Capstone Project- Credit Card Default- Taiwanaese Company

Uzair Faruqi

2020-11-24

Credit Card Default Data- for a Taiwanese Company

In This Project I have Worked with the Taiwanese Credit card Data set. This data set has 30000 rows and 24 columns. The objective is to estimate the probability of default payment by credit card client using the data provided. These attributes are related to various details about a customer, his past payment information and bill statements. The Data set if available on the github repository https://github.com/uzaifaru2/Credit-Card-Default it is also available on the UCI Machine Learning repository.

I am anyways picking up the data from the repository and hence the code should run directly

Synopsis

The Objective of the assignment is to predict whether a particular customer will default payment next month or not. The result is a an extremely valuable piece of information for the bank to take decisions regarding offering credit to its customer and could affect the bank's revenue.

The data set is a tricky one as it has a mix of categorical and continuous variables also it is a classic case of an imbalanced data set and hence any model developed on the entire data set would have an element of bias as defined by the overall Accuracy which these models which many of the Machine learning Models will try and Maximize.

I would be be trying to address this issue using various means. First I will try and develop a normal logistic regression model to predict the probability default, measuring the probability at two different benchmarks of 0.5 and 0.25 and see the overall accuracy as well as the specificity and sensitivity.

Then next I would be developing a decision tree model using the rpart library to achieve the objective. In the decision tree I would be utilizing these 4 methods available in the rpart package to address the issue 1. Under Sampling the training set 2. Changing the prior probability 3. Assigning a Cost to Default 4. Changing the weights to default vs non Default. data As we see that when we develop a model on the entire set , the Accuracy might seem good but the specificity or Sensitivity gets compromised.

With the Various versions of the respective Tree models as we see that we can balance the Objectives in terms of selecting the models which has a Better Sensitivity or Specificity.

We will round it off by developing a Boosting model i.e a Gradient Boosting model, Multi Variate Adaptive Recursive Slime - Better Known as earth and Adaboost and see the confusion matrix developed on the same. The Boosting models have all been trained on the Undersampled Training set where we have made attempts to balance the data set. I have also tried to develop the model on the full data set. I would be comparing the performance of the various models or the various approaches applied.

As we see that we would have to strike e a balance between the Specificity and the Sensitivity of the models to achieve the Objectives .

Loading the necessary libraries and Extracting the Data

```r
library(tidyverse)
library(broom)
library(caret)
library(ipred)
library(lattice)
library(ggplot2)
library(caret)
library(MASS)
library(car)
library(knitr)
library(mice)
library(lattice)
library(reshape2)
library(skimr)
library(ROSE)
library(rpart)
library(rpart.plot)
library(earth)

data <- read.csv('https://raw.githubusercontent.com/uzaifaru2/Credit-Card-Default/main/credit_card_defa
```

```r
# Exploring the data

dim(data)
```

[1] 30000 25

```r
# We are having 30,000 records with around 25 columns - variables ( including a customer ID column)
# The defaults  have been encoded as 1 and 0 otherwise


# All the variables including the customer ID seems to have been considered as integers



# Looking at the data to see if any Inconsistency

table(data$EDUCATION)
```

```
0     1     2     3     4     5     6
```

14 10585 14030 4917 123 280 51

```r
# There are data with 5 and 6 defined for Education  which hasn't been defined in the data
# data understanding document. We might have to group them under 4 which  is others

table(data$MARRIAGE)
```

```
0     1     2     3
```

54 13659 15964 323

```
# here also we have many under 0 , which has not been defined , we can put it under 3 Which is Others


#replace 0's with NAN, replace others too
data$EDUCATION[data$EDUCATION == 0] <- 4
data$EDUCATION[data$EDUCATION == 5] <- 4
data$EDUCATION[data$EDUCATION == 6] <- 4
data$MARRIAGE[data$MARRIAGE == 0] <- 3

# Checking on whether changes have been Made

table(data$EDUCATION)
```

```
1     2     3     4
```

10585 14030 4917 468

```
table(data$MARRIAGE)
```

```
1     2     3
```

13659 15964 377

About the Data

There are around 30,000 data points i.e around 30,000 unique data points All the data including the customer ID have been considered as integers We would have to make the necessary corrections, converting some of the data into factors and some of the variables might have to be one hot encoded so that we can run the models relate to the Boosting Algorithms

Creating Derived Variables

One of the Important Tasks of a Data scientist is to understand the data and to work on making certain derived variables which would be more effective as variables then the Variables themselves

For Example the Billing Amount or the credit balance might not be so useful themselves say as compared to say the Ratio of Billing Amount to the Credit balance .

There could be many such variables that can be derived and used in the model developed to improve on its accuracy. Much of it depends on the ingenuity of the Data Scientists and the understanding of the domain by them.

Also while we might derive many variables there would have to be certain data engineering to be done to address anomalies etc

```
# Let us work out certain Derived variables
# One can be as innovative as possible  in Deriving these variables and these variables
# Can actually effect the performance of the model itself also.

# The Derived Variables that we can look at are
# The increase in the Billing amount in september as compared to August
# The Billing amount in a percentage of the Credit limit
# Payment to Bill AMNT Ration

data2 <-data %>% mutate( inc_billing_amnt = (BILL_AMT1- BILL_AMT2)/BILL_AMT2 ,
                         bill_amnt_Limit = BILL_AMT1/LIMIT_BAL,
                         pay_bill_amnt_ratio = PAY_AMT1/BILL_AMT2)


# Now Another Problem , what needs to be done for cases which are  Giving NA's or Inf/-inf

# One Option is to remove these variables another option might be to replace the NA's by
# 0's  or the maximum values
# Infy can be handled by replacing the divisor ( if it was a 0 by the median value) but it will end
# up changing a variable itself which  we should avoid.
# Depending on the prevalence of these values

# write.csv(data2, file = "data2.csv")

# We are going to replace the NA's by 0 and the Infinities by the maximum value in that column


max(data2$inc_billing_amnt[is.finite(data2$inc_billing_amnt)])
```

```
[1] 13009.38
```

```
data2$inc_billing_amnt[data2$inc_billing_amnt == Inf] <- max(data2$inc_billing_amnt
                                                        [is.finite(data2$inc_billing_amnt)])
data2$inc_billing_amnt[data2$inc_billing_amnt == -Inf] <- min(data2$inc_billing_amnt
                                                          [is.finite(data2$inc_billing_amnt)])
data2$inc_billing_amnt[is.na(data2$inc_billing_amnt) ] <- 0

# managing the Infinity Values for the  pay_bill_amnt_ratio both the negative and positive Infinities


max(data2$pay_bill_amnt_ratio[is.finite(data2$pay_bill_amnt_ratio)])
```

```
[1] 4444.333
```

```
min(data2$pay_bill_amnt_ratio[is.finite(data2$pay_bill_amnt_ratio)])
```

```
[1] -497.8
```

```
data2$pay_bill_amnt_ratio[data2$pay_bill_amnt_ratio == Inf] <- max(data2$pay_bill_amnt_ratio[is.finite(d
data2$pay_bill_amnt_ratio[data2$pay_bill_amnt_ratio == -Inf] <- min(data2$pay_bill_amnt_ratio[is.finite
data2$pay_bill_amnt_ratio[is.na(data2$pay_bill_amnt_ratio) ] <- 0





# Checking for NA's

sapply(data2, function(x) sum(is.na(x)))
```

```
             ID           LIMIT_BAL                 SEX           EDUCATION
              0                   0                   0                   0
       MARRIAGE                 AGE               PAY_0               PAY_2
              0                   0                   0                   0
          PAY_3               PAY_4               PAY_5               PAY_6
              0                   0                   0                   0
      BILL_AMT1           BILL_AMT2           BILL_AMT3           BILL_AMT4
              0                   0                   0                   0
      BILL_AMT5           BILL_AMT6            PAY_AMT1            PAY_AMT2
              0                   0                   0                   0
       PAY_AMT3            PAY_AMT4            PAY_AMT5            PAY_AMT6
              0                   0                   0                   0
default_payment    inc_billing_amnt     bill_amnt_Limit pay_bill_amnt_ratio
              0                   0                   0                   0
```

