UNIVERSITÀ
DEGLI STUDI DI TRIESTE

MACHINE LEARNING LAB

DEVELOPING A CUSTOMER LEAK DETECTION MODEL USING MACHINE LEARNING TECHNIQUES

CALATAYUD COQUILLAT, Marcos

Year 2019/2020

Index

Introduction and definition of the problem

What is Machine Learning?

Machine Learning is the science of getting computer to learn without being explicitly programmed.

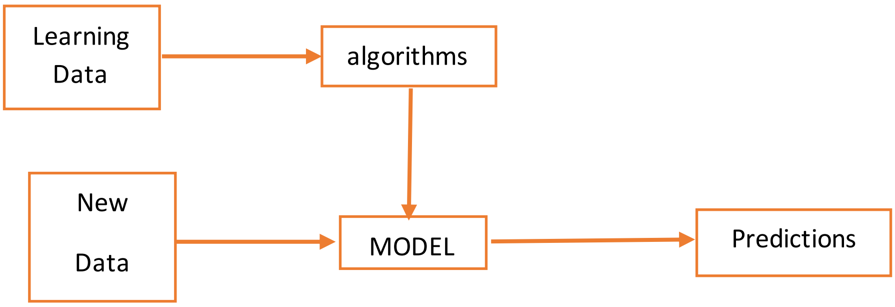
It is about the application of algorithms that can classify and perform groups in order to establish patterns of behaviour, which can help us to predict the future automatically.

One of the reasons of the creation of this new branch of the computer science is the big growth of the computational capacity which allows us to deal with big amounts of data in a cheaper way.

We can distinct between two different types of ML:

Supervised Learning:

Requires the human intervention. There exists an output variable (also called dependent variable) which must be explained with the rest of input variables (or independent variables).



This is the one that we are going to use in this case of study and will see how we can predict which costumers are going abandon a company.

We can also make a distinction between classification and regression problems.

In classification problems the, dependent variable is categorical and the propose of the model is to predict a class for it.

In Regression problems, the dependent variable is a numerical and continuous variable. The prediction should try to minimize the difference between the real and the predicted value.

Unsupervised Learning:

There is not an output variable, all the variables have the same nature.

An example could be the segregation of clients with similar attributes to customize the marketing campaigns. In this case we have all data and the machine makes a distribution of the types of clients depending on their similarities.

Definition of the case

The aim of this project is to apply these techniques to the telecommunications sector. More specifically, we will focus on the study of runaway customers in order to prevent future cases of customer abandonment.

Nowadays, the application of Machine Learning techniques to try to prevent customer leakage, in sectors where a considerable amount of customer data is available, is a widespread reality.

Having said that, there are also many other ML applications in this sector such as:

1. Personalisation of the product. Offering products to customers according to their profile.
2. Cross selling. Detecting possible clients interested in other types of services offered by the company, such as Internet at home, cable TV, etc.
3. Up selling. Detecting customers with needs to increase their payment rate.
4. Dynamic pricing. Offer different prices depending on the type of customer, season of the year etc...
5. Fraud management. Detecting fraudulent customer profiles which are not suitable for the company.

The focus on customer retention is especially interesting for several reasons.

It goes without saying that most of a company's revenue comes from its customers, so customer retention is a key task. That said, the cost of retaining a client is between 5 and 15 times cheaper than the cost of acquiring a new one. In addition, long-lasting and loyal customers often generate better results for companies than more volatile customers.

Knowing the potential customers to leave will allow us to take commercial action against the risk customers in addition to knowing what factors make our customers decide to hire our services in the competition.

The database we're going to work on is as follows:

https://drive.google.com/file/d/1Kwwowe768DbF00gEk--ikHfb97H-WxoV/view?usp=sharing

Each row represents a customer and each column represents a characteristic.

Each row is an observation or data point while each column corresponds to a different independent variable.

In this case we will have 7043 observations composed of 21 characteristics.

These characteristics can be divided into four groups:

1. Churn column: Which tells us if the client has left the company during the last month.
2. Services that the client has contracted.
3. Information on the client's preferences and relative to their seniority.
4. Customer demographic information.

But let’s first list and detail the information that each column offers us.

Customer ID

gender -Whether the customer is a male or a female

SeniorCitizen -Whether the customer is a senior citizen or not (1, 0)

Partner -Whether the customer has a partner or not (Yes, No)

Dependents -Whether the customer has dependents or not (Yes, No)

tenure -Number of months the customer has stayed with the company

PhoneService -Whether the customer has a phone service or not (Yes, No)

MultipleLines -Whether the customer has multiple lines or not (Yes, No, No phone service)

InternetService -Customer’s internet service provider (DSL, Fiber optic, No)

OnlineSecurity -Whether the customer has online security or not (Yes, No, No internet service)

OnlineBackup -Whether the customer has online backup or not (Yes, No, No internet service)

DeviceProtection -Whether the customer has device protection or not (Yes, No, No internet service)

TechSupport -Whether the customer has tech support or not (Yes, No, No internet service)

StreamingTV -Whether the customer has streaming TV or not (Yes, No, No internet service)

StreamingMovies -Whether the customer has streaming movies or not (Yes, No, No internet service)

Contract -The contract term of the customer (Month-to-month, One year, Two year)

PaperlessBilling -Whether the customer has paperless billing or not (Yes, No)

PaymentMethod -The customer's payment method (Electronic check, mailed check, Bank transfer (automatic), Credit card (automatic))

MonthlyCharges -The amount charged to the customer monthly

TotalCharges -The total amount charged to the customer

Churn -Whether the customer churned or not (Yes or No)

Work tool

The programming tool and language will be R, with the help of the IDE (integrated development environment) RStudio. Both are focused on statistical analysis.

https://cran.r-project.org/bin/windows/base/

https://rstudio.com/products/rstudio/

The reason of our choice is that it is an Open Source software, which has many packages and libraries that are easy to use.

Here is a document detailing the use of its main functions:

https://drive.google.com/file/d/15i7iHh5P0tZvD5PO2sMLSVVmA5rY_aGk/view?usp=sharing

Another great advantage of R is that it works with a wide variety of hardware and software (Windows, Linux, Unix ...)

Preparing the dataset

Preparing the data set is one of the most complicated and delicate stages is the development of a quality and useful database for our analysis.

Before we start to process the data, we must make sure that the information that comes to us has been completed based on clear and useful criteria for analysis. For example, in this case we would have to ensure that only voluntary abandonment has been counted as churn. Clients who have cancelled their services for reasons beyond our control, such as the death of a client or migration to another country, have been excluded from the lists.

We must also ensure that the sample we have obtained has the following characteristics:

-Information. The variables selected have a real relationship with the result. Sometimes there is too much information available and much of it is not useful, besides giving rise to heavier computing. A generally useless example may be a user ID, which has no relation to practically any useful variable.

-Representativeness. The sample is large enough that the actual proportions are not affected. Apart from size, there is sometimes a tendency to consider samples that have been collected in ways that exclude certain sectors from the whole as representative. An example could be samples obtained through the Internet, which often exclude older people.

-Precision. Sometimes the data is not precise enough and can end up adding noise to the system which will worsen the results obtained.

Having said that, we can now start with the preparation of the database.

The first step is to load the csv file:

```
telco=read.csv("Telco-Customer-Churn.csv",header = T,na.strings = "?")
```

Next, we assign a factor to binary or tertiary variables that are often defined as text or number but only take 2 or 3 different values in the whole database. This is very useful when applying the algorithms.

```
telco$gender=as.factor(telco$gender)

telco$SeniorCitizen=as.factor(telco$SeniorCitizen)

telco$Partner=as.factor(telco$Partner)

telco$Dependents=as.factor(telco$Dependents)

telco$PhoneService=as.factor(telco$PhoneService)

telco$MultipleLines=as.factor(telco$MultipleLines)

telco$InternetService=as.factor(telco$InternetService)

telco$OnlineSecurity=as.factor(telco$OnlineSecurity)

telco$OnlineBackup=as.factor(telco$OnlineBackup)

telco$DeviceProtection=as.factor(telco$DeviceProtection)
```

```
telco$TechSupport=as.factor(telco$TechSupport)

telco$StreamingTV=as.factor(telco$StreamingTV)

telco$StreamingMovies=as.factor(telco$StreamingMovies)

telco$Contract=as.factor(telco$Contract)

telco$PaperlessBilling=as.factor(telco$PaperlessBilling)

telco$PaymentMethod=as.factor(telco$PaymentMethod)

telco$Churn=as.factor(telco$Churn)
```

Another thing we need to do is to eliminate observations that are incomplete, those containing NAs.

To check if they exist and where they are, we use the following function.

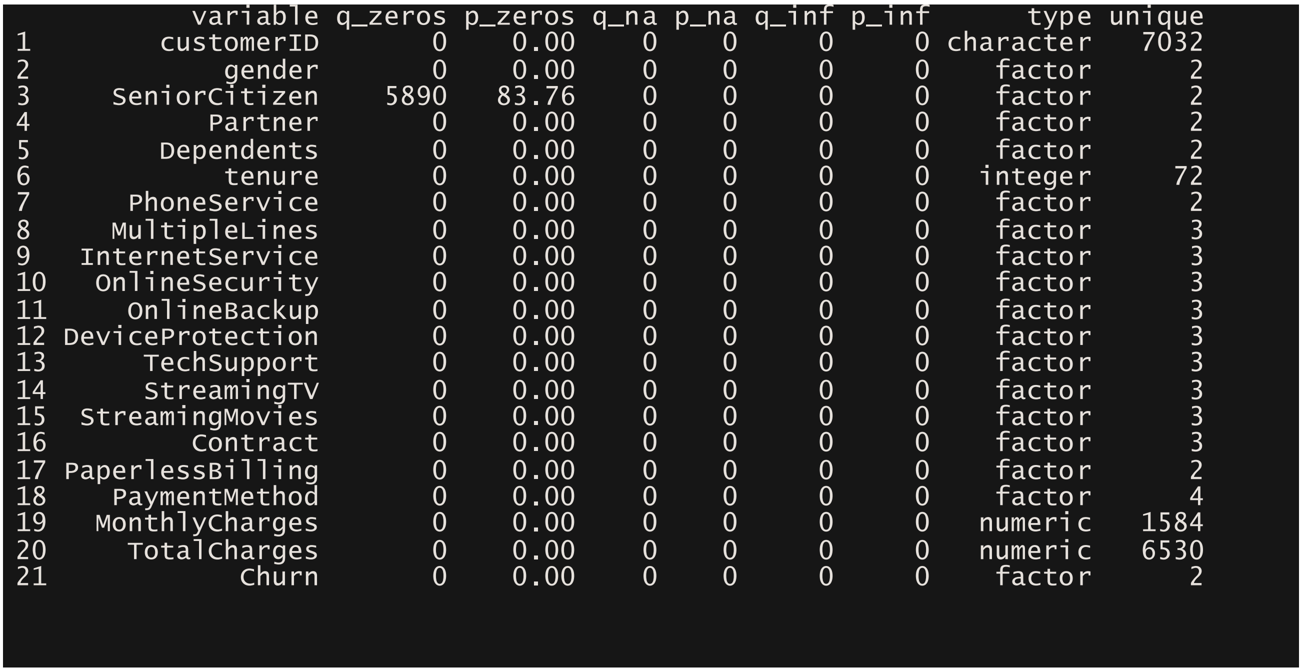
```
sapply(telco, function(x) sum(is.na(x)))
```

We note that there are 11 observations that have an empty TotalCharges column, so we proceed to remove them.

```
telco=na.omit(telco)
```

Now we visualize how the database finally looks. For this, we use a function of the FunModeling library:

```
library(funModeling)

df_status(telco)
```



```
           variable q_zeros p_zeros q_na p_na q_inf p_inf      type unique
1        customerID       0    0.00    0    0     0     0 character   7032
2            gender       0    0.00    0    0     0     0    factor      2
3     SeniorCitizen    5890   83.76    0    0     0     0    factor      2
4           Partner       0    0.00    0    0     0     0    factor      2
5        Dependents       0    0.00    0    0     0     0    factor      2
6            tenure       0    0.00    0    0     0     0   integer     72
7      PhoneService       0    0.00    0    0     0     0    factor      2
8     MultipleLines       0    0.00    0    0     0     0    factor      3
9   InternetService       0    0.00    0    0     0     0    factor      3
10   OnlineSecurity       0    0.00    0    0     0     0    factor      3
11     OnlineBackup       0    0.00    0    0     0     0    factor      3
12 DeviceProtection       0    0.00    0    0     0     0    factor      3
13      TechSupport       0    0.00    0    0     0     0    factor      3
14      StreamingTV       0    0.00    0    0     0     0    factor      3
15  StreamingMovies       0    0.00    0    0     0     0    factor      3
16         Contract       0    0.00    0    0     0     0    factor      3
17 PaperlessBilling       0    0.00    0    0     0     0    factor      2
18    PaymentMethod       0    0.00    0    0     0     0    factor      4
19   MonthlyCharges       0    0.00    0    0     0     0   numeric   1584
20     TotalCharges       0    0.00    0    0     0     0   numeric   6530
21            Churn       0    0.00    0    0     0     0    factor      2
```

The "unique" column tells us how many different values the variable takes.

Now we will discard the variables that have no influence on the dependent variable.

To do this, we will use the Boruta algorithm.

Before being able to use a package in R we must proceed to download it (Tools>Install Packages) and then call the library.

The algorithm works as follows:

1. First, randomness is added to the data set by creating shuffled copies of the variables, which are called shadow variables.

2. Then a random slot is trained with the extended data set and a mean of the importance of the variables is obtained (the default measure is Mean Decrease Accuracy).

3. In each iteration, it is checked whether the real variable is more important than the shadow variable, and the variables that are considered unimportant are eliminated.

