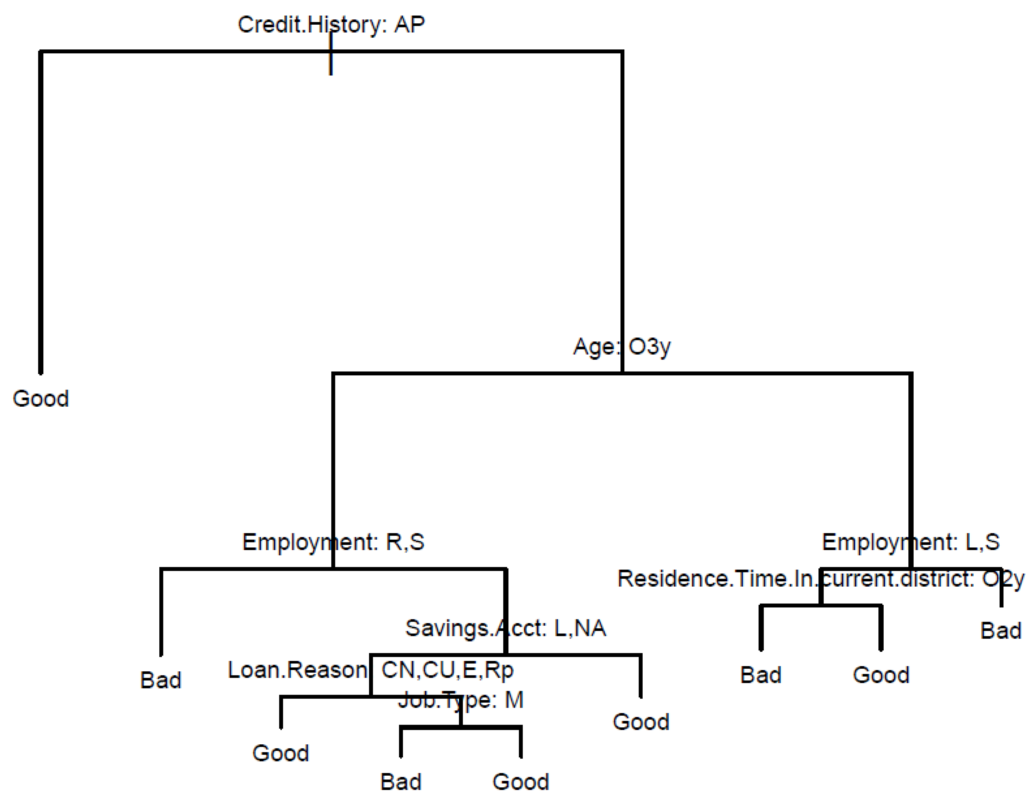
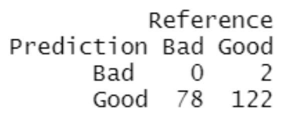


Fig 3.8.1.3 Decision Tree on Manual Binary Predicators

NOTE: Because I have changed the data.train numeric predictors to factors in order to make them a binary predictor I cannot use the original data.test to build a prediction model because this original data has the numeric column as numeric and prediction will not allow this. Since I was told in the report to change the data.train data to binary and not all the columns I saw two workarounds:

1) Do Binary splits on all the data – I did not do this as the report guidelines said to only do binary splits on the training data.
2) The second workaround is to only change the numeric column to the same factors when I changed the numeric columns to binary. This will allow the model to 'overfit' (overfit is not done on individual columns but it will predict these parts easier) for these columns but the categorical columns will not have changed. I suspect the results will show slightly better accuracy results because of this. But I needed to have a way to compare this model to a model on the raw data. Besides the binary split data.train is always going to do worse than the original data. More on that later.

Next, we will use a confusion matrix to test the predicted results against the test data. We can see that the model has been very good at predicting Good outcomes and has guessed 122 of the Good outcomes but has been very bad at guessing the Bad outcomes. This matches what we see in the Tree Plot in Fig 3.8.1 where we see one side of the root node goes to Good as a terminal Node. This gives us an accuracy percentage of 0.6039, Fig 3.8.2.



```
          Reference
Prediction Bad Good
      Bad    0    2
      Good  78  122
```

Fig 3.8.2 Model Results

Let's look at a Decision Tree ran on a model with the raw data that has not been binarized.