Data Engineeringa and creating the Training set and the Test Set

Now we would be working on making the data ready for the models being developed. This would Involve handling the NA's if any, Doing One Hot Encoding since I intend to use the Boosting algorithms to measure their efficacy. Handling the Numerical data as well as factors since though the target variable is stored as a 1 or a 0 it is stored as integer which we might have to convert into factors for the Algo to work properly.

```r
# lets remove the ID columns from the data set
data2 <- data2[ , -1]




# We see that certain variables which  should be categorical are stored as integers or
# numeric manner
# These should be converted into factors for further analysis
# The variables which  should be factors are " Sex", " Education " , " Marriage",
# Using them as numeric would give a false indication of their relevance for the End
# Many of the variables like the History of the payment status can also be used as numeric
# but as numeric they are not significant as conveyed by the correlation matrix and
# hence we are using  # these are factors


names <- c(2:4)

data2[,names] <- lapply(data2[,names] , factor)
# str(data2)
#One-Hot Encoding
# Creating dummy variables is converting a categorical variable to as many binary variables as
#mhere are categories.
dmy <- dummyVars(" ~ .", data = data2,fullRank = T)

train_transformed <- data.frame(predict(dmy, newdata = data2))
# # See the structure of the new dataset
# str(train_transformed)

#Converting the dependent variable back to categorical
train_transformed$default_payment<-as.factor(train_transformed$default_payment)
```

```r
# checking the same and the other variables
#str(train_transformed)

#Splitting training set into two parts based on outcome: 75% and 25%
set.seed(7)
index <- createDataPartition(train_transformed$default_payment, p = 0.75, list=FALSE, times = 1)
trainSet <- train_transformed[ index,]
testSet <- train_transformed[-index,]
```

Now we have the Training set Ready and the Test Set, we will start with developing a Simple Logistics regression model and see the Results and we can also observe the Implications of an Imbalanced data

Logistic Regression Model

Logistic regression is the First model usually attempted in any Classification problem. It is the Linear regression equivalent model as far as classification problem is concerned.

In short, Logistic Regression is used when the dependent variable(target) is categorical. For example:

To predict whether an email is spam (1) or not spam (0) Whether the tumor is malignant (1) or not (0)

It is named as 'Logistic Regression', because it's underlying technique is quite the same as Linear Regression. There are structural differences in how linear and logistic regression operate. Therefore, linear regression isn't suitable to be used for classification problems.

Its name is derived from one of the core function behind its implementation called the logistic function or the sigmoid function. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

```r
log.model <- glm(default_payment ~., data = trainSet, family = binomial)

summary(log.model)
```

Call: glm(formula = default_payment ~ ., family = binomial, data = trainSet)

Deviance Residuals: Min 1Q Median 3Q Max
-3.1122 -0.6900 -0.5502 -0.2768 3.8256

Coefficients: Estimate Std. Error z value Pr(>|z|)
(Intercept) -8.034e-01 1.073e-01 -7.484 7.19e-14 ***LIMIT_BAL -1.500e-06 2.299e-07 -6.524 6.85e-11***
SEX.2 -1.361e-01 3.561e-02 -3.822 0.000133 ***EDUCATION.2 -8.311e-02 4.121e-02 -2.017 0.043705***
EDUCATION.3 -1.356e-01 5.544e-02 -2.447 0.014425 *
EDUCATION.4 -1.115e+00 2.173e-01 -5.131 2.88e-07 *MARRIAGE.2 -1.785e-01 4.030e-02 -4.429 9.48e-06* **MARRIAGE.3 -1.278e-01 1.498e-01 -0.853 0.393689**
AGE 5.434e-03 2.173e-03 2.501 0.012398 *
PAY_0 5.907e-01 2.063e-02 28.628 < 2e-16 *PAY_2 8.221e-02 2.384e-02 3.448 0.000566* **PAY_3 1.004e-01 2.660e-02 3.773 0.000161** *PAY_4 1.356e-02 2.889e-02 0.469 0.638798*
PAY_5 5.860e-02 3.122e-02 1.877 0.060540 .
PAY_6 -8.698e-03 2.584e-02 -0.337 0.736407
BILL_AMT1 -4.647e-06 1.371e-06 -3.390 0.000699 **BILL_AMT2 3.967e-06 1.682e-06 2.358 0.018353 ***
BILL_AMT3 7.636e-08 1.513e-06 0.050 0.959737
BILL_AMT4 1.131e-06 1.499e-06 0.755 0.450428
BILL_AMT5 -7.089e-07 1.722e-06 -0.412 0.680613
BILL_AMT6 7.665e-07 1.378e-06 0.556 0.578029

PAY_AMT1 -1.425e-05 2.624e-06 -5.431 5.61e-08 *PAY_AMT2 -7.568e-06 2.338e-06 -3.237 0.001208* ***PAY_AMT3 -3.483e-06 1.998e-06 -1.744 0.081241 .***
PAY_AMT4 -2.245e-06 1.906e-06 -1.178 0.238892
PAY_AMT5 -2.088e-06 1.898e-06 -1.100 0.271249
PAY_AMT6 -1.794e-06 1.480e-06 -1.213 0.225255
inc_billing_amnt 2.621e-05 8.337e-06 3.143 0.001671 *bill_amnt_Limit -3.550e-01 7.354e-02 -4.828 1.38e-06* ** pay_bill_amnt_ratio -9.398e-05 1.400e-04 -0.671 0.502046
— Signif. codes: 0 '*' ***0.001*** '' *0.01* '' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

```
Null deviance: 23779  on 22499  degrees of freedom
```

Residual deviance: 20765 on 22470 degrees of freedom AIC: 20825

Number of Fisher Scoring iterations: 5

Summarising From The Model

By Creating the Model and Looking at the Summary of the Model we can see which of the variables are important , i.e the ones with the stars

- Bill Amount to the Limit Ratio
- INcrease in Billing Amount
- Payment Amount 2 - Amount Paid 2 months back
- Payment Amount 1 - Payment made 1 Month Back
- Bill Amount last Month
- Bill Amount 2 months back
- Marriage
- Education
- Increase in Billing Amount as compared to previous month
- Increase in Billing Amount as a Ratio

These variables are important as far as creating and running a model is concerned.

I had run the model on these set of Important variables as well as the Model with all the variables . There wasnt any noticeable difference in the output and hence sticking with all the variables for this project.

```r
# Two of the derived variables etc

log.predictions <- predict(log.model, testSet, type="response")

log.predictions.rd <- ifelse(log.predictions > 0.5, 1, 0)
log.predictions.rd <- as.factor(log.predictions.rd)



con_matrix_log_0.5 <- confusionMatrix(log.predictions.rd,testSet$default_payment)

# With This Type of Mode The Sensitivity Seems to be High but the  Precision seems to be lower.
# Accuracy is around 80 % but that could be the case , but we could have got an accuracy of around
# 75 % by predicting non Defaults
# Let us see how we can change things , by Changing the default probability from 0.5 to say 0.25
```

```r
log.predictions.rd1 <- ifelse(log.predictions > 0.25, 1, 0)
log.predictions.rd1 <- as.factor(log.predictions.rd1)




con_matrix_log_0.25 <- confusionMatrix(log.predictions.rd1,testSet$default_payment)




# Extracting the Accuracy , Sensitivity and Specificity from the Confusion Matrix Created

log.accuracy.0.5 <- con_matrix_log_0.5$overall[["Accuracy"]]
log.accuracy.0.25 <- con_matrix_log_0.25$overall[["Accuracy"]]
log.specificity.0.5 <- con_matrix_log_0.5$byClass[["Specificity"]]
log.specificity.0.25 <- con_matrix_log_0.25$byClass[["Specificity"]]
log.sensitivity.0.5 <- con_matrix_log_0.5$byClass[["Sensitivity"]]
log.sensitivity.0.25 <- con_matrix_log_0.25$byClass[["Sensitivity"]]
```

The Output of the Logistic Regression

The Accuracy obtained is around 80.6 % and the Specificity 23.6 % for the cut off @ 0.4

The Accuracy obtained is around 75.6 % and the Specificity increases to 54.8 % for the cut off @ 0.4

Here for our case the the Positive is non default and the negative is default (Not Referring to the 1's and 0s')

Hence the specificity measures our efficiency in terms of identifying out of the total number of defaulters how well we have been able to identify them.

We would be evaluation various measures to do that specifically using various measures as given below

Tree Based Models

Now we would be working with some Tree based Models to see how they perform

Sensitivity and specificity both increases depending on the cut off chosen , accuracy Decreases, it might all depend on the priority of the organization

These are the typical characteristics of an imbalanced data set

We will now go through a set of measures designed to address these type of Imbalanced data

These Measures are -Undersampling (Since the Data set is sufficiently large we can as of now avoid Over sampling) -The Undersampling would be done only on the Training Set -Changing the Prior Probabilities -Including a Loss Matrix

These are some of the methods that can be utilised to address this problem of imbalanced data there are many other methods also.