4. The algorithm will stop when all the variables have been accepted or rejected.

```
library(Boruta)

library(ranger)
```

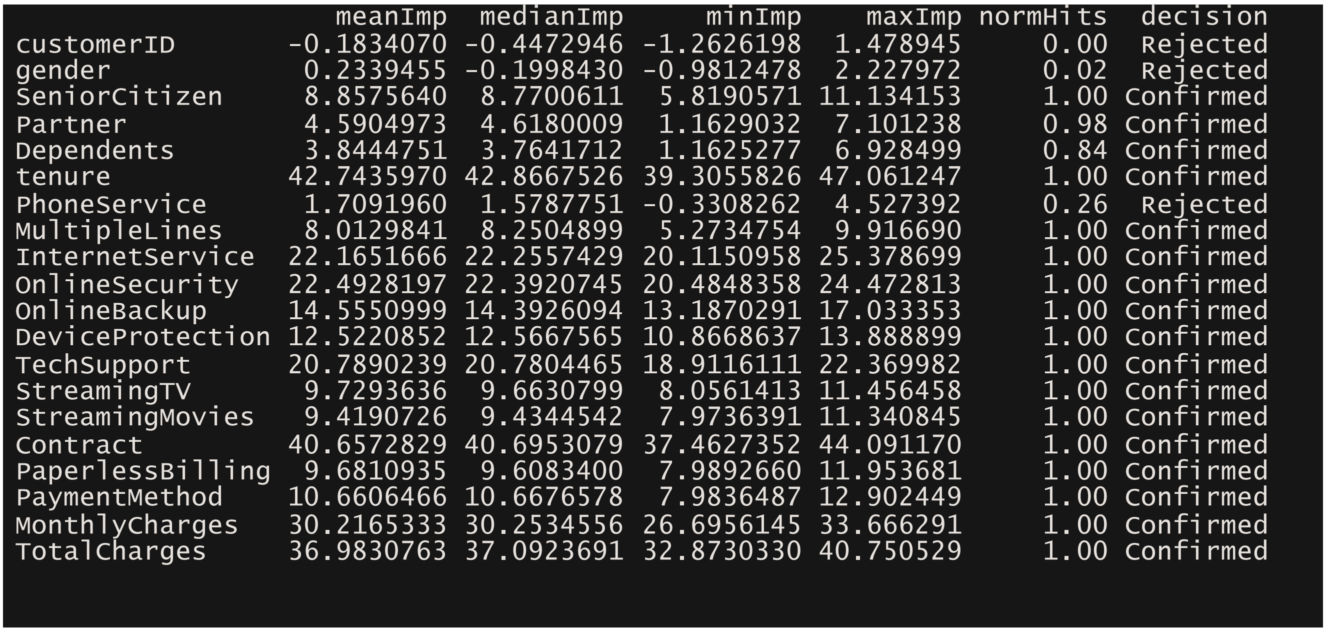
We call the function indicating the target variable and the dataset to which we apply it.

```
boruta=Boruta(Churn~., data = telco, doTrace = 2)

print(boruta)
```

The algorithm rejects 3 variables: "customer ID", "gender" and "PhoneService"

If we want to obtain more information about the importance of the variables, we can call the attStats() function, which will show us different indicators of the importance of the variables such as the mean, median, minimum and maximum.

```
borutadf=attStats(boruta)

print(borutadf)
```

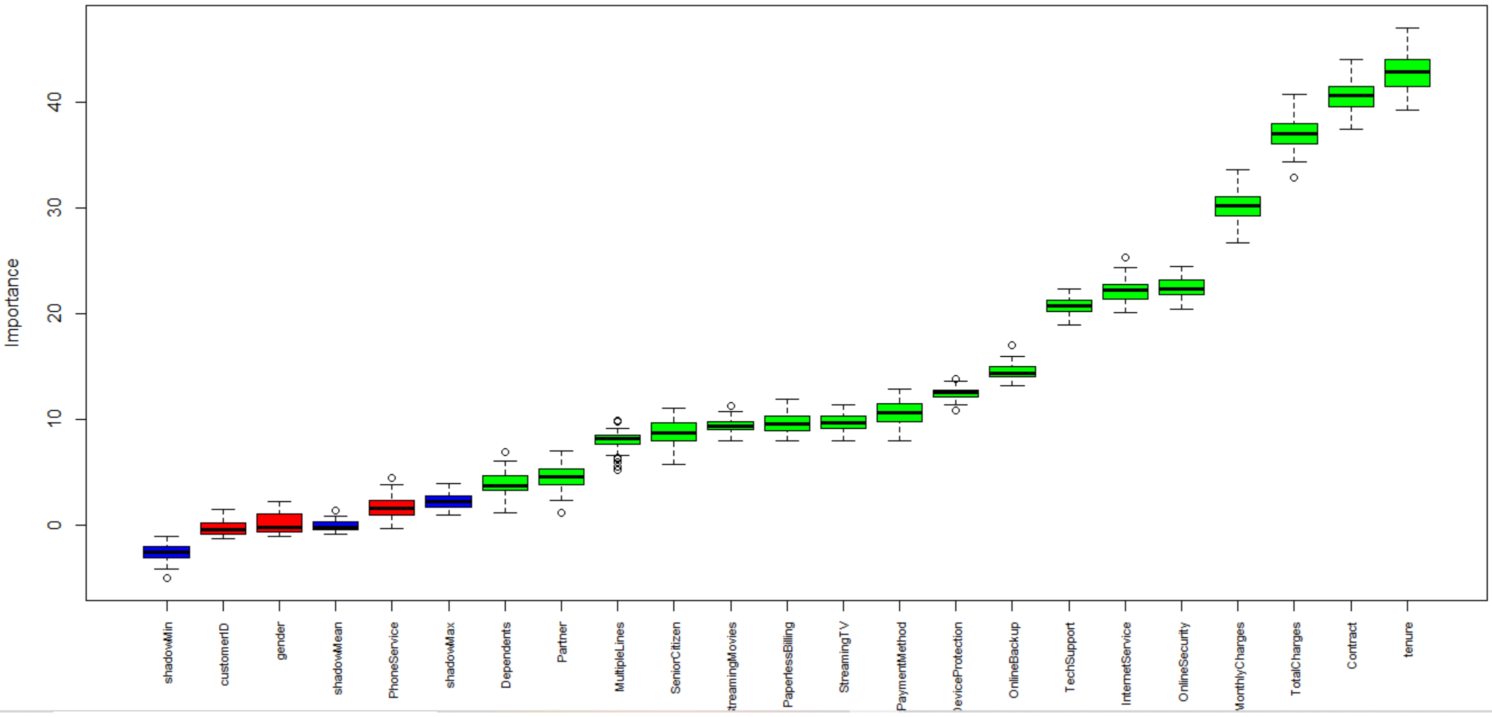


```
                    meanImp  medianImp     minImp    maxImp normHits  decision
customerID       -0.1834070 -0.4472946 -1.2626198  1.478945     0.00  Rejected
gender            0.2339455 -0.1998430 -0.9812478  2.227972     0.02  Rejected
SeniorCitizen     8.8575640  8.7700611  5.8190571 11.134153     1.00 Confirmed
Partner           4.5904973  4.6180009  1.1629032  7.101238     0.98 Confirmed
Dependents        3.8444751  3.7641712  1.1625277  6.928499     0.84 Confirmed
tenure           42.7435970 42.8667526 39.3055826 47.061247     1.00 Confirmed
PhoneService      1.7091960  1.5787751 -0.3308262  4.527392     0.26  Rejected
MultipleLines     8.0129841  8.2504899  5.2734754  9.916690     1.00 Confirmed
InternetService  22.1651666 22.2557429 20.1150958 25.378699     1.00 Confirmed
OnlineSecurity   22.4928197 22.3920745 20.4848358 24.472813     1.00 Confirmed
OnlineBackup     14.5550999 14.3926094 13.1870291 17.033353     1.00 Confirmed
DeviceProtection 12.5220852 12.5667565 10.8668637 13.888899     1.00 Confirmed
TechSupport      20.7890239 20.7804465 18.9116111 22.369982     1.00 Confirmed
StreamingTV       9.7293636  9.6630799  8.0561413 11.456458     1.00 Confirmed
StreamingMovies   9.4190726  9.4344542  7.9736391 11.340845     1.00 Confirmed
Contract         40.6572829 40.6953079 37.4627352 44.091170     1.00 Confirmed
PaperlessBilling  9.6810935  9.6083400  7.9892660 11.953681     1.00 Confirmed
PaymentMethod    10.6606466 10.6676578  7.9836487 12.902449     1.00 Confirmed
MonthlyCharges   30.2165333 30.2534556 26.6956145 33.666291     1.00 Confirmed
TotalCharges     36.9830763 37.0923691 32.8730330 40.750529     1.00 Confirmed
```

As the three variables that we have mentioned before have been rejected, the algorithm also allows us to have an idea of which variables are more important. We observe that there are two variables that are especially relevant, and they are "tenure" and "Contract".

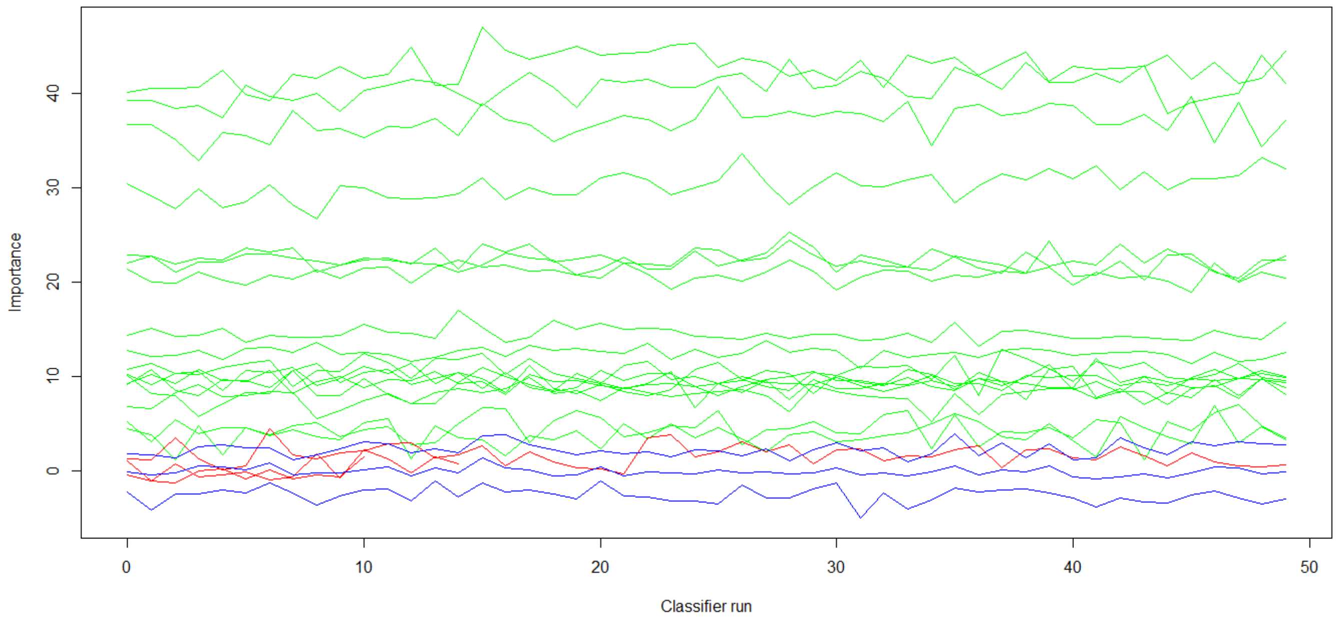
These results can also be displayed graphically by executing the following code:

```
plot(boruta, xlab = "", xaxt = "n")

lz=lapply(1:ncol(boruta$ImpHistory),function(i)

boruta$ImpHistory[is.finite(boruta$ImpHistory[,i]),i])

names(lz)=colnames(boruta$ImpHistory)

Labels=sort(sapply(lz,median))

axis(side = 1,las=2,labels = names(Labels),at = 1:ncol(boruta$ImpHistory), cex.axis = 0.7)
```



In green the important variables appear, in red the rejected ones and in blue, the called shadows.