3.8.2 Decision trees on original Data

We saw in section 3.8.1 that we got a 0.60 accuracy result for our manual binary split predictors. Now we will let the Decision Tree itself take the original data with all the levels in the factors and all the numeric values in the numeric predictors. We mentioned previously we saved a copy of the data as "data.train.org" for this purpose.

3.8.2.1 Decision Tree: Basic Tree: dt_model_1

Running the tree on the default settings for a tree gives us the plot in Fig 3.8.2.1 which has 8 predictors[Credit History, Age, Employment, Residence.Time.in.current.District, Savings.Acct, Loan.Reason, Checking.Acct, Years.since.Chekcing.Acct.Opened], it has one more predictive over the non-binary model and 12 terminal nodes. Also straight away we can see that this time it has switched the Good and Bad for the root node and now Bad is at a terminal node after the root node(orange circle). We can see with the Red boxes and blue boxes that Credit History and Employment are used at different points in the tree (and not just on the same node level).

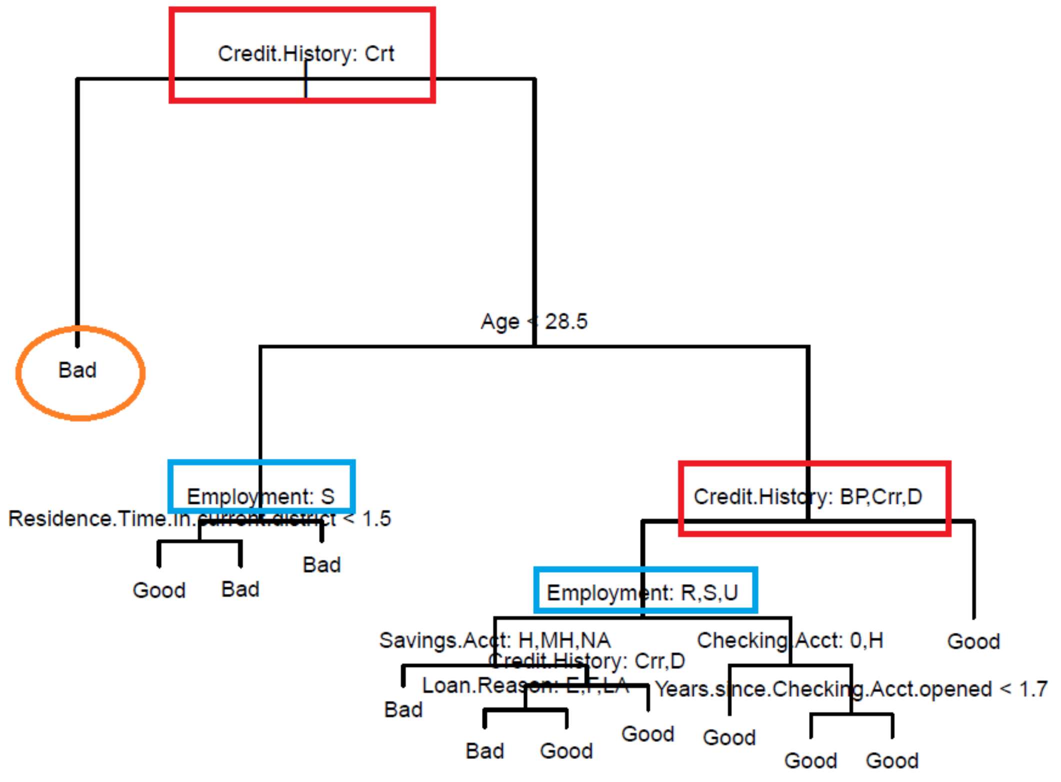


Fig 3.8.2.1 Model : dt_model_1 tree

Running a confusion matrix again gives an accuracy of 0.7277. A very large improvement over the first model.

Comparing our binary model and non-binary model.