Creating an Undersample Data set from the Training Set We would be using the Rose Library of R to do this and to develop the respective Machine learning model.

First Let us Look at the Proportions of the Default in the Training Set

```r
table(trainSet$default_payment)
```

```
0     1
```

17523 4977

```r
prop.table(table(trainSet$default_payment))
```

```
 0      1
```

0.7788 0.2212

we See that the Proportion of defaults is around 22 % , which is a Unbalanced Dataset we will try and Create an UNder sample Data set where the Proportion of defaults and the Proportion of Non Defaults are equal and we would be training our models on this Balanced Data set and the testing would be done on the Test Data set Developed

```r
train_under_sample <- ovun.sample(default_payment ~ ., data = trainSet,
                                  method = "under", N = 9954, seed = 1)$data
table(train_under_sample$default_payment)
```

0 1 4977 4977

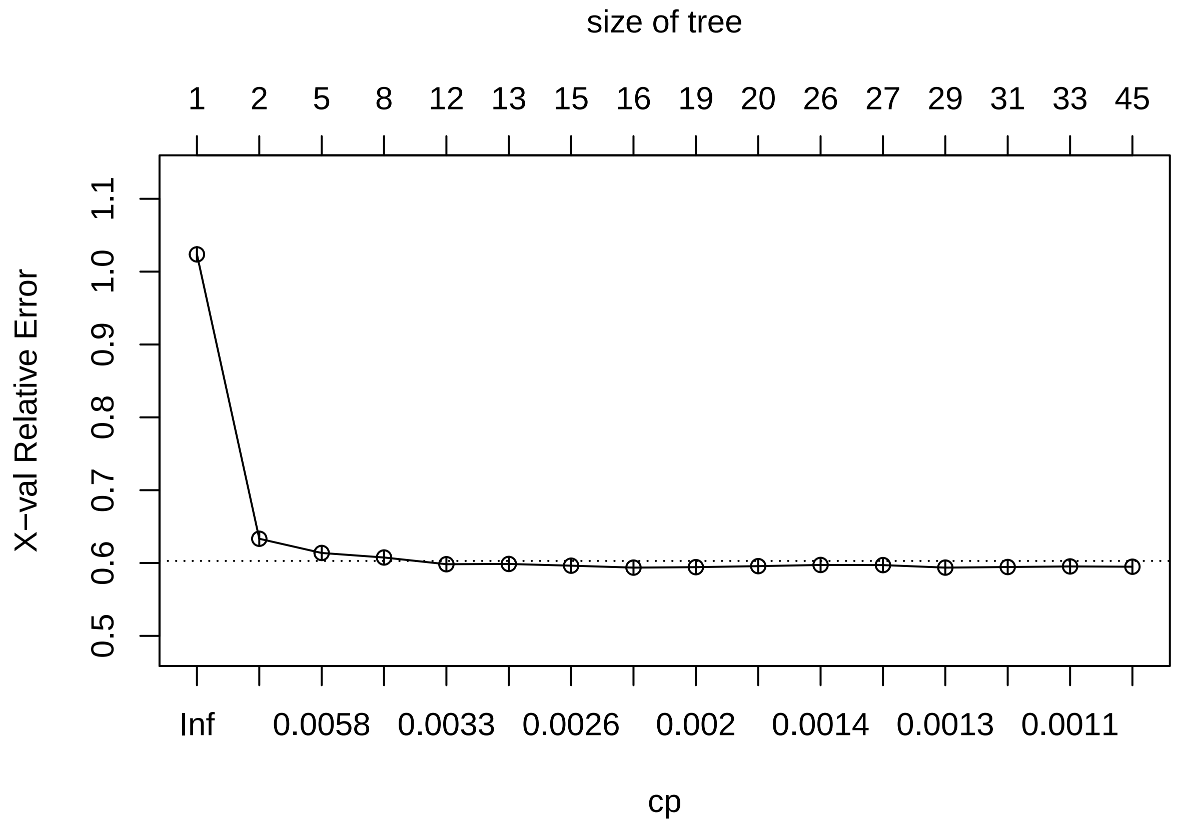
Here we are building the models using the undersample data set

```r
# Now Lets  start Building  Tree Based Models and observe whether we are able to improve on
# The Accuracy or the other measures like Specificity

tree_undersample <- rpart(default_payment ~ ., method = "class",
                          data =  train_under_sample, control = rpart.control(cp = 0.001))

# Now Let us Plot the Tree's

#plot(tree_undersample, uniform = TRUE)

# Add labels to the decision tree
# text(tree_undersample)

# I am avoiding PLotting the Trees , since the Image isn't coming out
# properly in the RMD generated File ,
# it Can be seen properly in a normal R Studio Console
```

Pruning the Trees

We are now pruning the trees to have least error and to avoid over fitting

```r
# Now Lets  Prune the tree for the Model tree_undersample

# Plotting the cross-validated error rate as a function of the complexity parameter
plotcp(tree_undersample)
```



```r
# Using  printcp() to identify for which complexity parameter
# the cross-validated error rate is minimized.

printcp(tree_undersample)
```

Classification tree: rpart(formula = default_payment ~ ., data = train_under_sample, method = "class", control = rpart.control(cp = 0.001))

Variables actually used in tree construction: [1] AGE bill_amnt_Limit BILL_AMT1 BILL_AMT2
[5] BILL_AMT3 BILL_AMT4 BILL_AMT5 BILL_AMT6
[9] EDUCATION.2 inc_billing_amnt LIMIT_BAL PAY_0
[13] PAY_2 PAY_3 PAY_4 PAY_6
[17] PAY_AMT1 PAY_AMT2 PAY_AMT3 PAY_AMT4
[21] PAY_AMT5 PAY_AMT6

Root node error: 4977/9954 = 0.5

n= 9954

```
       CP nsplit rel error  xerror      xstd
```

1 0.3666868 0 1.00000 1.02371 0.0100203 2 0.0076351 1 0.63331 0.63331 0.0093249 3 0.0043534 4 0.61041 0.61382 0.0092455 4 0.0038176 7 0.59735 0.60759 0.0092192 5 0.0028129 11 0.57685 0.59835 0.0091791 6 0.0027125 12 0.57404 0.59875 0.0091808 7 0.0024111 14 0.56862 0.59634 0.0091702 8 0.0021097 15 0.56620 0.59373 0.0091586 9 0.0018083 18 0.55957 0.59433 0.0091613 10 0.0014467 19 0.55777 0.59574 0.0091676 11 0.0014065 25 0.54692 0.59735 0.0091747 12 0.0013060 26 0.54551 0.59715 0.0091738 13 0.0012055 28 0.54290 0.59373 0.0091586 14 0.0011051 30 0.54049 0.59453 0.0091622 15 0.0010046 32 0.53828 0.59534 0.0091658 16 0.0010000 44 0.52381 0.59494 0.0091640

```r
# Creating  an index for of the row with the minimum xerror
index0 <- which.min(tree_undersample$cptable[ , "xerror"])

# Creating tree_min
tree_min <- tree_undersample$cptable[index0, "CP"]

#  Prune the tree using tree_min
ptree_undersample <- prune(tree_undersample, cp = tree_min)
```