We can also know the evolution of the importance of the variables as the algorithm has been advancing in the executions. We observe that the importance of each of the variables has behaved with relative stability, which reinforces the confidence in the veracity of the results obtained.



```
plotImpHistory(boruta)
```

Finally, we generate a new vector where we store the variables that we have catalogued as important and which we will use from now on.

```
imp.var=getSelectedAttributes(boruta)
```

Exploratory analysis of the data

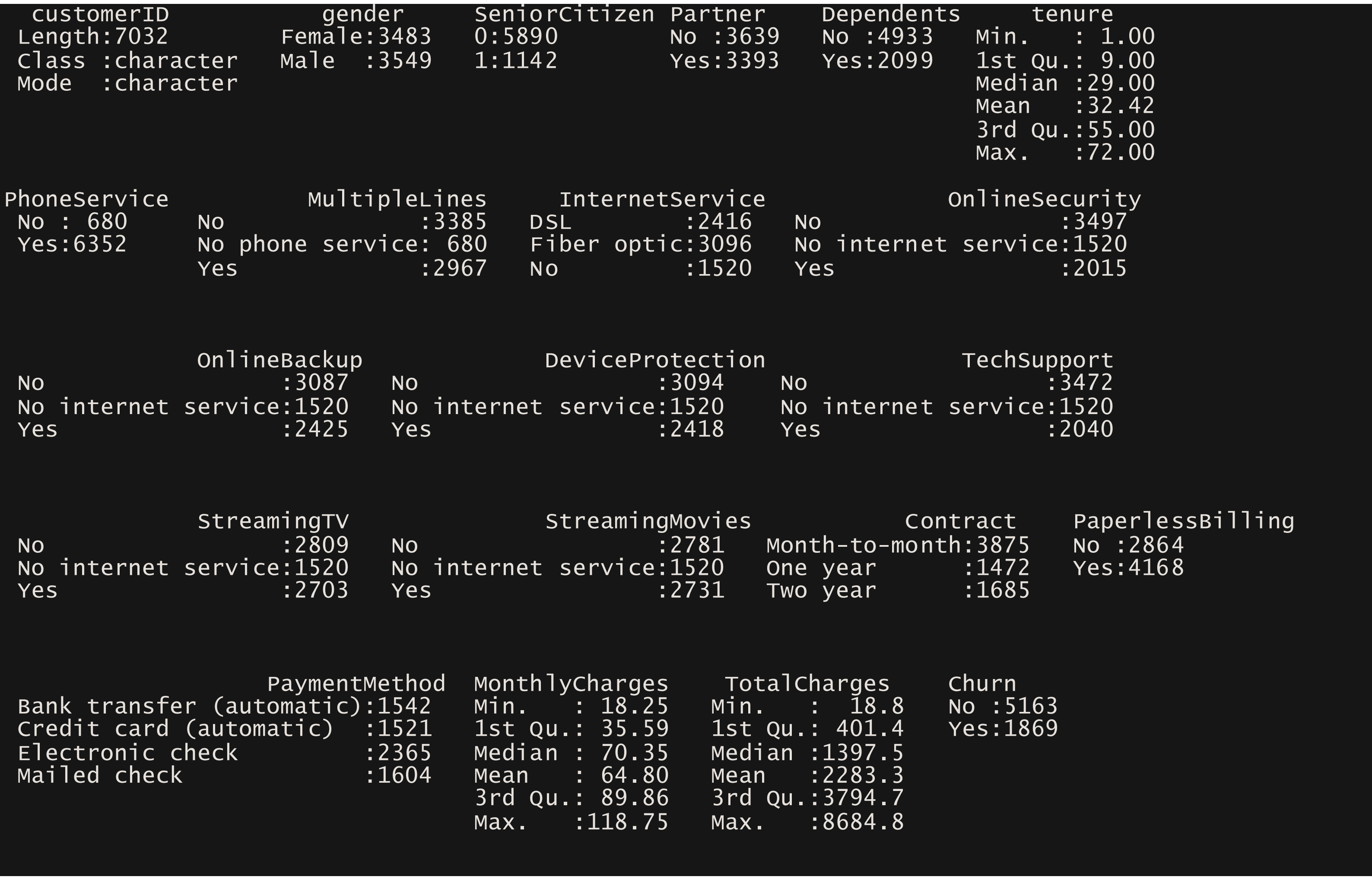
Before applying algorithms, it is recommended to have a general idea of the data that are going to be analysed. This will allow us to make a more relevant analysis once the ML techniques have been applied, since we will have a better knowledge of the clients. The data always have a sense and we should not reduce the problem to a simple computational analysis.

Another great objective of this section is to introduce the graphic visualization tools that can be of great help to us.

To do this, there are several functions from different packages.

The first is the basic summary function, which comes by default in R and gives us general data on each of the variables.

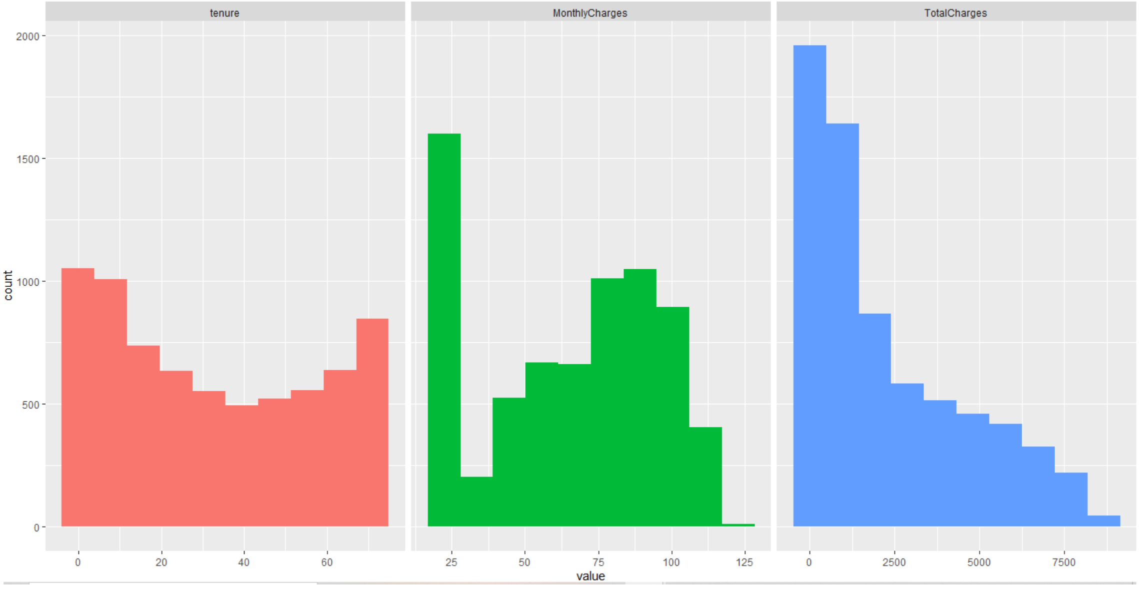
```
summary(telco)
```



```
  customerID            gender      SeniorCitizen Partner    Dependents     tenure     
 Length:7032        Female:3483   0:5890        No :3639   No :4933   Min.   : 1.00  
 Class :character   Male  :3549   1:1142        Yes:3393   Yes:2099   1st Qu.: 9.00  
 Mode  :character                                                     Median :29.00  
                                                                      Mean   :32.42  
                                                                      3rd Qu.:55.00  
                                                                      Max.   :72.00  

PhoneService          MultipleLines     InternetService             OnlineSecurity
 No : 680     No              :3385   DSL        :2416   No                 :3497  
 Yes:6352     No phone service: 680   Fiber optic:3096   No internet service:1520  
              Yes             :2967   No         :1520   Yes                :2015  


              OnlineBackup              DeviceProtection              TechSupport  
 No                 :3087   No                 :3094   No                 :3472  
 No internet service:1520   No internet service:1520   No internet service:1520  
 Yes                :2425   Yes                :2418   Yes                :2040  


              StreamingTV              StreamingMovies           Contract    PaperlessBilling
 No                 :2809   No                 :2781   Month-to-month:3875   No :2864        
 No internet service:1520   No internet service:1520   One year      :1472   Yes:4168        
 Yes                :2703   Yes                :2731   Two year      :1685                   


                   PaymentMethod  MonthlyCharges    TotalCharges     Churn     
 Bank transfer (automatic):1542   Min.   : 18.25   Min.   :  18.8   No :5163  
 Credit card (automatic)  :1521   1st Qu.: 35.59   1st Qu.: 401.4   Yes:1869  
 Electronic check         :2365   Median : 70.35   Median :1397.5             
 Mailed check             :1604   Mean   : 64.80   Mean   :2283.3             
                                  3rd Qu.: 89.86   3rd Qu.:3794.7             
                                  Max.   :118.75   Max.   :8684.8             
```

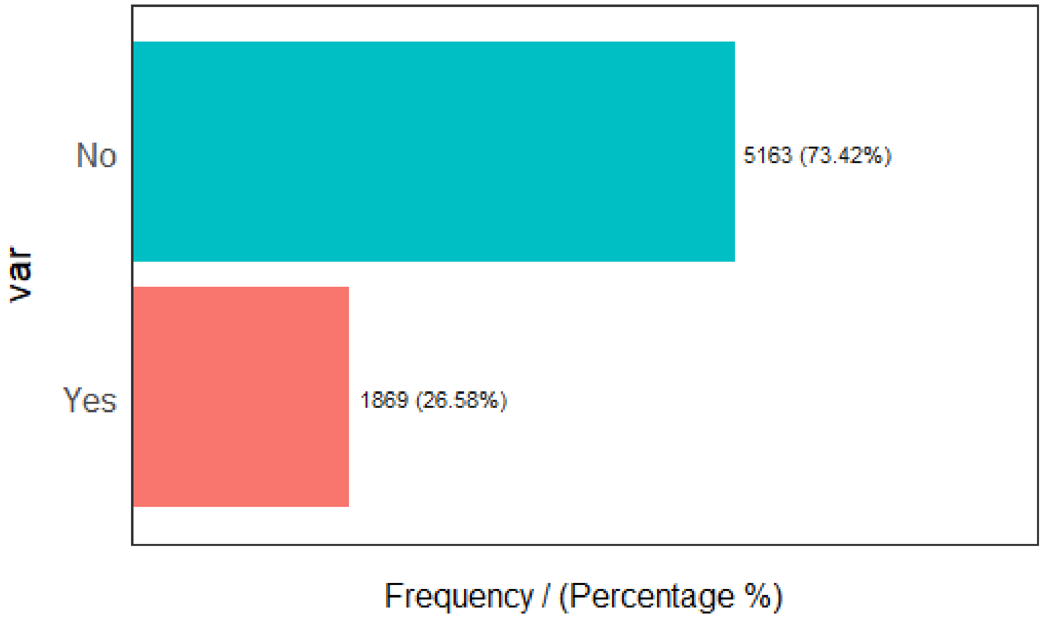
Something also quite useful is to be able to have available a graphic representation of the distribution of the numerical variables, since it is quite more intuitive than what the previous function gave us.

```
plot_num(telco)
```



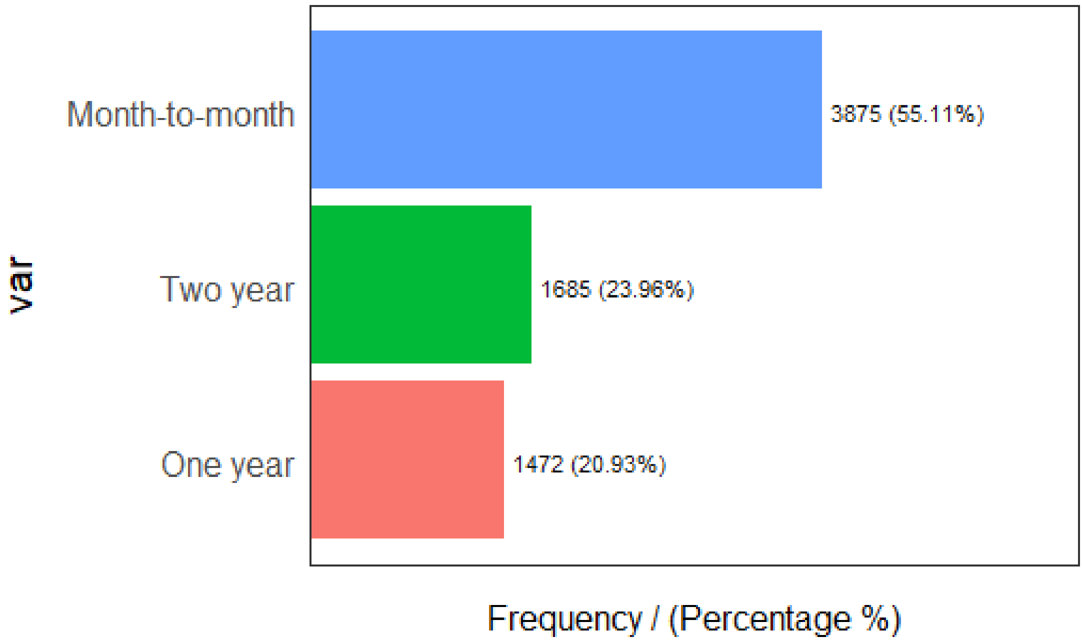
From the factor variables we can also obtain a graphic representation. In this case we represent only a few, but it should not be wrong to do it with all of them.

```
freq(telco$Churn)
```

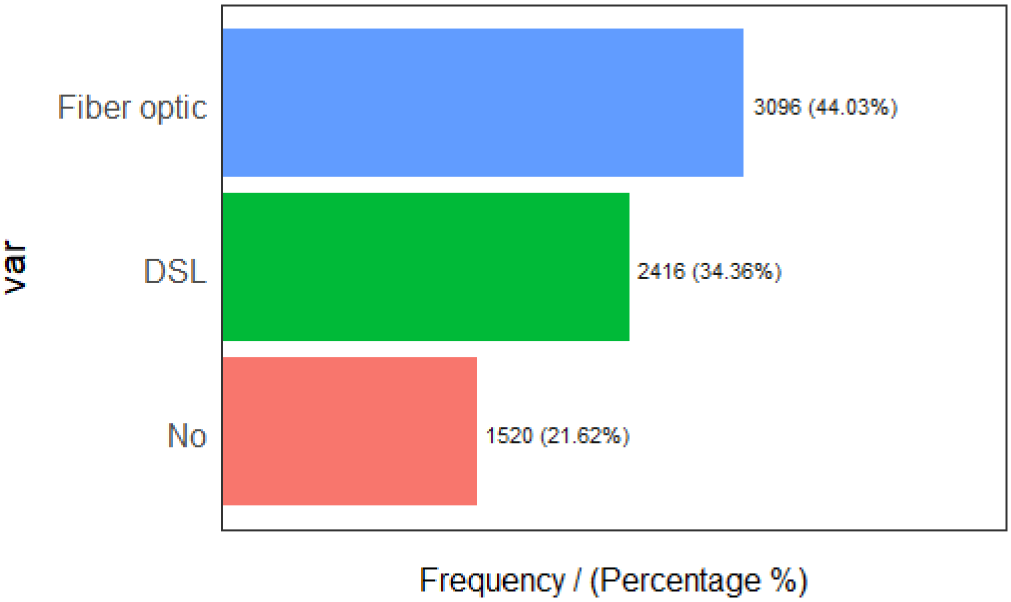


We note that 26.58% of the customers we had in the last year have left the company. This is not a negligible percentage and it should be solved.