From running these first two models. When we use the manually created predictor nodes we get an accuracy of 60% and when we use the raw data without creating binary splits we get an accuracy of 73%.

What does this tell us? It tells us that by manually changing the Predicators to binary splits we are giving the Decision Tree fewer “options” to run Trees on and this has a large effect on the accuracy of our models. Not only are we manually setting the entropy but because we are using binary splits the Decision Tree no longer has the option of using the classes within a predictor a second time if this is beneficial.

So manually creating binary splits should be avoided as it is adding in basis to the models.

3.8.2.2 Decision Tree: dt_model_2

Dt_model_2 uses the hyperparameters mentioned above to look at improving the models. Various values for minsplit, maxdepth and minbucket were used. In Fig 3.8.2.2 you can see just some of the values that were tried. Many more variations were run.

MODEL PARAMETERS	ACCURACY
CP = 0.01, MINSPLIT=20, MINBUCKET=5, MAXDEPTH=10	0.7475
CP = 0.05, MINSPLIT=20, MINBUCKET=10, MAXDEPTH=20	0.7129
CP = 0.03, MINSPLIT=20, MINBUCKET=10, MAXDEPTH=20	0.7129
CP = 0.02, MINSPLIT=20, MINBUCKET=10, MAXDEPTH=20	0.7277
CP = 0.005, MINSPLIT=10, MINBUCKET=20, MAXDEPTH=30	0.7277

The best model had an accuracy of 0.7475 so an improvement on the default model of 0.7277. Also, we used the rpart to make this decision tree to further explore other Decision Tree Options. Rpart also has a different plot diagram which we can see in Fig 3.8.2.2