Changing the Prior Probabilities

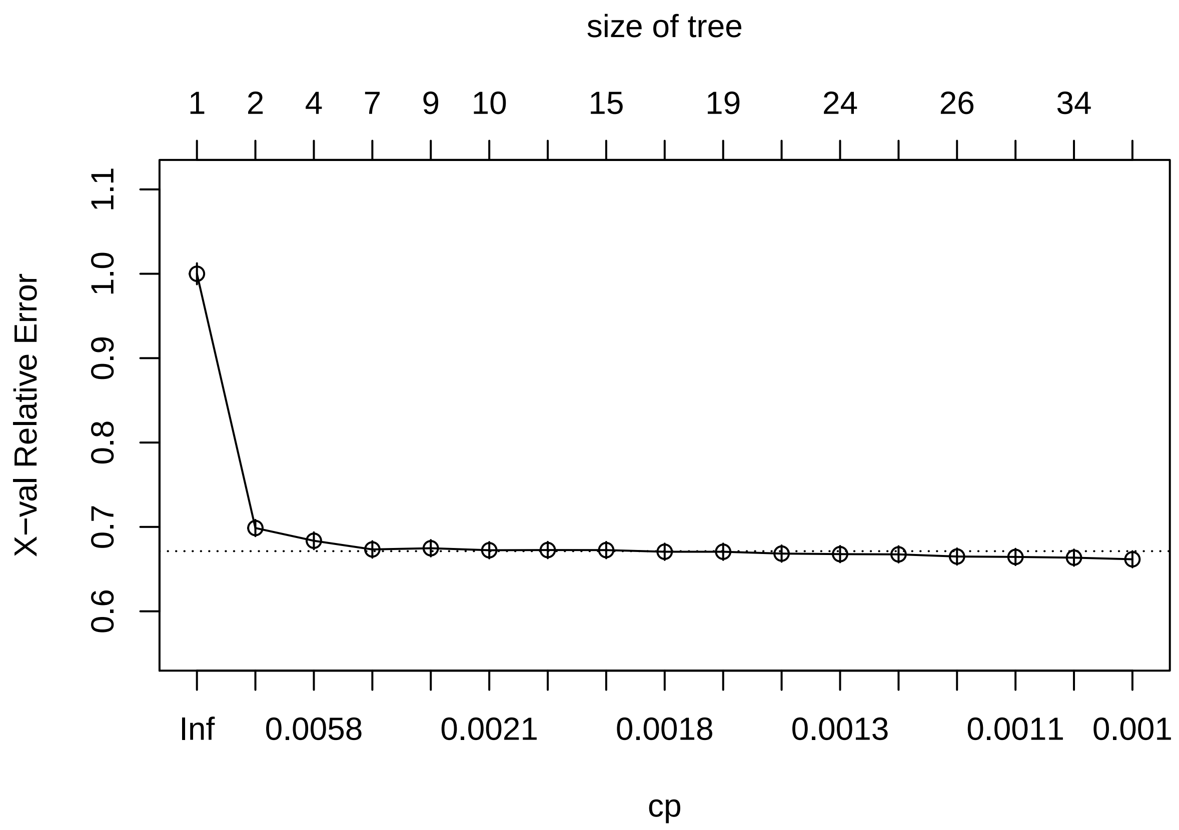
in the rpart package the prior probabilities by default are taken as per their proportions , we can sort of trick R by modifying the same in the parms function to say 60/40

```r
tree_prior <- rpart(default_payment ~ ., method = "class",parms = list(prior=c(0.6, 0.4)),
                    control = rpart.control(cp = 0.001),
                    data = trainSet)

# Plot the decision tree
# plot(tree_prior, uniform = TRUE)

# Add labels to the decision tree
# text(tree_prior)
```

Pruning the Tree

Pruning the Tree for the Model with the Changed Prio Probabilities

```r
# As the Trees get more complex, they are not clearly visible( The Nodes)

# Pruning the Trees with Changed Prior Probabilities

    # Plot the cross-validated error rate as a function of the complexity parameter
 plotcp(tree_prior)
```



```r
# Use printcp() to identify for which complexity parameter the
# cross-validated error rate is minimized.
printcp(tree_prior)
```

Classification tree: rpart(formula = default_payment ~ ., data = trainSet, method = "class", parms = list(prior = c(0.6, 0.4)), control = rpart.control(cp = 0.001))

Variables actually used in tree construction: [1] AGE BILL_AMT1 BILL_AMT4
[4] inc_billing_amnt LIMIT_BAL MARRIAGE.2
[7] PAY_0 PAY_2 PAY_3
[10] PAY_4 PAY_5 PAY_AMT1
[13] PAY_AMT2 PAY_AMT3 PAY_AMT4
[16] PAY_AMT5 pay_bill_amnt_ratio

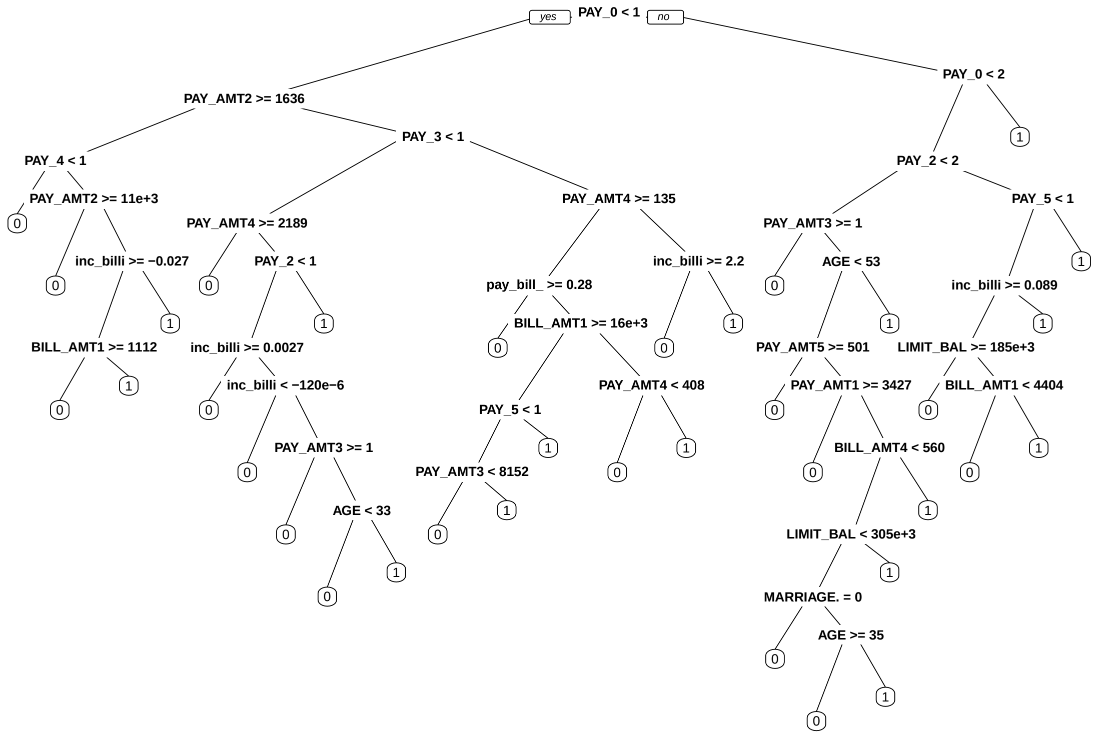
Root node error: 9000/22500 = 0.4

n= 22500

```
    CP nsplit rel error  xerror      xstd
```

1 0.3012985 0 1.00000 1.00000 0.0125092 2 0.0100265 1 0.69870 0.69870 0.0096775 3 0.0033079 3 0.67865 0.68359 0.0100397 4 0.0025746 6 0.66872 0.67330 0.0098142 5 0.0022597 8 0.66358 0.67474 0.0097367 6 0.0019117 9 0.66132 0.67237 0.0097038 7 0.0018915 12 0.65545 0.67259 0.0097267 8 0.0018500 14 0.65166 0.67253 0.0097278 9 0.0017414 15 0.64981 0.67056 0.0097216 10 0.0016745 18 0.64459 0.67056 0.0097216 11 0.0013292 22 0.63789 0.66838 0.0097104 12 0.0012840 23 0.63656 0.66777 0.0096641 13 0.0011984 24 0.63528 0.66748 0.0096589 14 0.0011105 25 0.63408 0.66491 0.0096518 15 0.0011060 29 0.62906 0.66442 0.0096523 16 0.0010082 33 0.62448 0.66357 0.0096511 17 0.0010000 34 0.62347 0.66166 0.0095991

```r
# Create an index for of the row with the minimum xerror
index <- which.min(tree_prior$cptable[ , "xerror"])

# Create tree_min
tree_min <- tree_prior$cptable[index, "CP"]

#  Prune the tree using tree_min
ptree_prior <- prune(tree_prior, cp = tree_min)

# Use prp() to plot

prp(ptree_prior)
```



Tree with a Loss Matrix Included

We are now developing a Tree by including a loss matrix where we are penalizing a Default 10 Times more as compared to Non Default. An Arbitrary Number chosen. The Company can choose any other number depicting the actual loss that they might face in case of a default.