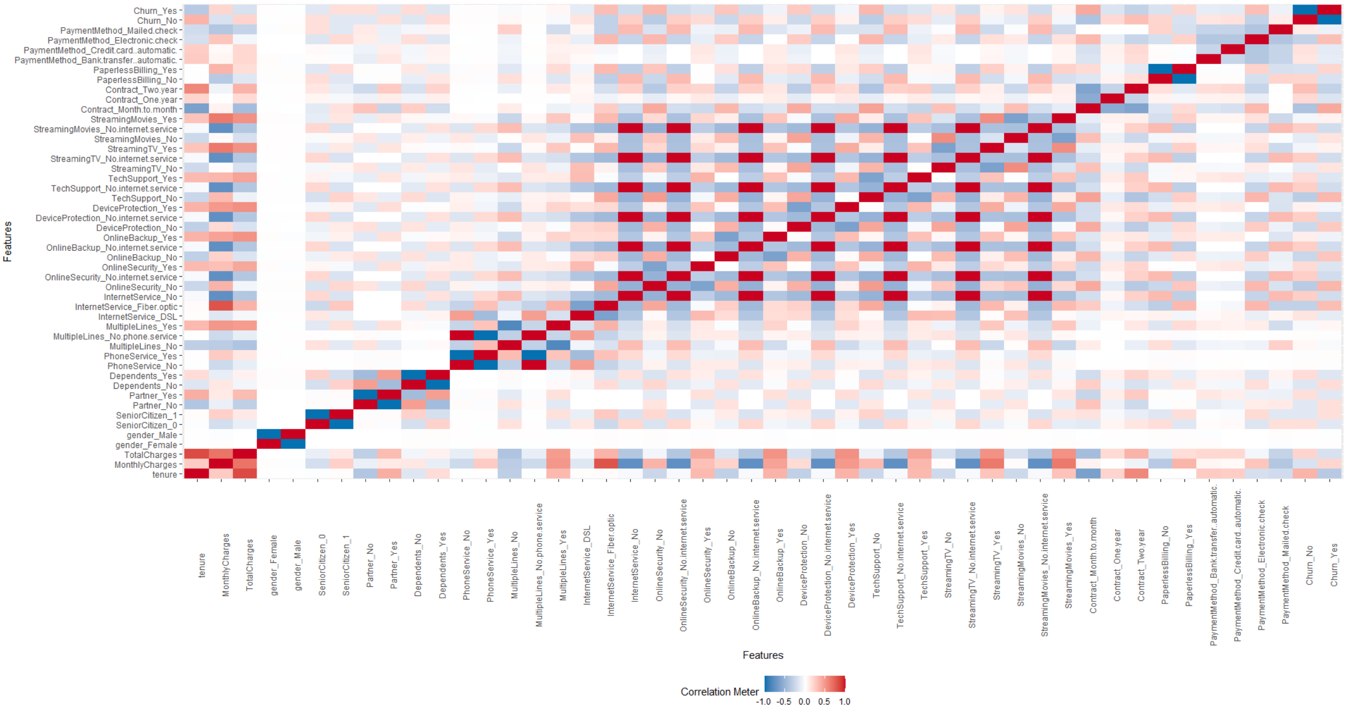
```
freq(telco$Contract)
```



```
freq(telco$InternetService)
```



Another very useful indicator is correlation. The plot_correlation() function gives us a heat map that indicates the intensity of the correlation. Positive correlations are shown in red, while negative correlations are shown in blue.



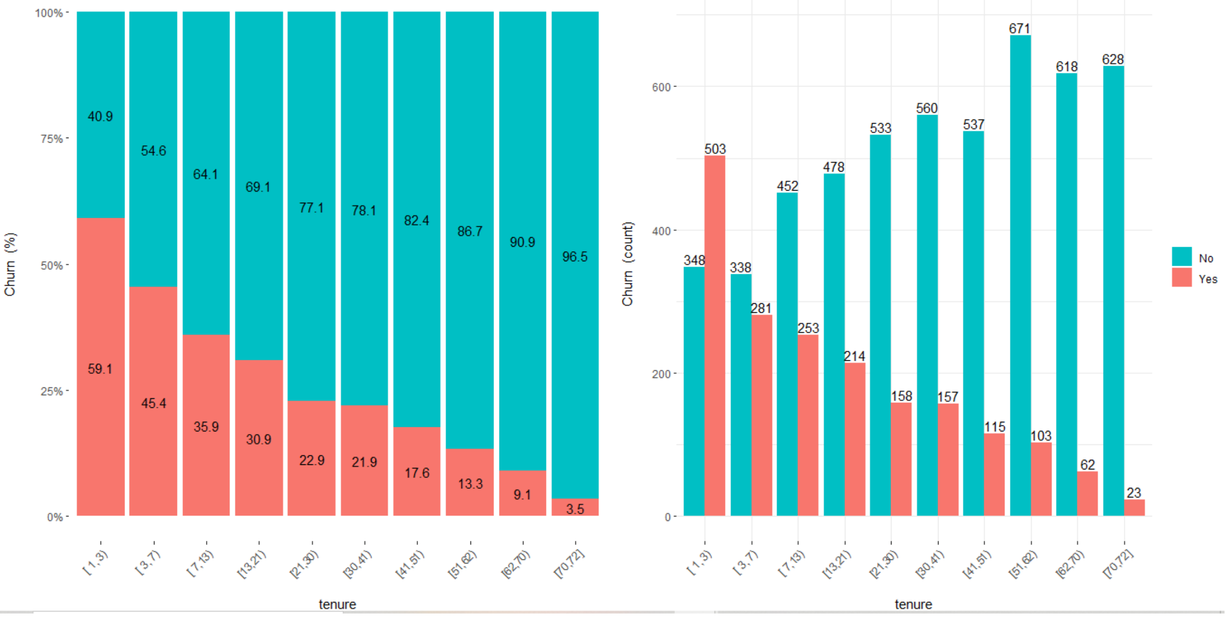
The graph shows us that some of the variables have an exact and positive correlation. This is because redundant information is available. An example could be the high correlation between the StreamingMovies variable and InternetService Fiber.optic as one implies the other.

If we look at the churn column, we see that there are results like those obtained with the boruta algorithm.

Another function that is quite useful is the cross_plot() function, which allows us to make a more thorough analysis of the relationship between two variables.

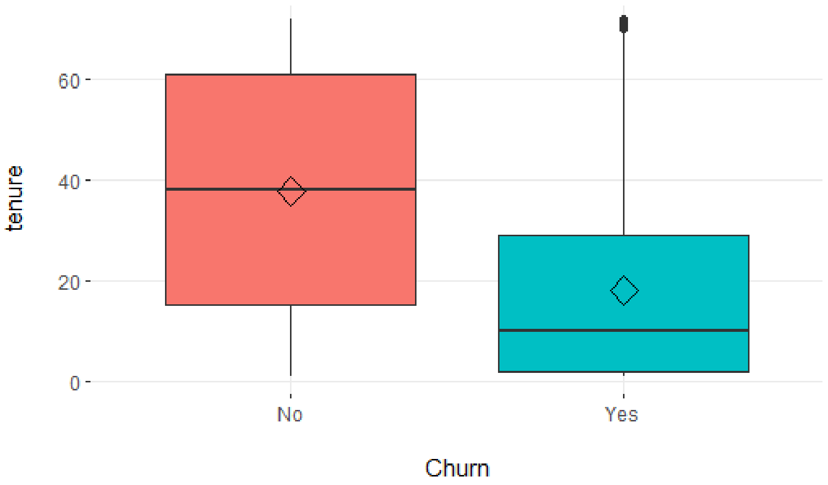
With this function we can check the relationship of the dependent variable with variables such as tenure and Contract.

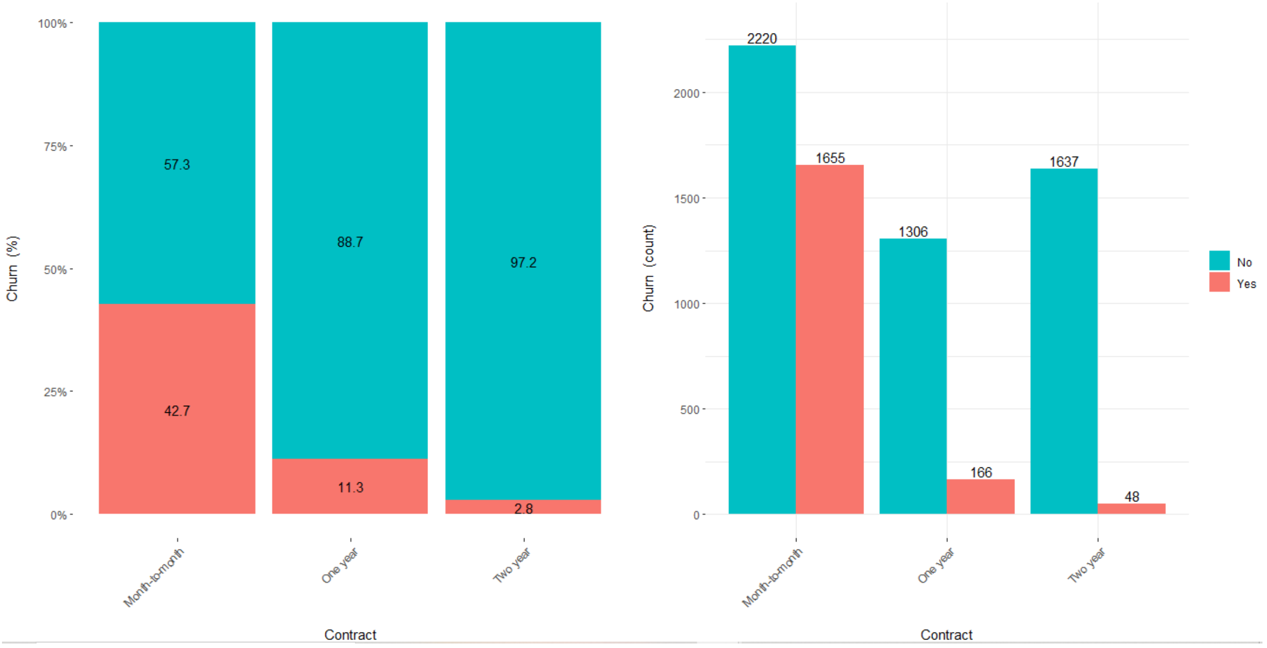
```
cross_plot(data = telco, input = c("tenure","Contract"), target = "Churn")
```



If we prefer, we can also use the box and moustache format

```
plotar(data = telco, input = c("tenure"), target = "Churn", plot_type = "boxplot")
```





In both cases we see an obvious relationship, which is otherwise quite logical. The longer a customer has been with the company the less likely he is going to leave. Depending on the type of contract something similar happens, customers with longer contracts are loyal to the company.

With this small approach we conclude the exploratory analysis.

Classification using ML algorithms

A necessary step before starting to apply the algorithms is to make the distinction between learning data and test data. The way to proceed will be to find a model with the training data and to be able to validate it with the test data.

In this case we are going to make an 80/20 separation, this separation will be done in a random way. There is a way to control the randomness and it is by establishing a random seed. To do this we will use the function set.seed(). This will allow us to make an identical distribution if we plant the same seed. As we will see later, it will also allow us to do an iteration with different training/test data by simply changing this seed.

```
set.seed(1)

tr=sample(1:nrow(telco), round(nrow(telco)*0.8))

train.telco=telco[tr,]
```

```
test.telco=telco[-tr,]
```

We will now move on to remove the columns that were rejected by the boruta algorithm to make the computational process more agile and reliable. We have also added the churn column that was not contemplated and finally we removed the auxiliary variables train.telco and test.telco.

```
train=train.telco[,imp.var]

test=test.telco[,imp.var]

train$churn=train.telco$Churn

test$churn=test.telco$Churn

rm(test.telco,train.telco)
```

Naïve Bayes Classifier

We will start with a model, which despite being quite simple (and assuming that the variables are independent of each other), usually provides quite good results.

To do this we will use functions found in the e1071 library

```
library(e1071)

m01=naiveBayes(churn~., data = train)

m01pred=predict(m01, type = "raw", test)[,2]

m01pred_F=as.factor(round(m01pred))

levels(m01pred_F)=c("No","Yes")
```

Confusion matrix

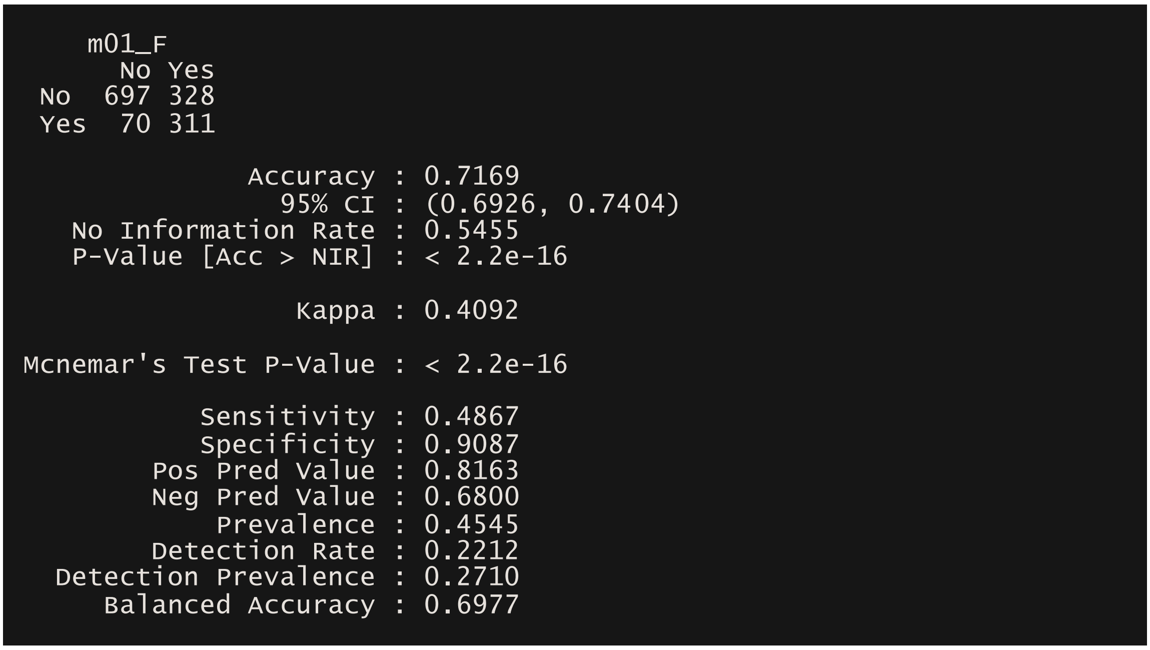
The confusion matrix is a fundamental tool in classification problems, and even more crucial when dealing with unbalanced data. Therefore, indicators other than accuracy are sought.