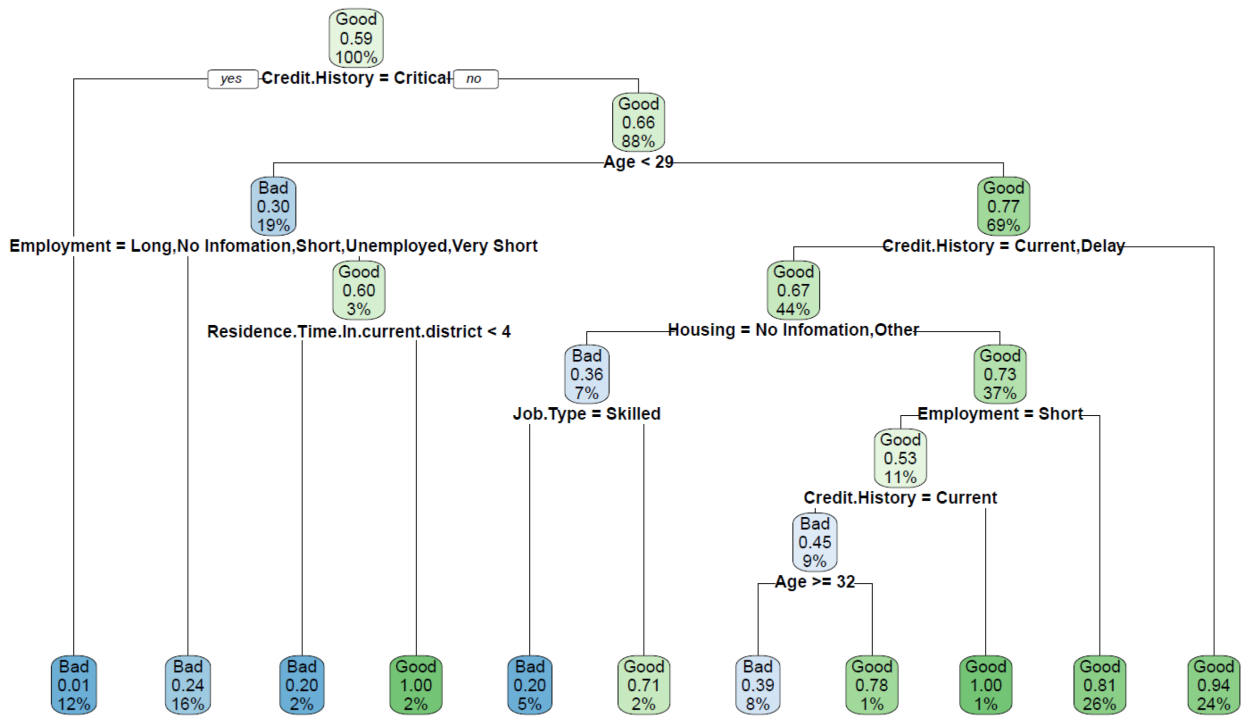


Fig 3.8.2.2 Model : dt_model_2 best model

3.8.4 Cross Fold Validation and Pruning: dt_model_3-4

We have already used Cost Complexity Pruning In the hyperparameters but next, we will use it with Cross Validation to prune the tree. In fig 3.8.4 we can see the tree has been pruned back to only the root node with an accuracy of 0.7277. Although this is a drop In accuracy from our hyperparameter best of 0.7475 often it is better to have a simpler model and with only one branch it's hard to argue it's not better for only one branch and two terminal nodes.

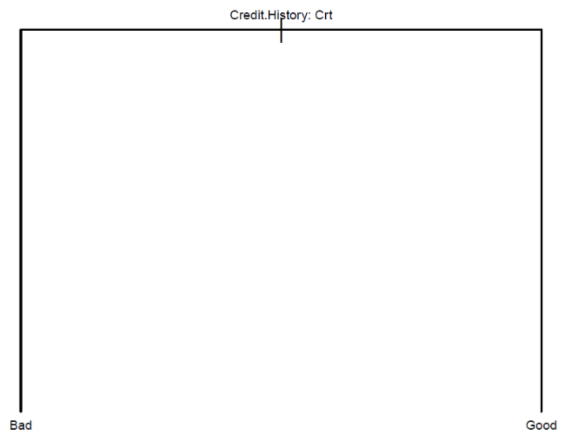


Fig 3.8.2.2 Model : dt_model_3 Best Model

3.9 Random Forests

Random Forests are built on Decision Trees, they are made up of a large number of random decision trees. It is called "Random Forest " as it randomly chooses what predictors to use, It will not always pick the best predictor and each tree in a random forest gives a result and the trees with the most votes become the best model. Often, the more trees that are ran the better the results. Since a random forest runs many trees and randomly takes predictors, some results will be wrong but since it runs so many, many more will be correct.

Random Forest takes less time compared to some other models, their predictions have high accuracy, and for very large datasets it still runs well, they are built on Decision Trees and so work well with missing data and are good at preventing overfitting.

Random forests are not so good at generalizing cases with completely new data. If you have 1 chocolate bar that costs 1 euro, 2 chocolate bars cost 2 euros, and 3 chocolate bars cost 3 euros, how much are 5 chocolate bars? A linear regression model would easily be able to calculate this in comparison. Random forests can also be biased towards categories that have multiple classes.

Hyperparameters

Hyperparameters for Random Forests have less of an effect on Random Forest than say Parameters on Decision Trees but there still are many we can use. Some of the more important are:

- **N-tree** –the number of trees running in the forests.
    - Please note: In my code, I have reduced the number of trees so that when the code is run it will not spend hours creating models with a large number of trees.
- **Mtry** – The number of random variables sued in each tree.

3.10.1 Random Forrest Model: rf_model_1

Rf_model_1 is a basic Random Forrest Model run with the default parameters and scores an impressive 0.8115 accuracy value. A large improvement on the 0.7475 we saw when we tuned the hyperparameters on a single decision tree. Credit History was still the most important predicator with Age being second.

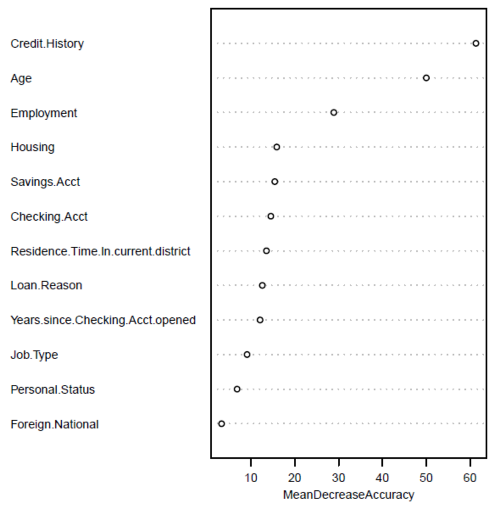


Fig. 3.10.1 Predictor Importance.