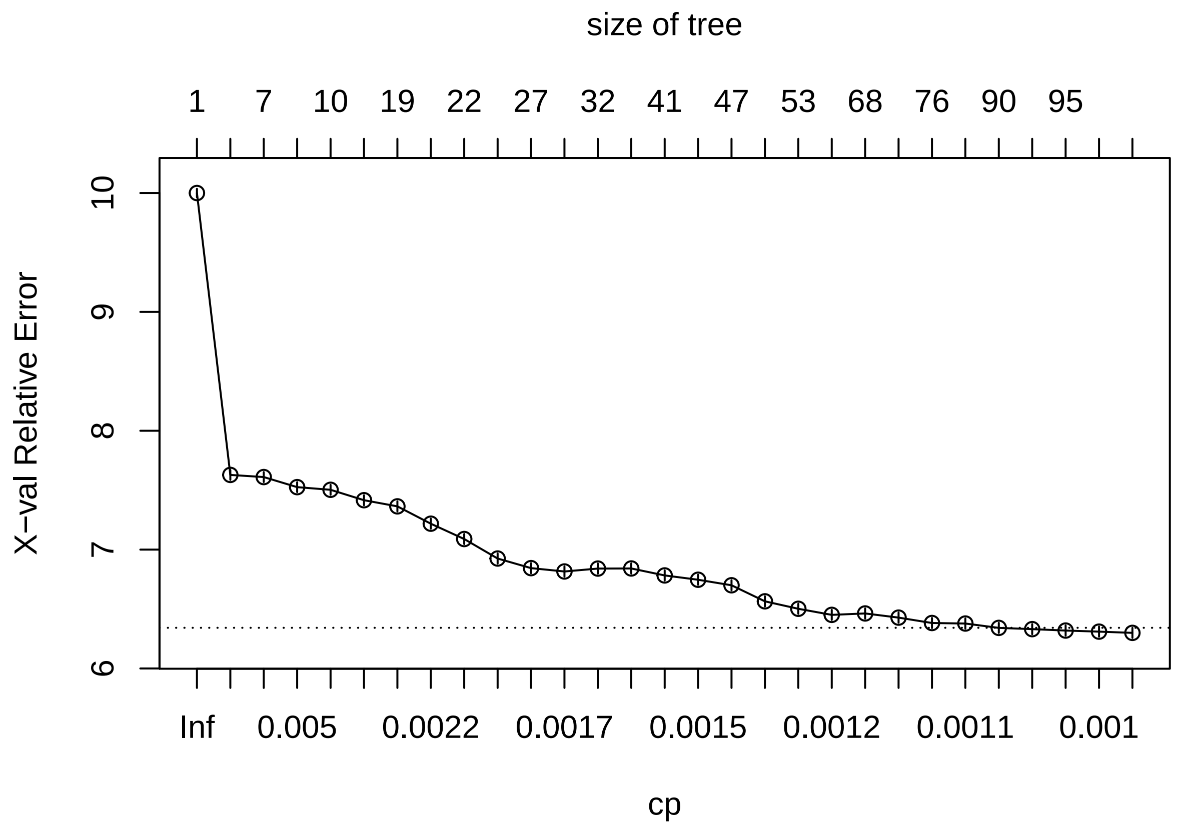
```r
tree_loss_matrix <- rpart(default_payment ~ ., method = "class",
                          parms = list(loss = matrix(c(0, 10, 1, 0), ncol=2)),
                          control = rpart.control(cp = 0.001), data =  trainSet)

# Plot the decision tree
```

```
#plot(tree_loss_matrix, uniform = TRUE)

# Add labels to the decision tree
#text(tree_loss_matrix)
```

Pruning the Tree

Pruning the Tree for the Model developed by including a Loss Matrix

```
# Pruning the tree with the loss matrix

set.seed(345)
tree_loss_matrix  <- rpart(default_payment ~ ., method = "class", data = trainSet,
                           parms = list(loss=matrix(c(0, 10, 1, 0), ncol = 2)),
                           control = rpart.control(cp = 0.001))

# Plot the cross-validated error rate as a function of the complexity parameter
plotcp(tree_loss_matrix)
```



```
printcp(tree_loss_matrix)
```

Classification tree: rpart(formula = default_payment ~ ., data = trainSet, method = "class", parms = list(loss = matrix(c(0, 10, 1, 0), ncol = 2)), control = rpart.control(cp = 0.001))

Variables actually used in tree construction: [1] AGE bill_amnt_Limit BILL_AMT1
[4] BILL_AMT2 BILL_AMT3 BILL_AMT4
[7] BILL_AMT5 BILL_AMT6 EDUCATION.2
[10] inc_billing_amnt LIMIT_BAL PAY_0
[13] PAY_3 PAY_4 PAY_5
[16] PAY_6 PAY_AMT1 PAY_AMT2
[19] PAY_AMT3 PAY_AMT4 PAY_AMT5
[22] PAY_AMT6 pay_bill_amnt_ratio SEX.2

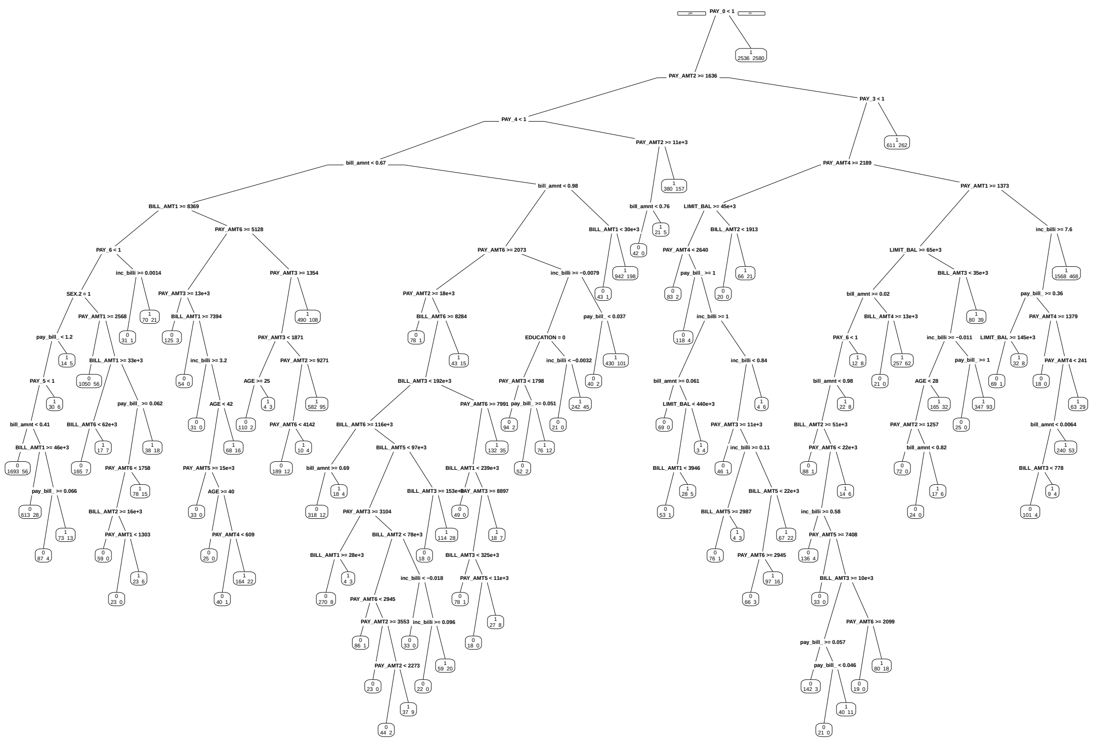
Root node error: 17523/22500 = 0.7788

n= 22500

```
        CP nsplit rel error  xerror     xstd
```

1 0.0128260 0 1.00000 10.0000 0.035529 2 0.0068481 5 0.90675 7.6284 0.041937 3 0.0067911 6 0.89990 7.6104 0.041962 4 0.0036238 7 0.89311 7.5257 0.042063 5 0.0027849 9 0.88586 7.5031 0.042088 6 0.0027107 16 0.86349 7.4156 0.042181 7 0.0024539 18 0.85807 7.3639 0.042230 8 0.0020544 20 0.85316 7.2181 0.042357 9 0.0018832 21 0.85111 7.0891 0.042447 10 0.0018262 23 0.84734 6.9251 0.042531 11 0.0017691 26 0.84147 6.8441 0.042563 12 0.0017120 28 0.83793 6.8160 0.042573 13 0.0016550 31 0.83279 6.8404 0.042563 14 0.0016093 34 0.82783 6.8413 0.042561 15 0.0014838 40 0.81676 6.7827 0.042577 16 0.0014457 42 0.81379 6.7463 0.042586 17 0.0013126 46 0.80797 6.6999 0.042598 18 0.0012840 50 0.80272 6.5649 0.042619 19 0.0012555 52 0.80015 6.5020 0.042624 20 0.0012127 56 0.79513 6.4509 0.042622 21 0.0011984 67 0.77692 6.4628 0.042623 22 0.0011414 73 0.76973 6.4278 0.042621 23 0.0011223 75 0.76745 6.3824 0.042615 24 0.0010843 85 0.75621 6.3781 0.042613 25 0.0010653 89 0.75187 6.3415 0.042608 26 0.0010558 92 0.74867 6.3309 0.042606 27 0.0010272 94 0.74656 6.3190 0.042604 28 0.0010082 98 0.74245 6.3099 0.042603 29 0.0010000 107 0.73252 6.2995 0.042599

```r
# Create an index for of the row with the minimum xerror
index2 <- which.min(tree_loss_matrix$cptable[ , "xerror"])

# Create tree_min
tree_min2 <- tree_loss_matrix$cptable[index2, "CP"]
# Prune the tree using cp = 0.0012788
ptree_loss_matrix <- prune(tree_loss_matrix, cp = tree_min2)

# Use prp() and argument extra = 1 to plot the pruned tree
prp(ptree_loss_matrix, extra = 1)
```