Let's take an example. There is a disease that only affects 1% of the population, and for this reason we created a model that tries to predict whether an individual will have it or not. The model is bad and whatever the patient is, it tells us that the patient is not going to have the disease and it is right 99% of the time because only 1% will have the disease. It would be wrong to say that this is a good model simply by looking at the success rate.

Most algorithms calculate the accuracy of models based on the percentage of correctly classified observations. The confusion matrix also reports the number of incorrectly classified instances and in what class they occurred. If the importance of the classes is different, as is the case in fraud detection, knowing where the classification errors are occurring will allow decisions to be made that are more appropriate to the solution of the problem.

One way to obtain a large number of these indicators is with the functionconfusionMatrix() integrated in the Caret library.

```
library(caret)

cm01=confusionMatrix(table(test$churn,m01pred_F),positive = "Yes")
```



```
     m01_F
       No  Yes
  No   697 328
  Yes   70 311

               Accuracy : 0.7169
                 95% CI : (0.6926, 0.7404)
    No Information Rate : 0.5455
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4092

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.4867
            Specificity : 0.9087
         Pos Pred Value : 0.8163
         Neg Pred Value : 0.6800
             Prevalence : 0.4545
         Detection Rate : 0.2212
   Detection Prevalence : 0.2710
      Balanced Accuracy : 0.6977
```

The meaning of these indicators is detailed below:

		Prediction	
		Negative	Positive
Real	Negative	True negative (TN)	False positive (FP)
	Positive	False negative (FN)	True positive (TP)

Other measures can be extracted from the confusion matrix.

- Accuracy (Accuracy): (TP + TN) / (TP + TN + FP + FN)

- Error rate: 1- Accuracy = (FP + FN) / (TP + TN + FP + FN)

- No Information Rate: (TP + FP) / (TP + TN + FP + FN) In two-by-two confounding matrices it is equal to the Prevalence.

- Sensitivity: TP / (TP + FP)

- Specificity: TN / (TN + FN)

- True Positive Rate (Pos. Pred. Value): TP / (TP + FN) Percentage of positive instances correctly classified.

- True negative rate (Neg. Pred. Value): TN / (TN + FP). Percentage of negative instances correctly classified.

- False Positive Rate: FP/ (FP + TN) Percentage of incorrectly classified negative instances.

- False Negative Rate: FN / (TP + FN) Percentage of incorrectly classified positive instances.

The sum of the True Positive Rate and False Negative Rate will result in 1. Similarly, the True Negative Rate and False Positive Rate, the addition of them should be equal to one.

- Detection Rate: TP / (TP + TN + FP + FN)

Cohen's Kappa statistic is an index that compares the overall accuracy of the model with the accuracy that would be obtained if the model were to randomly rank the instances. Kappa is defined as the difference between the overall precision and the expected precision divided by 1 minus the expected precision.

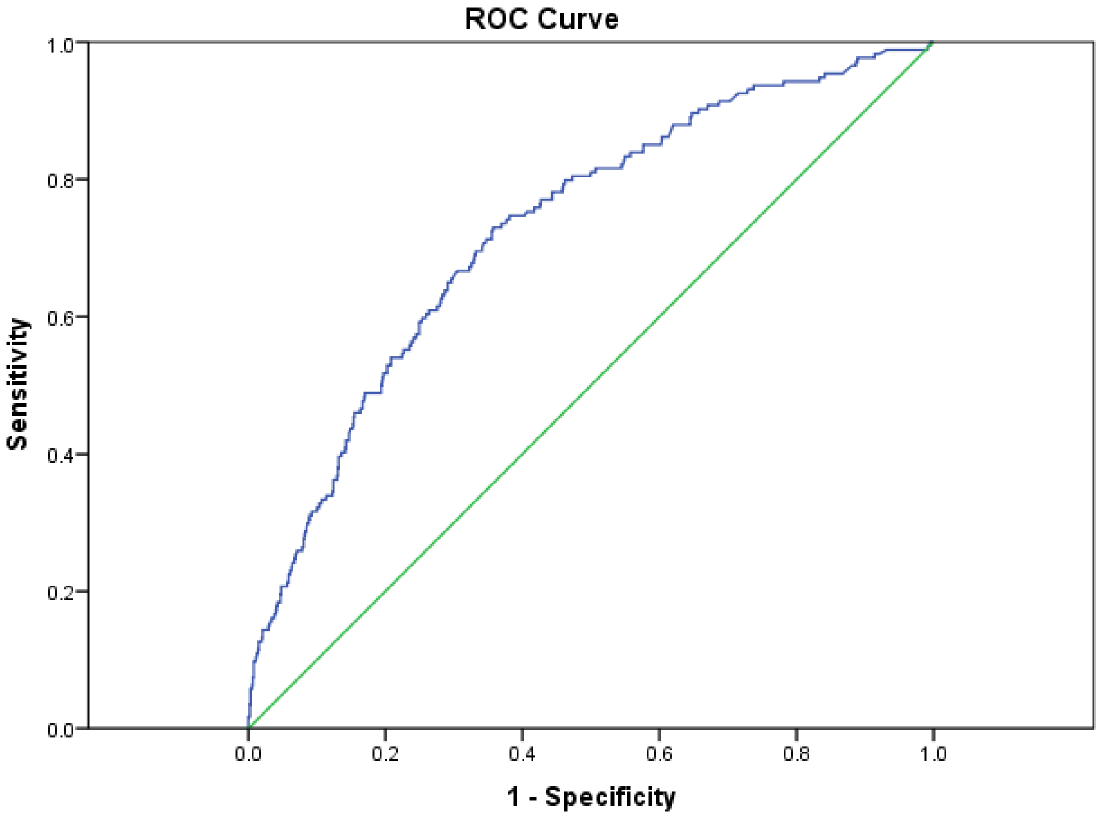
A value of +1 would indicate total match (the ideal value), values of 0 indicate that the match is the same as can be expected by chance and values of -1 would express total disagreement.

It can be obtained by using the following formula:

$$Kappa = \frac{Accuracy - 0{,}5}{1 - 0{,}5}$$

Area under the ROC curve.

Another measure that gives us an idea of the quality of a sorter is the Area under the curve AUC. To obtain this measure, simply draw the ROC curve (Receiver Operating Characteristic) and compute its area under the curve.



Diagonal segments are produced by ties.

To understand this curve, it is necessary to understand the concept of threshold.

Generally, the most used techniques offer us a model that returns a probabilistic output. That is, it offers us a probability and the classification is made in function of that given probability. In our case, the models return a probability of abandonment of the client. If that probability is higher than 0.5, we say that the client will leave the company and if not, that the client will not. This value of 0.5 is what we call threshold and can be varied according to our interests. As we vary it, the classification will be different, and the measures of TP rate and FP rate will also vary. The roc curve is not more than a representation of the TP rate versus the FP rate for all the values of the threshold between 0 and 1.

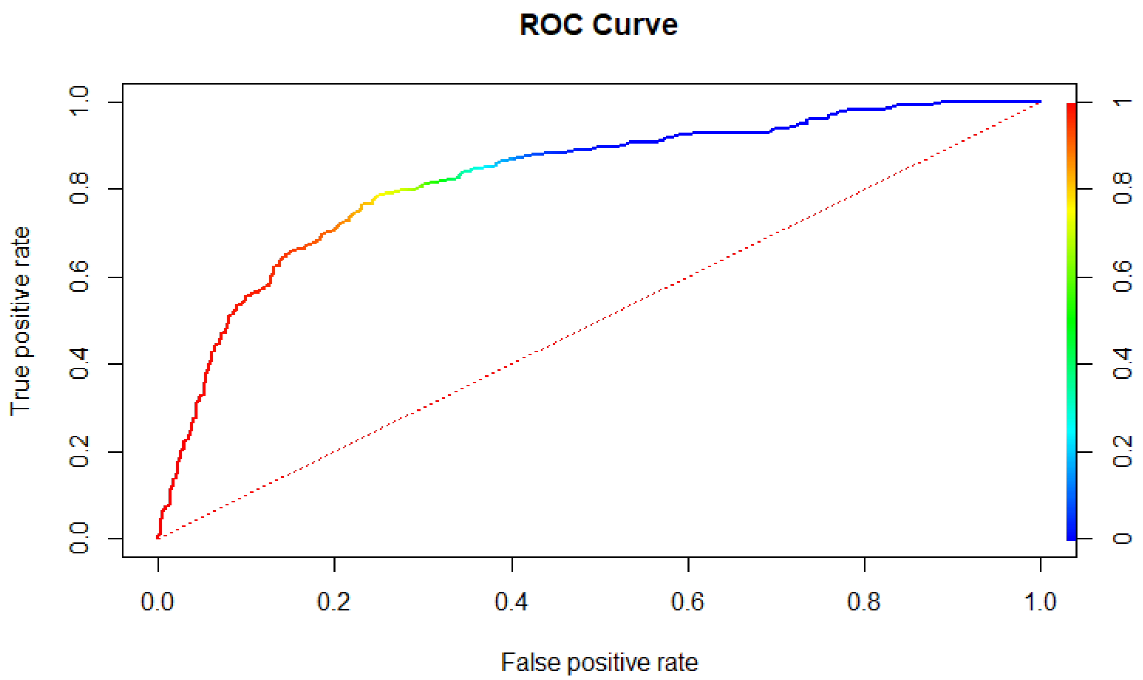
The closer the value of AUC is to 1, the better the quality of the classifier.

To display the curve and obtain its value we create the following function:

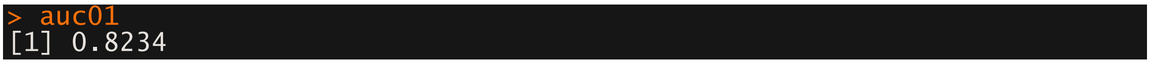
```
auc.trees=function(modelpred,test){
  pred=prediction(modelpred, test)
  perf=performance(pred, "tpr", "fpr")
  plot(perf, lwd=2, colorize=TRUE, main="ROC Curve")
  lines(x=c(0,1), y=c(0,1), col="red", lwd=1, lty=3)
  a=round(performance(pred, measure = "auc")@y.values[[1]],4)
  return (a)
}
```

Once the function is defined, we call it indicating the correct arguments

```
auc01=auc.trees(m01pred,test$churn)
```



The value we obtain corresponding to the AUC is 0.8234 and the representation of the ROC curve is as follows.



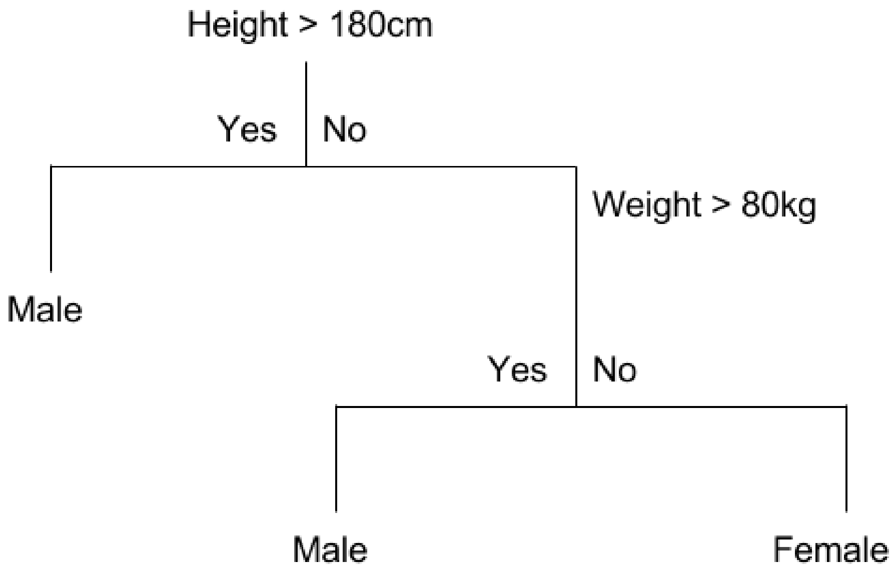
```
> auc01
[1] 0.8234
```

Decision tree

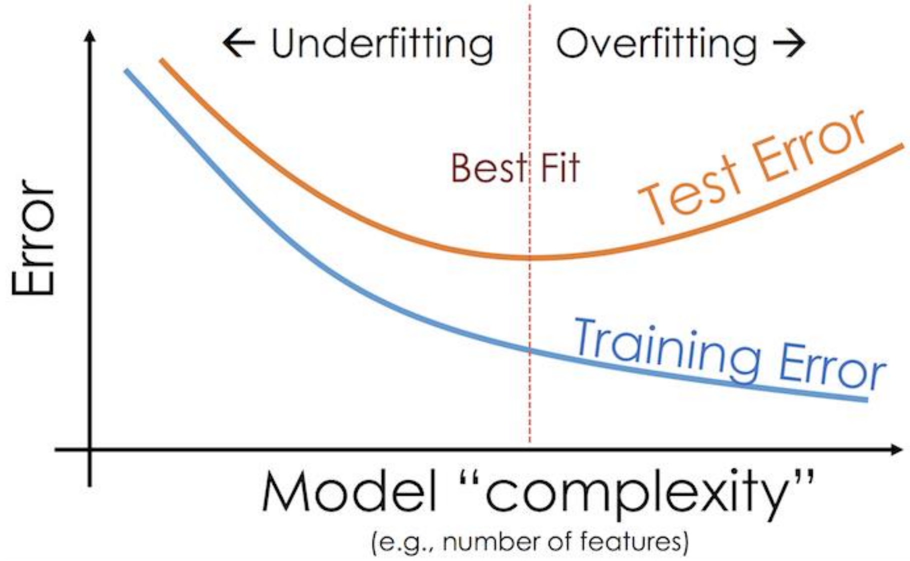
Next, we will use the decision tree, this method is the basis of all methodologies that use decision trees. It obtains quite good results.