3.10.2 Random Forrest Model: rf_model_2

Next, we will look at some other models in Random Forest and try different parameters to see if we can improve the default random forest model. Many models were run and below are some examples. The last model is the best we have found so far and it was run 100,000 times and gave an accuracy of 0.8165. The best model we found overall.

MODEL PARAMETERS	ACCURACY
NTREE=1,000	0.8016
NTREE=10,000	0.8066
NTREE=100,000	0.8132
MTRYSTART=3	0.8148
MTRYSTART=2	0.8046
MTRYSTART=4, STEPFACTOR=4	0.8082
MTRYSTART=3, STEPFACTOR=3, NTREE=100000, DO.TRACE=TRUE	0.8165

Note: Previously we mentioned that parameters for Random Forest do not improve models as much, we can see above the best model accuracy is 0.8165, only a small improvement on 0.8.115 for the default settings for a random forest.

3.11 GDPR Models

Decision Trees

Maeve has been told because of GDPR she can no longer use the Age, Foreign National or Personal Status predictors. Since Age was our second best predictor in Decision Trees we would think this would affect the modelling but as we say we got very good results for a tree with only the root node of Credit History so it may not have an effect on decision trees. In Fig 3.11.1 we can see a plot of the tree with Credit Standing still as the root node and also has Credit Standing as the second node and twice more lower down the tree.

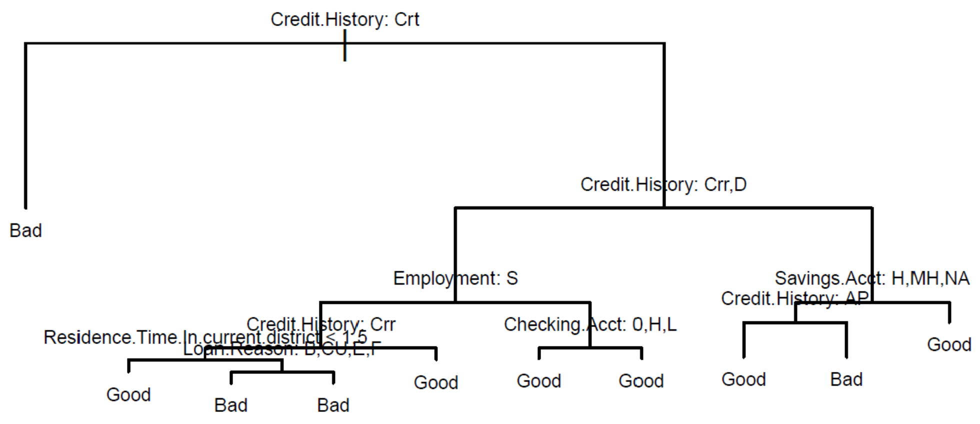


Fig. 3.11.1 Plot of GDPR Decision Tree

```
          Reference
Prediction Bad Good
      Bad   43   10
      Good  35  114
```

Fig. 3.11.2. Accuracy results.

The accuracy for this model is 0.7772 which is very interesting as it is higher than previous decision trees. I ran this model with both the Tree and Rpart Decision Trees and got the same results.

Random Forest

Running a new random forest minus the GDPR predictors we can now look at which are the most important. Previously we saw that Age was the most important with Credit History a close second. Now with Fig 3.11.3, we see Credit History is the most important Predicator by a large amount. This does make sense as when we looked at the Decision Tree Model above it used Credit History 4 times In its tree.