Tree with Weights

Here we are developing Tree model by Assigning different weights to default vs Non Default

```
# Pruning the Tree for the case with Weights
# This vector contains weights of 1 for the non-defaults in the training set,
# and weights of 3 for defaults in the training sets.
# By specifying higher weights for default, the model will assign higher
# importance to classifying defaults correctly.

# Creating a Vector of weights

case_weights <- ifelse(trainSet$default_payment == 1, 3,1 )

head(case_weights)
```

```
[1] 3 3 1 1 1 1
```

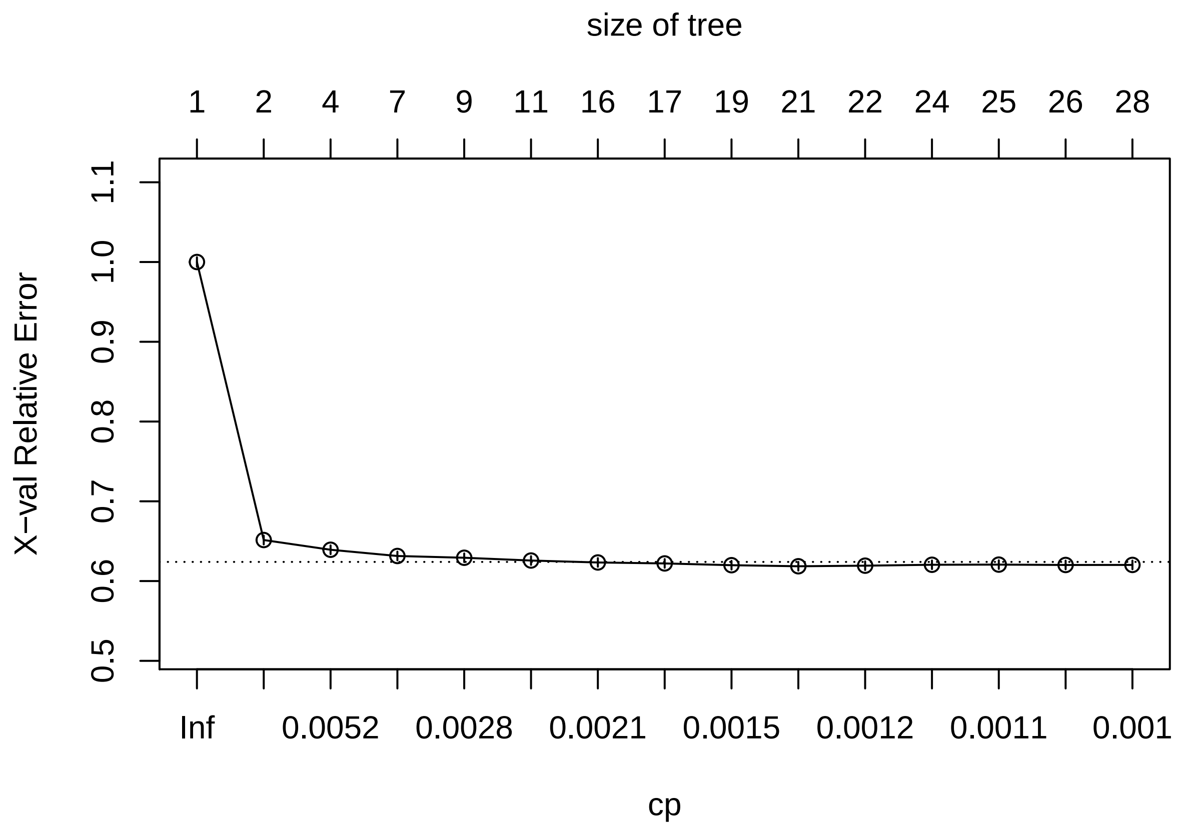
```
str(case_weights)
```

```
num [1:22500] 3 3 1 1 1 1 1 1 1 1 ...
```

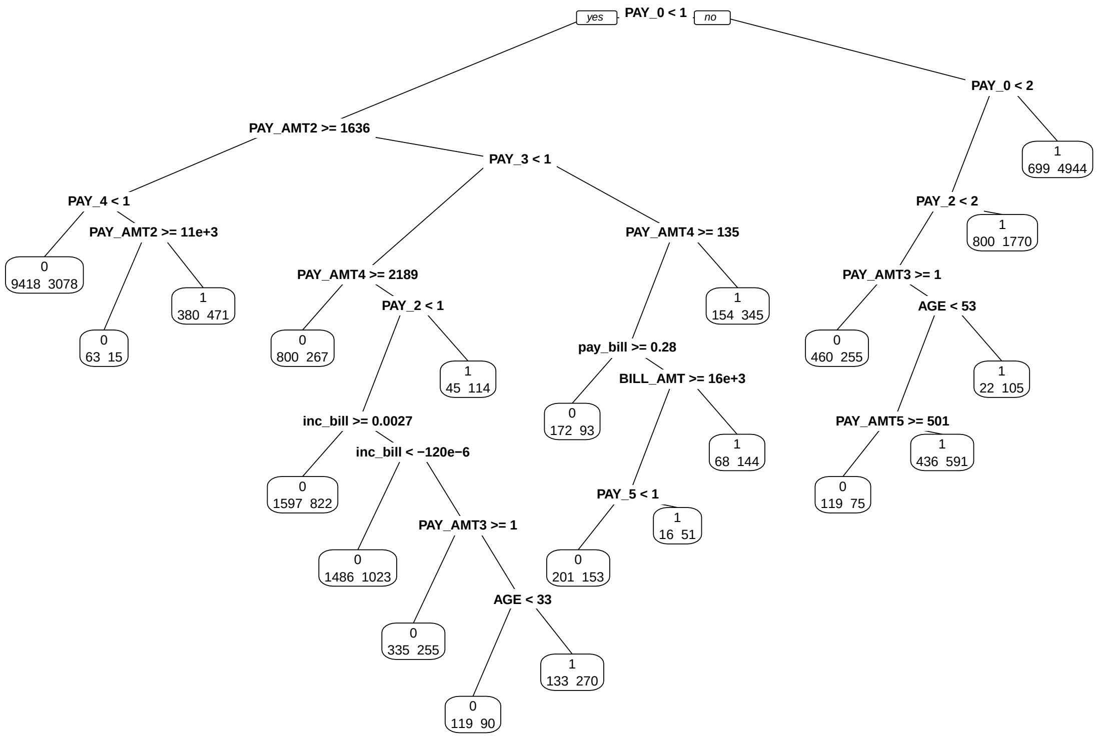
```
head(trainSet$default_payment)
```

[1] 1 1 0 0 0 0 Levels: 0 1

```
set.seed(345)
tree_weights <- rpart(default_payment ~ ., method = "class",
                      data = trainSet, weights = case_weights,
                      control = rpart.control(minsplit = 5, minbucket = 2, cp = 0.001))

# Plot the cross-validated error rate for a changing cp
plotcp(tree_weights)
```



```
# Create an index for of the row with the minimum xerror
index <- which.min(tree_weights$cp[ , "xerror"])

# Create tree_min
tree_min <- tree_weights$cp[index, "CP"]

#  Prune the tree using tree_min
ptree_weights <- prune(tree_weights, tree_min)

# Plot the pruned tree using the rpart.plot()-package
prp(ptree_weights, extra = 1)
```