The method consists of creating a binary decision tree in which each node corresponds to an input variable and each branch to the values that this variable can take. This allows us to decide about the nature of the dependent variable, knowing the rest of the dependent variables. You can assign a probability to each of the branches and also obtain the prediction in the form of a probability.

"A tree is built by splitting the source set, constituting the root node of the tree, into subsets which constitute the successor children. The splitting is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions"



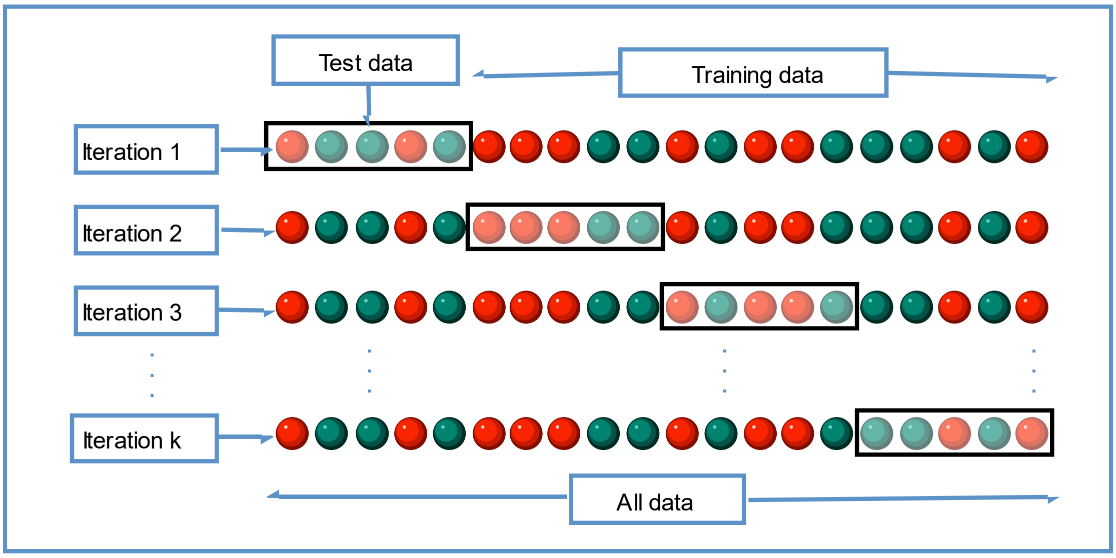
The problem with these trees is that they often incur a phenomenon called overfitting. This consists in that the tree that is created, is too faithful to the data with which it has been taught. That is to say, branches are created so that they describe particularities of the training data but then this error leads us to obtain worse results with the test data.



One way to avoid this is by doing a tree pruning process. That is, removing the last branches that give us a major test error.

To do this we use what we call cross-validation.

It consists of making several iterations using different training and test data within the same sample.



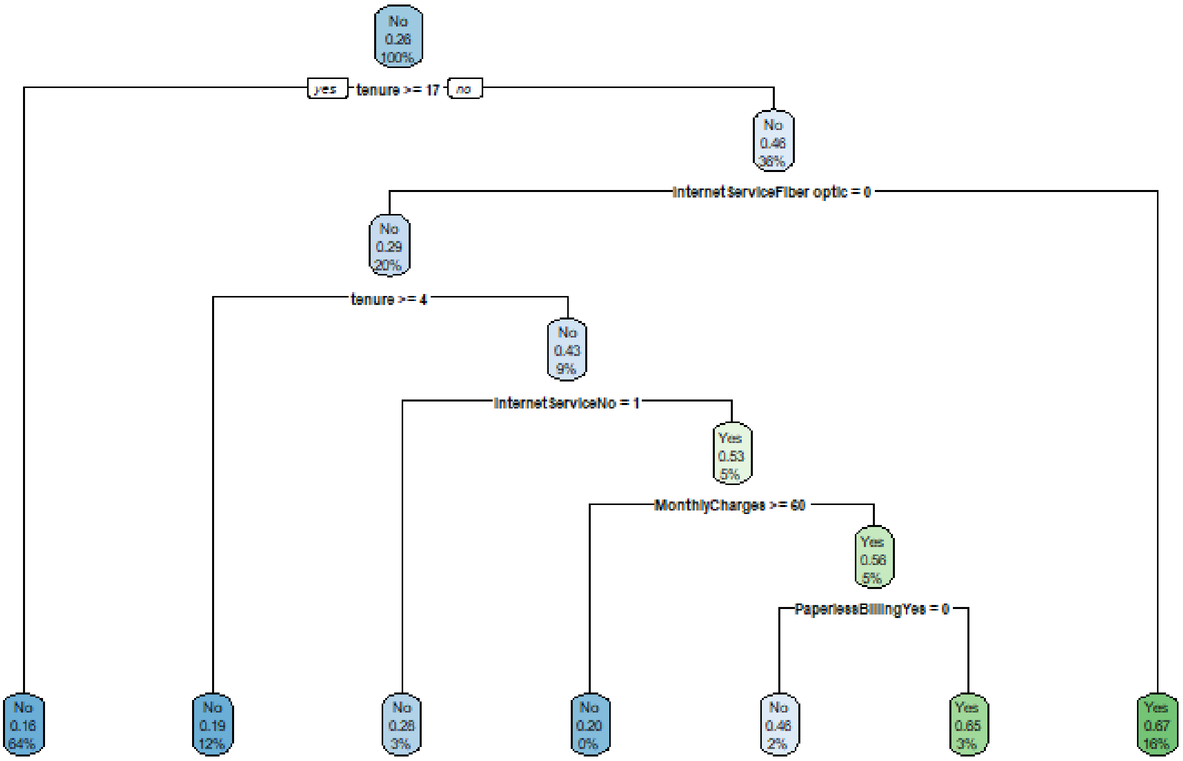
By making several iterations varying the training data (and also the depth of the tree), we get to know which is the optimal depth and thus it enables us to build the tree that gives us better results.

The called libraries include the option to use this method:

```
library(caret)

library(rpart)

library(rpart.plot)

trainctrl=trainControl(method = "repeatedcv", number = 5, repeats = 3)

m02=train(churn~., data=train, method="rpart", trControl=trainctrl)
```

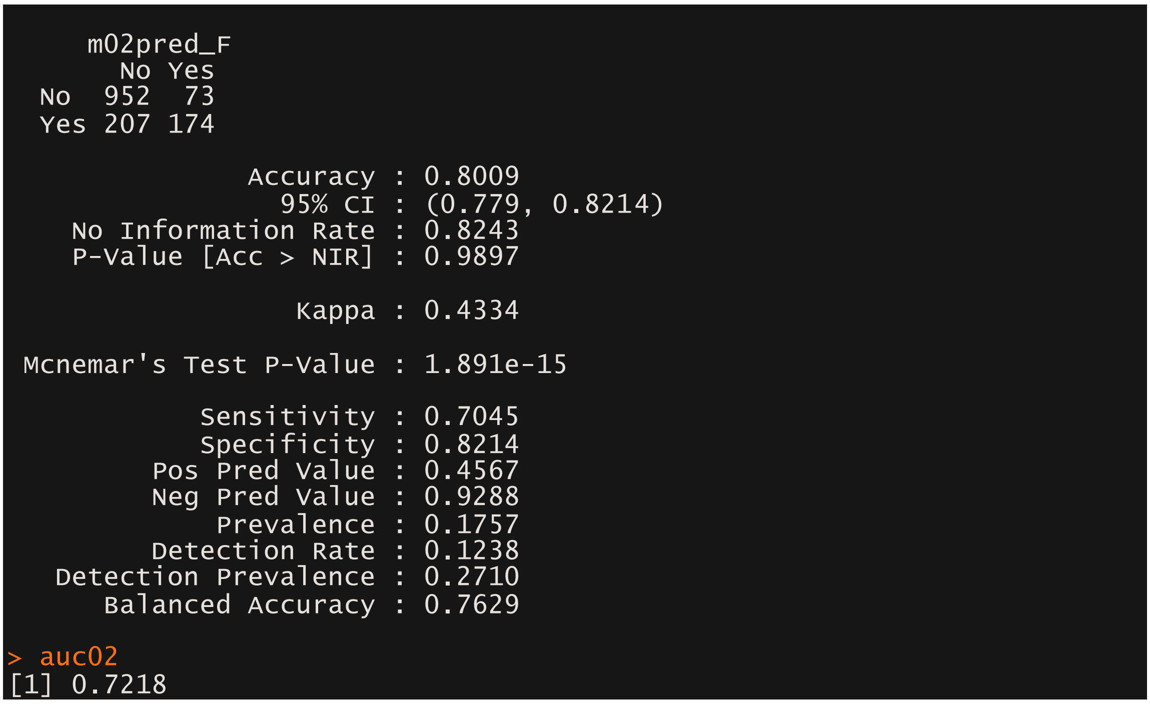
We visualize the tree we have built:

```
m002<-m02$finalModel

rpart.plot(m002, type = 2, fallen.leaves = T)
```



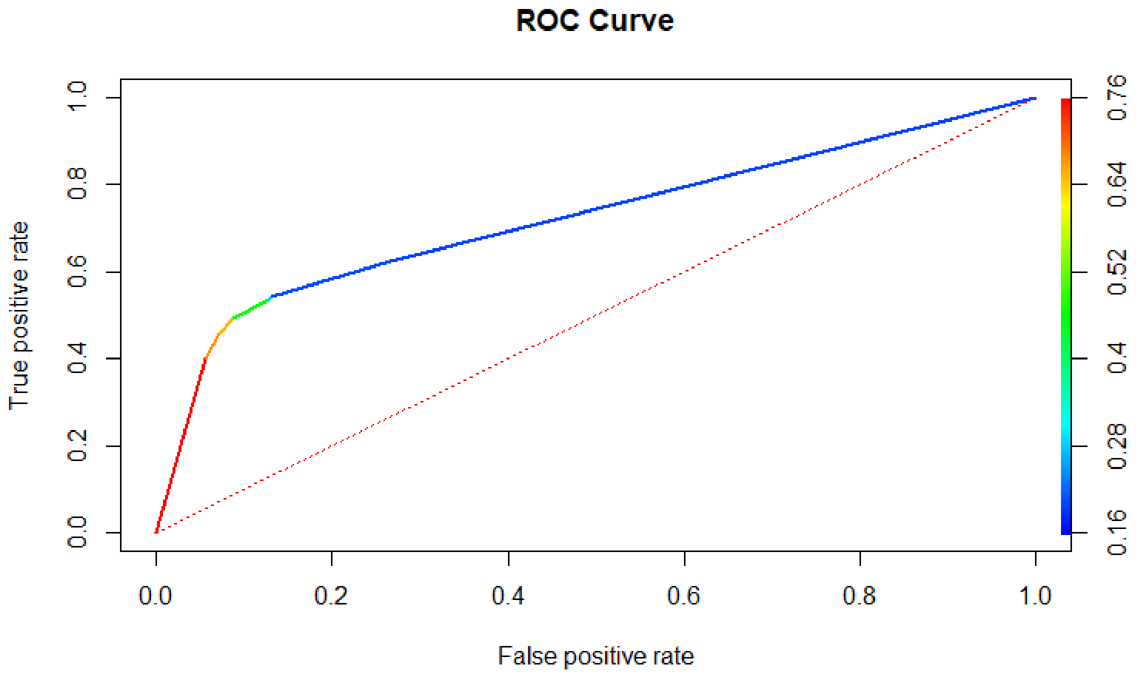
We use the tree that we have trained to make the desired predictions, and then calculate the confusion matrix and the AUC that we defined in the previous section.

```
m02pred=predict(m02, type = "prob", test)[,2]

m02pred_F=as.factor(round(m02pred))

levels(m02pred_F)=c("No","Yes")

cm02=confusionMatrix(table(test$churn, m02pred_F),positive = "Yes")
```



```
     m02pred_F
       No Yes
  No  952  73
  Yes 207 174
                                          
               Accuracy : 0.8009          
                 95% CI : (0.779, 0.8214) 
    No Information Rate : 0.8243          
    P-Value [Acc > NIR] : 0.9897          
                                          
                  Kappa : 0.4334          
                                          
 Mcnemar's Test P-Value : 1.891e-15       
                                          
            Sensitivity : 0.7045          
            Specificity : 0.8214          
         Pos Pred Value : 0.4567          
         Neg Pred Value : 0.9288          
             Prevalence : 0.1757          
         Detection Rate : 0.1238          
   Detection Prevalence : 0.2710          
      Balanced Accuracy : 0.7629          

> auc02
[1] 0.7218
```

auc02=auc.trees(m02pred,test$churn)



We see that the precession increases significantly to 80.01%, however, the AUC decreases.

Bagging

This technique, also known as Bootstrap aggregation, consists of applying the Bootstrap procedure to a machine learning algorithm, generally decision trees.

The bootstrap procedure basically consists of creating sub-samples, approximately 60% of the size of the learning data. These sub-samples (bags) are made up of randomly selected observations with replacement.