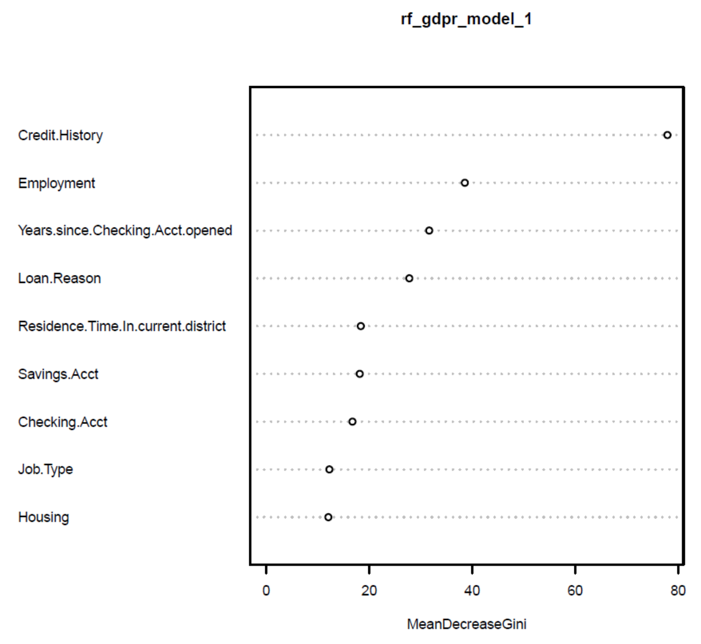


Fig. 3.11.3 Accuracy results.

Model results for this random forest model are not much lower than that we saw with all the predictors and is 0.8016. So not much of a drop for losing three predictors.

3.12 Issues with Grading System

Since this is Supervised Learning, labelling of the data is very important for training models as the Y predictor is used as the target output. If during a period, whoever was labelling customers with a Good or Bad credit rating was mislabelling or a fault in the system was mislabelling the Y target variable this can affect how well the model trains which then will affect new data that we want to predict on.

3.12.1 Credit Score Grading System and Human Input

A credit rating is based on the information given in credit reports, which are taken from different credit organisations as well as Maeve's own. It would not be uncommon for organisations like Maeve's to use other sources to help determine this source which then leads to a Credit Rating score within Maeve's own organisation. This score would include how much has been borrowed, if payments were made on time, if loans were paid back, etc. A customer is then given a Grade with A for customers who have paid back all payments and loads on time down to F for someone who has defaulted or declared bankruptcy.

However, there can be issues with credit grading systems beyond the personal issues of having not paid back previous loans. While the aim is to remove bias and only use the facts of past history, there has been a long history of discrimination that credit grading systems have used which can be influenced by race and background. Mistakes can also happen in data scores which can affect a credit grading system. These issues are part of the reason for the EU's GDPR rules and regulations.

Human input can also have issues. Humans can make clerical errors and can mistakenly label a customer with a bad credit rating and they can also be subject to bias and give a label that would also be wrong. These issues can result in data that is then mislabelled, any model can then suffer from biases when the models are learning. This then further leads to issues with the labelling of new customers based on the training data.

3.12.2 Periods of bad or mislabelling.

1) **Clustering.**
   What Mave is looking for is periods where the model performed very poorly. By looking at clustering models based on the ID she can look for areas that group together based on the ID. If she starts to see any area of ID that have been labelled similar she can further investigate these areas.

2) **Time series**
   Since we can use the ID as timestamp values Maeve could also use time series analysis to look for periods the same as above based on a time-stamp where once again data was mislabelled.

3.12.3 Time Series plot of the data.

Using the ID as a time series and creating a new column where I convert the Credit Standing values of Good to 1 and Bad to 0 we can do a time series plot to look to see if there any areas that look of interest that might indicate some of the things mentioned above. There does appear to be the case and you can see in the red boxes in Fig 3.12.3 where there have been periods were more Bad or Good scores have been given. These can be the start point for Maeve to investigate more.