## Testing of the Different Tree Based Models

Now we would be predicting the probabilities of defaults on the different models by using the Predict Function and by Printing out the Confusion Matrixes

```r
# Now Lets Predict on the test set using the different models that we have developed and
#Compute the Confusion Matrices for the respective models

# Make predictions for each of the pruned trees using the test set.
pred_undersample <- predict(ptree_undersample, newdata = testSet,  type = "class")
pred_prior <- predict(ptree_prior, newdata = testSet,  type = "class")
pred_loss_matrix <- predict(ptree_loss_matrix, newdata = testSet,  type = "class")
pred_weights <- predict(ptree_weights, newdata = testSet,  type = "class")

# Now Lets Create the Confusion Matrices of the Resulting Models and observe the Accuracies
#and the Specificity/Sensitivity Obtained

pred_undersample <- as.factor(pred_undersample)
pred_prior <- as.factor(pred_prior)
pred_loss_matrix <- as.factor(pred_loss_matrix)
pred_weights <- as.factor(pred_weights)




con_matrix_undersample <- confusionMatrix(pred_undersample,testSet$default_payment)
con_matrix_prior <- confusionMatrix(pred_prior,testSet$default_payment)
```

```r
con_matrix_loss_matrix <- confusionMatrix(pred_loss_matrix,testSet$default_payment)
con_matrix_weights<- confusionMatrix(pred_weights,testSet$default_payment)

# Extracting the Accuracy, Specificity and the Sensitivity

ptree_undersample.accuracy <- con_matrix_undersample$overall[["Accuracy"]]
ptree_undersample.specificity <- con_matrix_undersample$byClass[["Specificity"]]
ptree_undersample.sensitivity <- con_matrix_undersample$byClass[["Sensitivity"]]

ptree_prior.accuracy <- con_matrix_prior$overall[["Accuracy"]]
ptree_prior.specificity <- con_matrix_prior$byClass[["Specificity"]]
ptree_prior.sensitivity <- con_matrix_prior$byClass[["Sensitivity"]]

ptree_loss_matrix.accuracy <- con_matrix_loss_matrix$overall[["Accuracy"]]
ptree_loss_matrix.specificity <- con_matrix_loss_matrix$byClass[["Specificity"]]
ptree_loss_matrix.sensitivity <- con_matrix_loss_matrix$byClass[["Sensitivity"]]

ptree_weights.accuracy <- con_matrix_weights$overall[["Accuracy"]]
ptree_weights.specificity <- con_matrix_weights$byClass[["Specificity"]]
ptree_weights.sensitivity <- con_matrix_weights$byClass[["Sensitivity"]]

## All the Accuracies, Specificity and Sensitivity has been stored in
# separate variables since I Intend to compile these and print out a summarized sheet
```

Boosting Algorithms

Unlike many ML models which focus on high quality prediction done by a single model, boosting algorithms seek to improve the prediction power by training a sequence of weak models, each compensating the weaknesses of its predecessors.

boosting is a generic algorithm rather than a specific model. Boosting needs you to specify a weak model (e.g. regression, shallow decision trees, etc) and then improves it.

We would be using two Boosting Algoriths

MARS

Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines, or MARS, is an algorithm for complex non-linear regression problems.

The algorithm involves finding a set of simple linear functions that in aggregate result in the best predictive performance. In this way, MARS is a type of ensemble of simple linear functions and can achieve good performance on challenging regression problems with many input variables and complex non-linear relationships.

AdaBoost

AdaBoost is a specific Boosting algorithm developed for classification problems (also called discrete AdaBoost). The weakness is identified by the weak estimator's error rate.

In each iteration, AdaBoost identifies miss-classified data points, increasing their weights (and decrease the weights of correct points, in a sense) so that the next classifier will pay extra attention to get them right.

Creating the Training Set and the Test Set for the Boosting Algorithms

We would be creating the models for both the undersample data as well as the Full data set

```
# Creating the Training Set and the Test Set
# Here we will be scaling the Data to see the performance of the models with Scaled Data
# We Try and Build These models on the Entire set as well as the Undersampled data set
# and Observe The accuracies and the Specificity/Sensitivity Achieved

# Training the Complete Data set

trainSet1 <- trainSet %>% mutate(default_payment =
                                   factor(ifelse(default_payment == "1", "Yes", "No")))
preProcess_range_model <- preProcess(trainSet1, method='range')
trainSet1 <- predict(preProcess_range_model, newdata = trainSet1)

train_under_sample1 <- train_under_sample %>% mutate(default_payment =
                                    factor(ifelse(default_payment == "1", "Yes", "No")))

preProcess_range_model1 <- preProcess(train_under_sample1, method = 'range')
train_under_sample1 <- predict(preProcess_range_model1, newdata = train_under_sample1)

# Preparing the test Set

testSet1 <- testSet %>% mutate(default_payment =
                          factor(ifelse(default_payment == "1", "Yes", "No")))

# Scaling the Test Set for the Full Model

testSetFull <- predict(preProcess_range_model, newdata = testSet1)
testSet_under_sample1 <- predict(preProcess_range_model1, newdata = testSet1)
```

```
# Developing a MARS Model- Multivariate Adaptive Regression Spline  to
# Predict the defaults on the Full data , in the later part
# we would be developing and Testing  it on the undersample Data
# We would be developing this model using both Tune Length and Tune Grid on the
# Undersampled Training Set
# we would be Observing the Accuracy and the Sensitivity as we as the Specificity to
# understand the performance of the model
# Define the training control


fitControl <- trainControl(
  method = 'cv',                   # k-fold cross validation
  number = 5,                      # number of folds
  savePredictions = 'final',       # saves predictions for optimal tuning parameter
  classProbs = TRUE,                  # should class probabilities be returned
  summaryFunction=twoClassSummary  # results summary function
)
```

```r
# Step 1: Tune hyper parameters by setting tuneLength
set.seed(100)
model_mars1 <- train(default_payment ~ ., data=trainSet1 , method='earth',
                     tuneLength = 5, metric='ROC', trControl = fitControl)
# model_mars1

# Step 2: Predict on testData and Compute the confusion matrix

predictedMars1 <- predict(model_mars1, testSetFull)

con_matrix_Mars <- confusionMatrix(predictedMars1,testSetFull$default_payment)

Mars.accuracy <- con_matrix_Mars$overall[["Accuracy"]]
Mars.specificity <- con_matrix_Mars$byClass[["Specificity"]]
Mars.sensitivity <- con_matrix_Mars$byClass[["Sensitivity"]]

# Doing the predictions Using a Tune Grid

# Step 1: Define the tuneGrid
marsGrid <-  expand.grid(nprune = c(2, 4, 6, 8, 10),
                         degree = c(1, 2, 3))

# Step 2: Tune hyper parameters by setting tuneGrid
set.seed(100)
model_mars2 = train(default_payment ~ ., data=trainSet1 , method='earth',
                    metric='ROC', tuneGrid = marsGrid, trControl = fitControl)
# model_mars2



# Step 3: Predict on testData and Compute the confusion matrix
predictedMars2 <- predict(model_mars2, testSetFull)

con_Matrix_Mars_Tuned <- confusionMatrix(predictedMars2,testSetFull$default_payment)



Mars_Tuned.accuracy <- con_Matrix_Mars_Tuned$overall[["Accuracy"]]
Mars_Tuned.specificity <- con_Matrix_Mars_Tuned$byClass[["Specificity"]]
Mars_Tuned.sensitivity <- con_Matrix_Mars_Tuned$byClass[["Sensitivity"]]
```

Devloping the MARS model on the undersampled data

```r
#  Developing a MARS Model- Multivariate Adaptive Regression Spline  to Predict the defaults
# on the under sampled Data
# We would be developing this model using both Tune Length and Tune Grid on the
# Undersampled Training Set
# we would be Observing the Accuracy and the Sensitivity as we as the Specificity to
# understand the performance of the model
# Define the training control
```

```r
fitControl <- trainControl(
  method = 'cv',                    # k-fold cross validation
  number = 5,                       # number of folds
  savePredictions = 'final',        # saves predictions for optimal tuning parameter
  classProbs = TRUE,                   # should class probabilities be returned
  summaryFunction=twoClassSummary  # results summary function
)


# Step 1: Tune hyper parameters by setting tuneLength
set.seed(100)
model_mars1_under_sample  <- train(default_payment ~ ., data=train_under_sample1 ,
                    method='earth', tuneLength = 5, metric='ROC', trControl = fitControl)
# model_mars1_under_sample

# Step 2: Predict on testData and Compute the confusion matrix

predictedMars1_under_sample <- predict(model_mars1_under_sample, testSet_under_sample1)

con_Matrix_Mars_Under_sample <- confusionMatrix(predictedMars1_under_sample,
                                                testSet_under_sample1$default_payment)





Mars_Under_Sample.accuracy <- con_Matrix_Mars_Under_sample$overall[["Accuracy"]]
Mars_Under_Sample.specificity <- con_Matrix_Mars_Under_sample$byClass[["Specificity"]]
Mars_Under_Sample.sensitivity <-con_Matrix_Mars_Under_sample$byClass[["Sensitivity"]]

# Doing the predictions Using a Tune Grid

# Step 1: Define the tuneGrid
marsGrid <-  expand.grid(nprune = c(2, 4, 6, 8, 10),
                         degree = c(1, 2, 3))

# Step 2: Tune hyper parameters by setting tuneGrid
set.seed(100)
model_mars2_under_sample = train(default_payment ~ ., data=train_under_sample1 ,
                method='earth', metric='ROC', tuneGrid = marsGrid, trControl = fitControl)

# model_mars2_under_sample



# Step 3: Predict on testData and Compute the confusion matrix
predictedMars2_under_sample <- predict(model_mars2_under_sample, testSet_under_sample1)

con_Matrix_Mars_Tuned_Under_sample <-confusionMatrix(predictedMars2_under_sample,
                                                testSet_under_sample1$default_payment)


Mars_Tuned_Under_Sample.accuracy <- con_Matrix_Mars_Tuned_Under_sample$overall[["Accuracy"]]
Mars_Tuned_Under_Sample.specificity <- con_Matrix_Mars_Tuned_Under_sample$byClass[["Specificity"]]
Mars_Tuned_Under_Sample.sensitivity <-con_Matrix_Mars_Tuned_Under_sample$byClass[["Sensitivity"]]
```