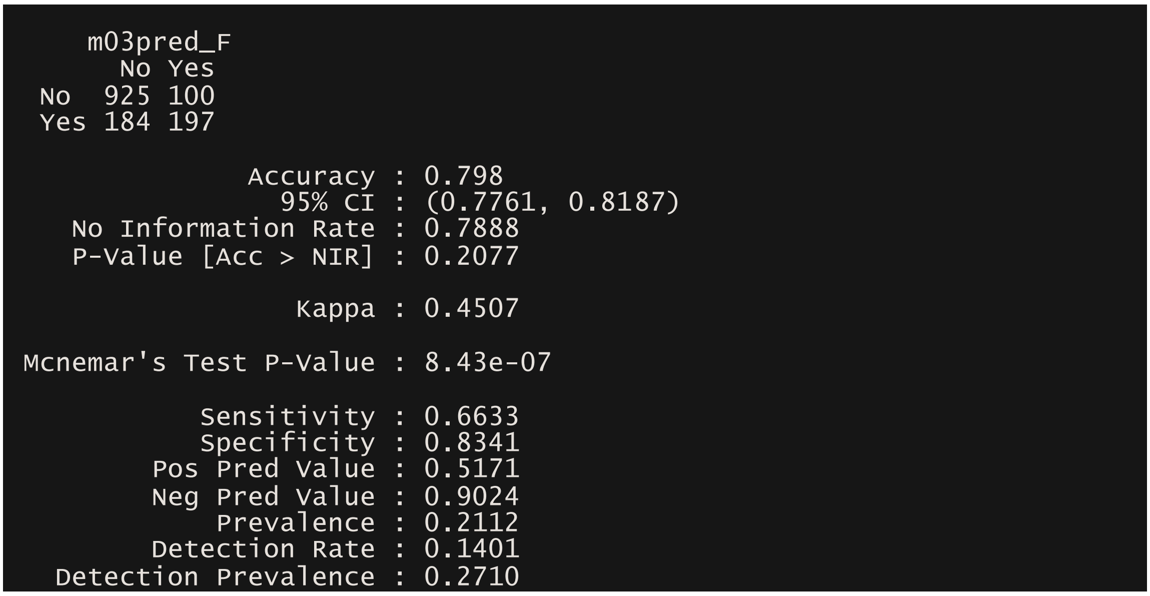
A tree is trained for each of these bags. This procedure is repeated hundreds of times so we may obtain an output of hundreds of trees that will constitute our model.

When using them, each tree will predict an output. We will keep as the classification prediction, the class predicted by most of the tress.

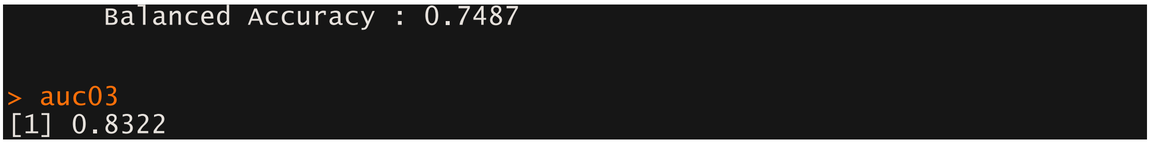
We implement this procedure through the following code:

```
library(randomForest)

m03=train(churn~., data=train, method="rf",tuneGrid=expand.grid(.mtry=16),
trControl=trainctrl)

m03pred=predict(m03, type = "prob", test)[,2]

m03pred_F=as.factor(round(m03pred))

levels(m03pred_F)=c("No","Yes")

cm03=confusionMatrix(table(test$churn,m03pred_F),positive = "Yes")

auc03=auc.trees(m03pred,test$churn)
```

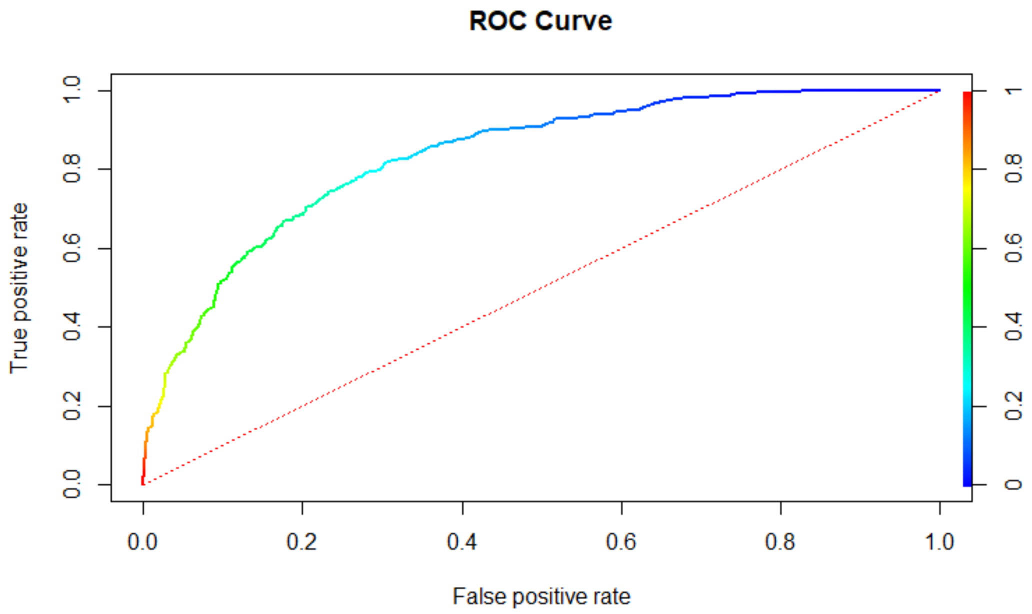
As a consequence, we get the following results:



```
     m03pred_F
       No  Yes
  No   925 100
  Yes  184 197

               Accuracy : 0.798
                 95% CI : (0.7761, 0.8187)
    No Information Rate : 0.7888
    P-Value [Acc > NIR] : 0.2077

                  Kappa : 0.4507

 Mcnemar's Test P-Value : 8.43e-07

            Sensitivity : 0.6633
            Specificity : 0.8341
         Pos Pred Value : 0.5171
         Neg Pred Value : 0.9024
             Prevalence : 0.2112
         Detection Rate : 0.1401
   Detection Prevalence : 0.2710
```



```
      Balanced Accuracy : 0.7487

> auc03
[1] 0.8322
```

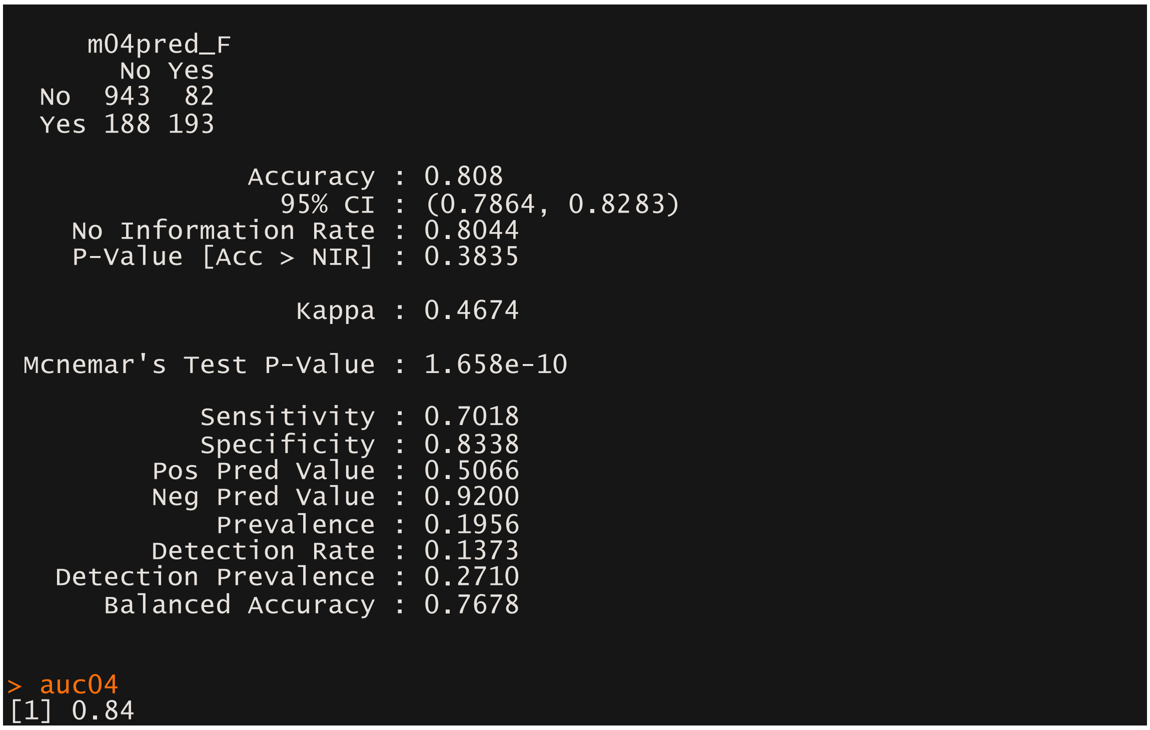


Random Forest

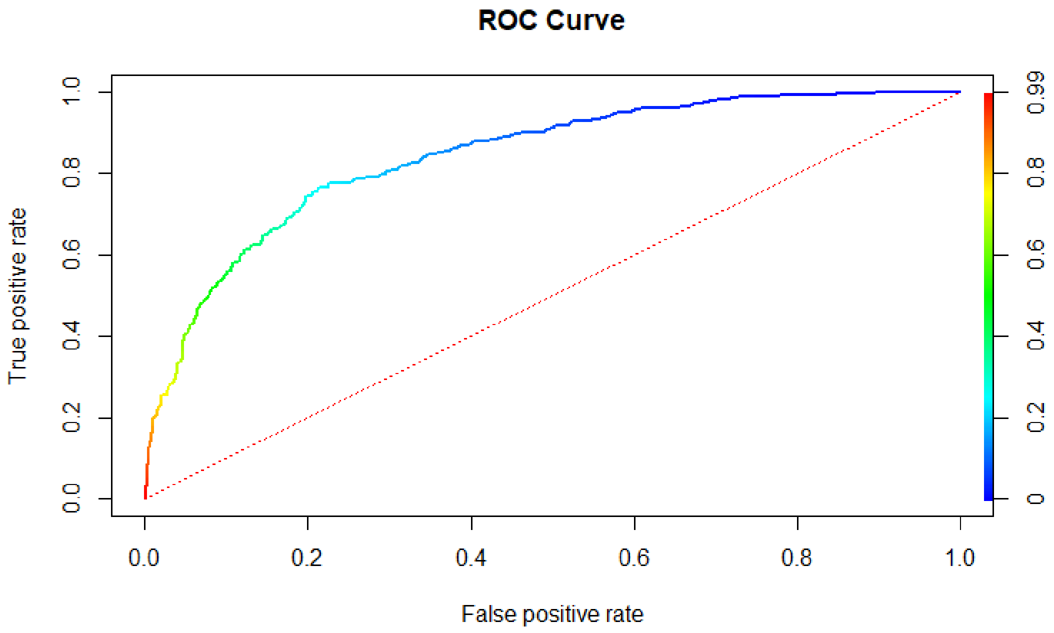
The Random Forest's methodology is very similar to bagging. We also have hundreds of randomly constituted sub-samples and for each one we build a tree. This time the trees will not be constructed considering all the independent variables. A certain number of variables will be considered for each tree, which will be chosen at random as well.

The number of recommended variables to consider in classification is the square root of the number of total independent variables. We have 16 independent variables so we will consider Sqrt(16)=4 variables.

```
m04=train(churn~., data=train, method="rf", tuneGrid=expand.grid(.mtry=4),
trControl=trainctrl)

m04pred=predict(m04, type = "prob", test)[,2]

m04pred_F=as.factor(round(m04pred))

levels(m04pred_F)=c("No","Yes")

cm04=confusionMatrix(table(test$churn,m04pred_F),positive = "Yes")

auc04=auc.trees(m04pred,test$churn)
```

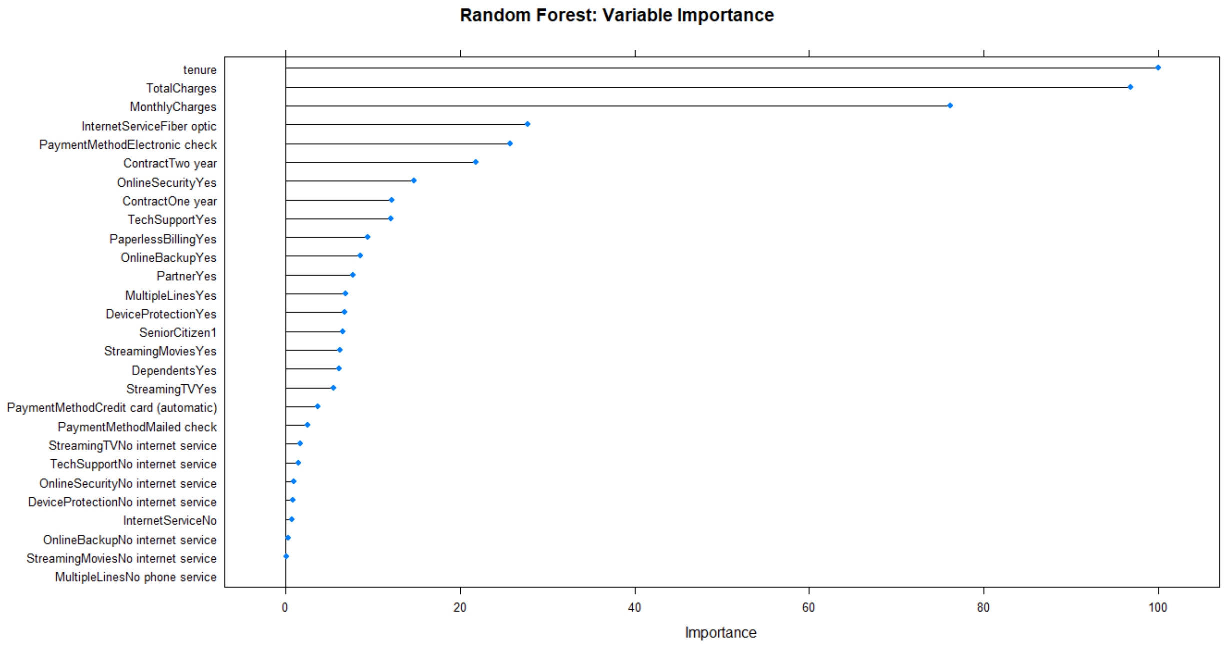


```
     m04pred_F
       No  Yes
  No   943   82
  Yes  188  193

                Accuracy : 0.808
                  95% CI : (0.7864, 0.8283)
    No Information Rate : 0.8044
    P-Value [Acc > NIR] : 0.3835

                   Kappa : 0.4674

 Mcnemar's Test P-Value : 1.658e-10

             Sensitivity : 0.7018
             Specificity : 0.8338
          Pos Pred Value : 0.5066
          Neg Pred Value : 0.9200
              Prevalence : 0.1956
          Detection Rate : 0.1373
    Detection Prevalence : 0.2710
       Balanced Accuracy : 0.7678

> auc04
[1] 0.84
```