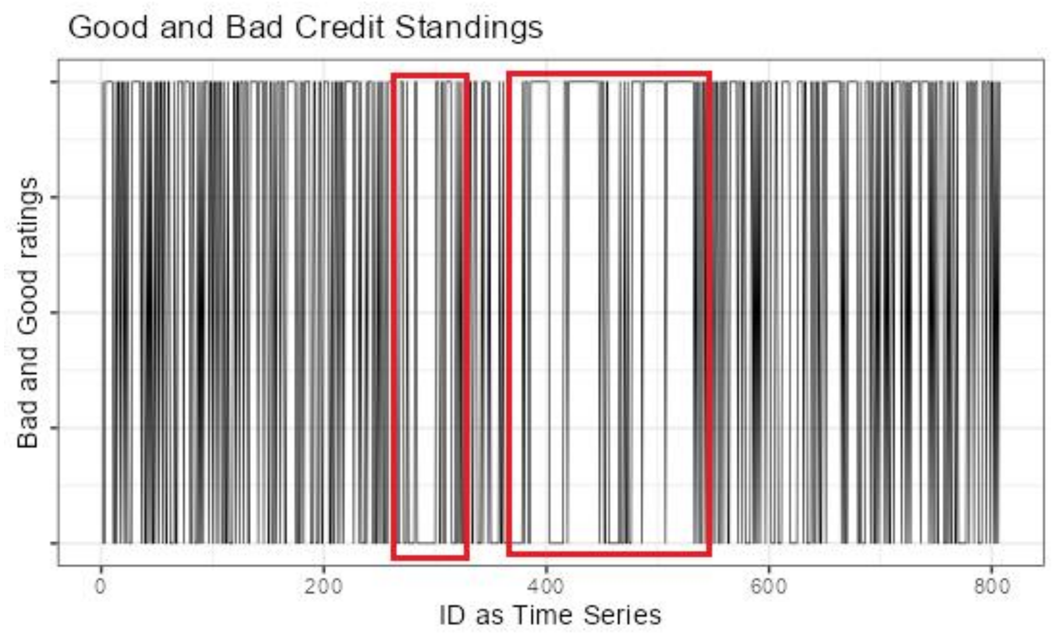


Fig 3.12.3 Time Series of Credit Standing

3.12.4 Anomaly Detection System

What we could also do is create an anomaly detection system (ref3.12.4), which could look at the data and look for spikes or dips in the Credit Standing Ratings. This article is for python but gives some ideas on how an anomaly detection system would work and different techniques that could be used.

Thinking on it, I'd look to potentially use a window system that takes a rolling average of Good and Bad ratings over time and look for anomalies periods where there are long spikes in Good ratings or dips in Bad ratings. You can see from Fig 3.12.3 that should a system would show a dip around the ID 300 mark and a very large spike of Good Ratings between ID 400 to 500

4. Conclusions

For this project, we looked at creating binary splits on the data at Maeve organisation. What we saw was that by manually creating binary splits we are limiting the impact of our models to create accurate models. We saw that when we ran a Decision Tree on our manually split binary data we got an Accuracy of 60% and when we ran a Decision Tree on the raw data without doing any binary splits we received a much better model score of 73%. A large increase.

We also saw that when we did not use binary splits the model was able to use the classes within a predictor more than once at different points in a tree which greatly improved the accuracy of the model.

We ran a number of Decision Trees and Random Forest models on the raw data. What we saw was hyperparameters had more of an effect on Decision trees than parameters had on Radom Forest models. We also saw that by running a large number of Random Forests this almost offered models with better accuracy results than on Decision Trees. Maeve was able to produce models with up towards almost 82% accuracy.

Maeve also briefly looked at time series models of the data and can pinpoint several periods that appear to show times when the Credit Rating system may have been mislabelling customers with good or bad ratings. In the future, she will look into this more and look to create an anomaly detection system that can be used to pinpoint when this happens again.

5. References

Ref 2.5 - https://careerfoundry.com/en/blog/data-analytics/multivariate analysis/#:~:text=There%20are%20two%20types%20of,the%20structure%20of%20a%20datasetRef

Ref 3.6.1 - https://www.guru99.com/r-decision-trees.html

Ref 3.8.1.1 https://blog.mindmanager.com/decision-tree-diagrams/

Ref 3.8.1.2 https://rpubs.com/Haibiostat/DecisionTreesSimpleShowCase).

https://rpubs.com/Haibiostat/DecisionTreesSimpleShowCase

REf3.12.4 https://medium.com/bukalapak-data/time-series-anomaly-detection-simple-yet-powerful-approaches-4449ffe1ca12