Creating the AdaBoost Model

```r
set.seed(100)

# Train the model using adaboost
model_adaboost1 <- train(default_payment ~ ., data=trainSet1, method='adaboost',
                         tuneLength=2, trControl = fitControl)
# model_adaboost1

predicted_adaboost1 <- predict(model_adaboost1, testSetFull)
con_Matrix_adaboost <-confusionMatrix(predicted_adaboost1, testSetFull$default_payment)

Adaboost.accuracy <- con_Matrix_adaboost$overall[["Accuracy"]]
Adaboost.specificity <- con_Matrix_adaboost$byClass[["Specificity"]]
Adaboost.sensitivity <-con_Matrix_adaboost$byClass[["Sensitivity"]]
```

Creating and testng the adaboost Model on the Under Sampled Data

```r
set.seed(100)

# Train the model using adaboost
model_adaboost1_under_sample = train(default_payment ~ ., data=train_under_sample1,
                              method='adaboost', tuneLength=2, trControl = fitControl)

# model_adaboost1_under_sample

predicted_adaboost1_under_sample <- predict(model_adaboost1_under_sample, testSet_under_sample1)
confusionMatrix(predicted_adaboost1_under_sample, testSet_under_sample1$default_payment)
```

Confusion Matrix and Statistics

```
      Reference
```

Prediction No Yes No 4288 586 Yes 1553 1073

```
           Accuracy : 0.7148
             95% CI : (0.7044, 0.725)
No Information Rate : 0.7788
P-Value [Acc > NIR] : 1

              Kappa : 0.3151
```

Mcnemar's Test P-Value : <2e-16

```
         Sensitivity : 0.7341
         Specificity : 0.6468
      Pos Pred Value : 0.8798
      Neg Pred Value : 0.4086
          Prevalence : 0.7788
      Detection Rate : 0.5717
```

Detection Prevalence : 0.6499
Balanced Accuracy : 0.6904

```
   'Positive' Class : No
```

```
con_Matrix_adaboost_under_sample <-confusionMatrix(predicted_adaboost1_under_sample, testSet_under_sampl

Adaboost_under_sample.accuracy <- con_Matrix_adaboost_under_sample$overall[["Accuracy"]]
Adaboost_under_sample.specificity <- con_Matrix_adaboost_under_sample$byClass[["Specificity"]]
Adaboost_under_sample.sensitivity <-con_Matrix_adaboost_under_sample$byClass[["Sensitivity"]]
```

Compiling the Results of All the Models

Comparison of the Performance of the different Models in terms of the Overall Accuracy, Sensitivity and Specificity

```
The_Models <- c('Logistic_Regression_Cut_Off_0.5', 'Logistic_Regression_Cut_Off_0.25',
'Pruned_Tree_Under_Sample','Pruned_Tree_Prior_Probability',
'Pruned_Tree_Loss_Matrix', 'Pruned_Tree_Weights',
'MARS', 'MARS_Tuned',
'MARs_Under_Sample', 'MARS_Under_Sample',
'Adaboost','Adaboost_Under_Sample')

Accuracy_models <- c(log.accuracy.0.5 , log.accuracy.0.25 ,ptree_undersample.accuracy ,
ptree_prior.accuracy , ptree_loss_matrix.accuracy, ptree_weights.accuracy , Mars.accuracy ,
Mars_Tuned.accuracy, Mars_Under_Sample.accuracy, Mars_Tuned_Under_Sample.accuracy,
Adaboost.accuracy,Adaboost_under_sample.accuracy )


Specificity_models <- c(log.specificity.0.5 , log.specificity.0.25 ,ptree_undersample.specificity ,
ptree_prior.specificity , ptree_loss_matrix.specificity, ptree_weights.specificity , Mars.specificity ,
Mars_Tuned.specificity, Mars_Under_Sample.specificity, Mars_Tuned_Under_Sample.specificity,
Adaboost.specificity,Adaboost_under_sample.specificity )


Sensitivity_models <- c(log.sensitivity.0.5 , log.sensitivity.0.25 ,ptree_undersample.sensitivity ,
ptree_prior.sensitivity , ptree_loss_matrix.sensitivity, ptree_weights.sensitivity , Mars.sensitivity ,
Mars_Tuned.sensitivity, Mars_Under_Sample.sensitivity, Mars_Tuned_Under_Sample.sensitivity,
Adaboost.sensitivity,Adaboost_under_sample.sensitivity )

Results <- data.frame(The_Models, Accuracy_models,Sensitivity_models ,Specificity_models)
```

Giving The Summary of all the Modeld developed for easy comparison

```r
kable(Results, col.names =  c("The Models", "Accuracy ", " Sensitivity", " Specificity"),
      align = "cc", caption = "The Summary of All the Models Developed.")
```

Table 1: The Summary of All the Models Developed.

The Models	Accuracy	Sensitivity	Specificity
Logistic_Regression_Cut_Off_0.5	0.8065333	0.9684985	0.2362869
Logistic_Regression_Cut_Off_0.25	0.7556000	0.8144153	0.5485232
Pruned_Tree_Under_Sample	0.7390667	0.7639103	0.6515973
Pruned_Tree_Prior_Probability	0.7854667	0.8572162	0.5328511
Pruned_Tree_Loss_Matrix	0.4829333	0.3660332	0.8945148
Pruned_Tree_Weights	0.7776000	0.8353022	0.5744424
MARS	0.8146667	0.9453861	0.3544304
MARS_Tuned	0.8142667	0.9476117	0.3447860
MARs_Under_Sample	0.7584000	0.7979798	0.6190476
MARS_Under_Sample	0.7636000	0.8108201	0.5973478
Adaboost	0.8041333	0.9250128	0.3785413
Adaboost_Under_Sample	0.7148000	0.7341209	0.6467752

Conclusions

Here in the Models developed the POsitive is considered as Non Defaults Denoted by "0" and Negative as Default denoted by "1".

We are here more Interested in the Defaults which are denoted by 1's

Now in such a scenario as mentioned earlier the Sensitivity Denotes the Power of the Model , i.e the accuracy wrt the Predictions made, and Specificity denotes that out of the existing defaults , how accurately has the model been able to predict the defaults.

From the Banks point of view Specificity is more important here. In Such a Scenario the the Prune Tree Loss matrix and the Pruned Tree Under Sample gives better results Pruned Tree has the Best SPecificity but the accuracy is heavily compromised in order to achieve this.

The Under Sample Models of the Boosting Algorithms MARS and Adaboost giver better Specificity as compared to the Other Models while somewhat maintaining the Accuracies.

This is an unbalanced data set and these type of problems would be faced by the data Scientists designing the ML model to predict the Default.

The Reason for trying out so many model was to check all the options from a basic model to the Boosting algorithms, From Scaling the data to not scaling it and see what are the results whether there exists any significant difference between the output of a basic model to a very sophisticated model. There isnt any drastic difference , the difference is minor.

If the Power of the Model being a criteria where the option is that out of the predictions made , the predictions should be most accurate then MARS seems to be doing a Good Job. But if we want to identify the Maximum of the Defaulters than the models using an undersample data set for training does a decent job.

Here the company might have to define its cost , Of not predicting a would be defaulter correctly and what sort of default rate is sort of acceptable . Based on that Insight the cost component can be Included in the model to give appropriate results which would be useful to the Organization which is the credit card company.

Based on the cost defined an appropriate model can be designed which would increase the relevant metric of interest to the organization , a Trade off might have to be decided upon.