The fact of having created a random forest allows us to visualize the importance of each one of the variables quickly.

plot(varImp(m04), main="Random Forest: Variable Importance")

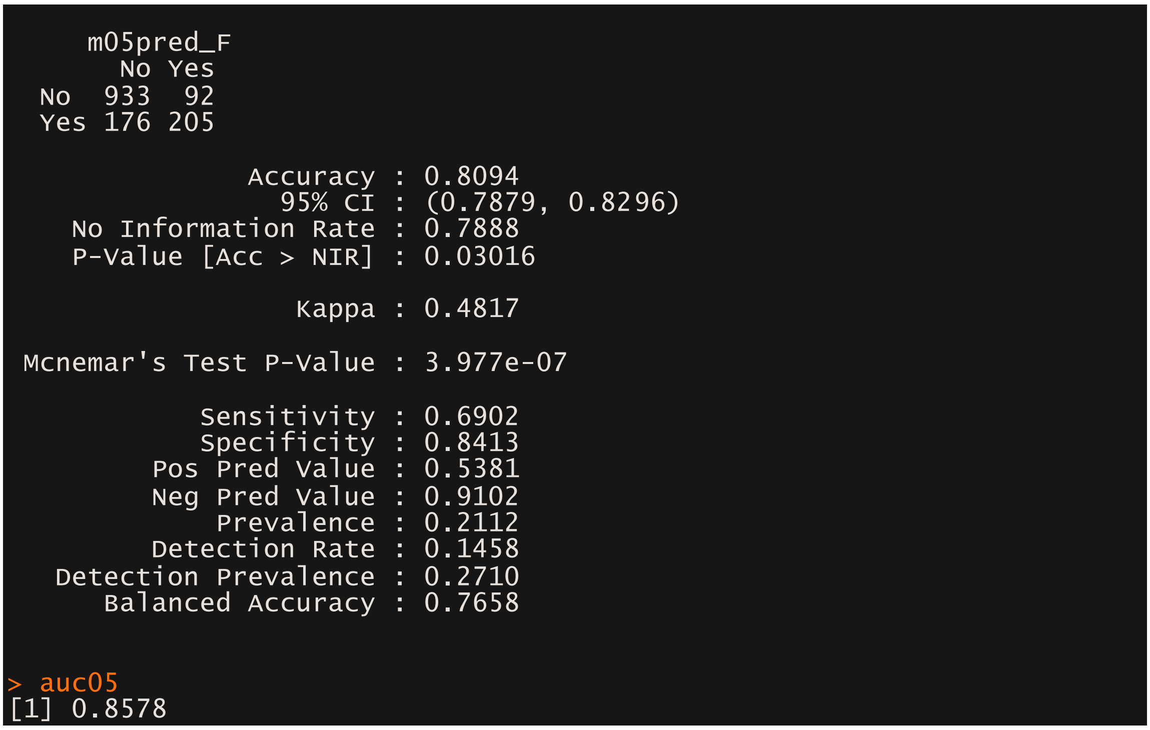


We verify that these results are quite similar to those offered by the Boruta algorithm

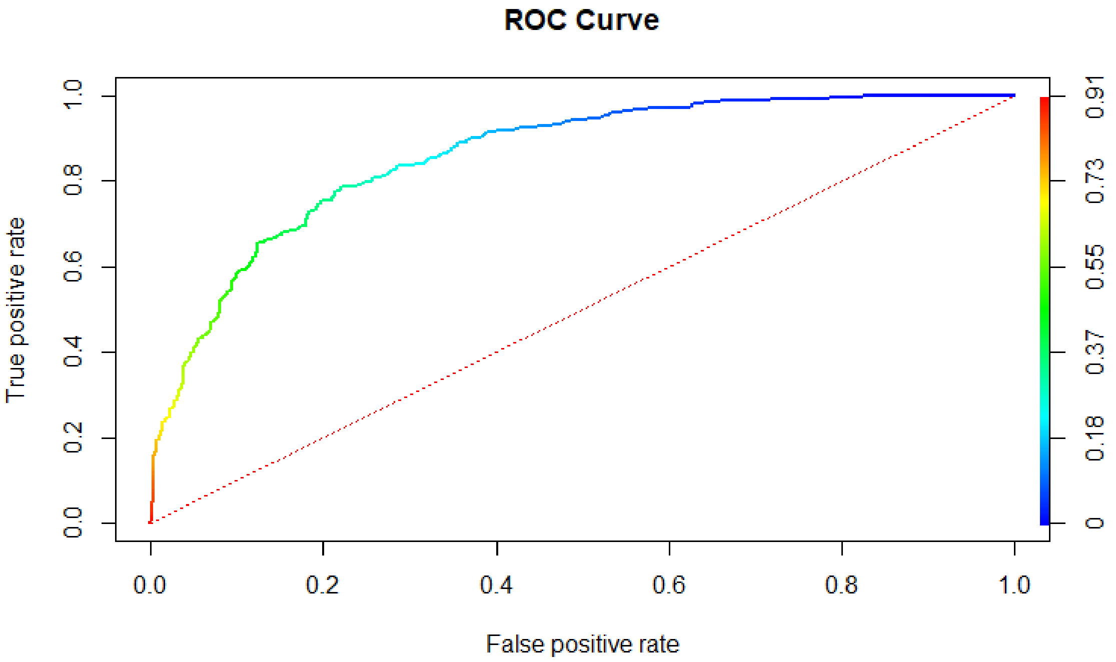
Xgboost

This algorithm is based on a general methodology called boosting. Like bagging and Random Forest, it uses a multitude of decision trees. The method consists of creating a decision tree and once created you obtain the residual data of the created tree. When we speak of residual data, we refer to the data resulting from the difference between the original data and the output of the tree. This process will be carried out iteratively and will allow us to go deeper into the observations where more errors occur, that can explain its great performance.

```
library(xgboost)

library(plyr)

trainctrl_2=trainControl(method = "repeatedcv", number = 5, repeats = 3,classProbs =
TRUE,allowParallel = TRUE)

m05=train(churn~., data=train, method="xgbTree",trControl=trainctrl_2,verbose=T)

m05pred=predict(m05, type = "prob", test)[,2]

m05pred_F=as.factor(round(m05pred))

levels(m05pred_F)<-c("No","Yes")

cm05=confusionMatrix(table(test$churn,m05pred_F),positive = "Yes")
```



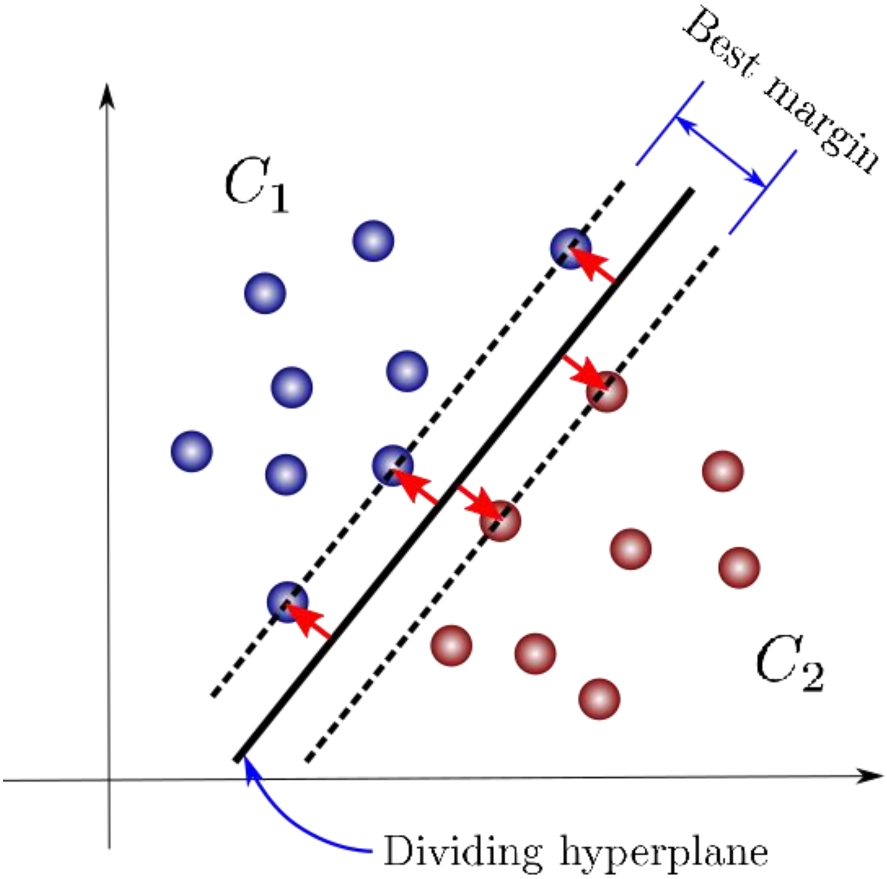
```
     m05pred_F
       No Yes
  No   933  92
  Yes 176 205

               Accuracy : 0.8094
                 95% CI : (0.7879, 0.8296)
    No Information Rate : 0.7888
    P-Value [Acc > NIR] : 0.03016

                  Kappa : 0.4817

 Mcnemar's Test P-Value : 3.977e-07

            Sensitivity : 0.6902
            Specificity : 0.8413
         Pos Pred Value : 0.5381
         Neg Pred Value : 0.9102
             Prevalence : 0.2112
         Detection Rate : 0.1458
   Detection Prevalence : 0.2710
      Balanced Accuracy : 0.7658

> auc05
[1] 0.8578
```



Support Vector Machine linear

An SVM is a model that represents the sample points in a space, separating the classes into 2 spaces as wide as possible by means of a defined separation hyperplane.

When the new samples are placed in correspondence with this model, depending on the spaces to which they belong, they can be classified in one or the other class.



Ideally, the SVM-based model should produce a hyperplane that completely separates the data of the studied universe into two categories. However, a perfect separation is not always possible, and if it is, the model output cannot be generalized to other data.

In order to allow some flexibility, the MSAs handle a C (cost) parameter that controls the compensation between training errors and rigid margins, thus creating a margin that allows for some errors in the classification while penalizing them.

To find out which cost gives the best results, we use the tune() function

```
tune.out06=tune(svm,churn~.,data=train,kernel="linear",ranges=list(cost=c(0.001, 0.01, 0.1,
1,5,10)))
```

These operations are computationally time consuming, even more when the data frame has a high number of variables and observations, as in our case. The execution of this calculation has come to take times that are close to the hour.

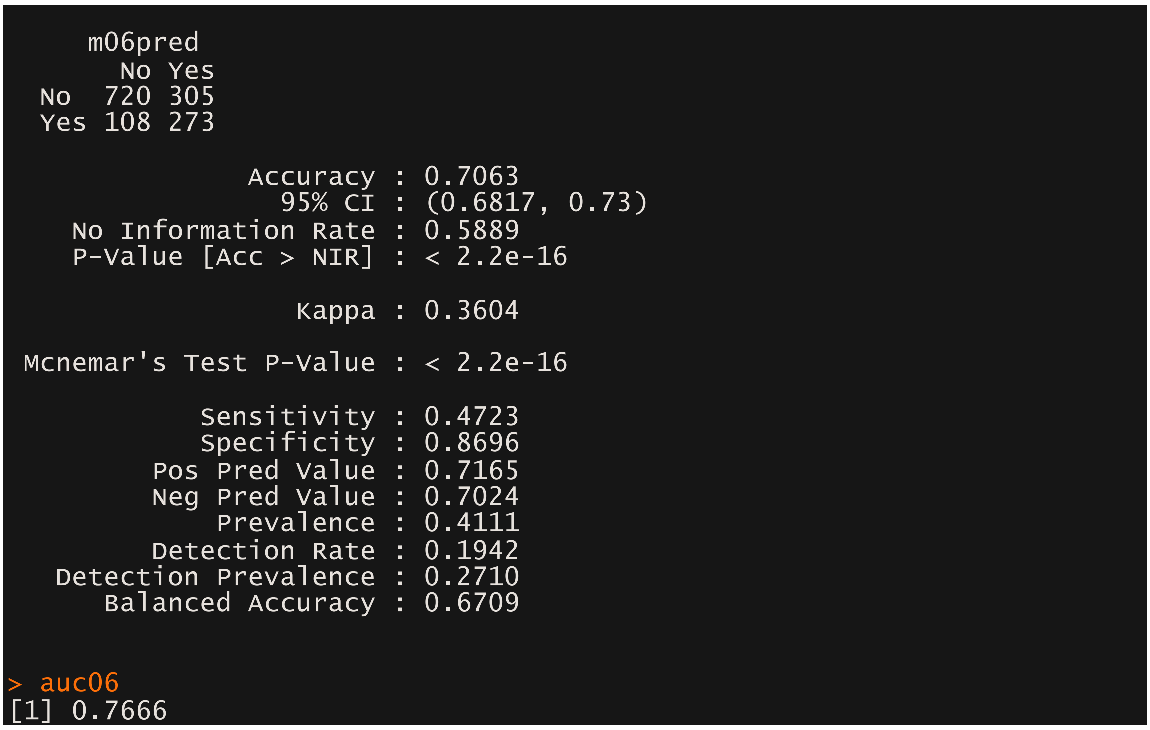
```
bestmod06=tune.out06$best.model
```

```
bestcost06=tune.out06$best.model$cost
```

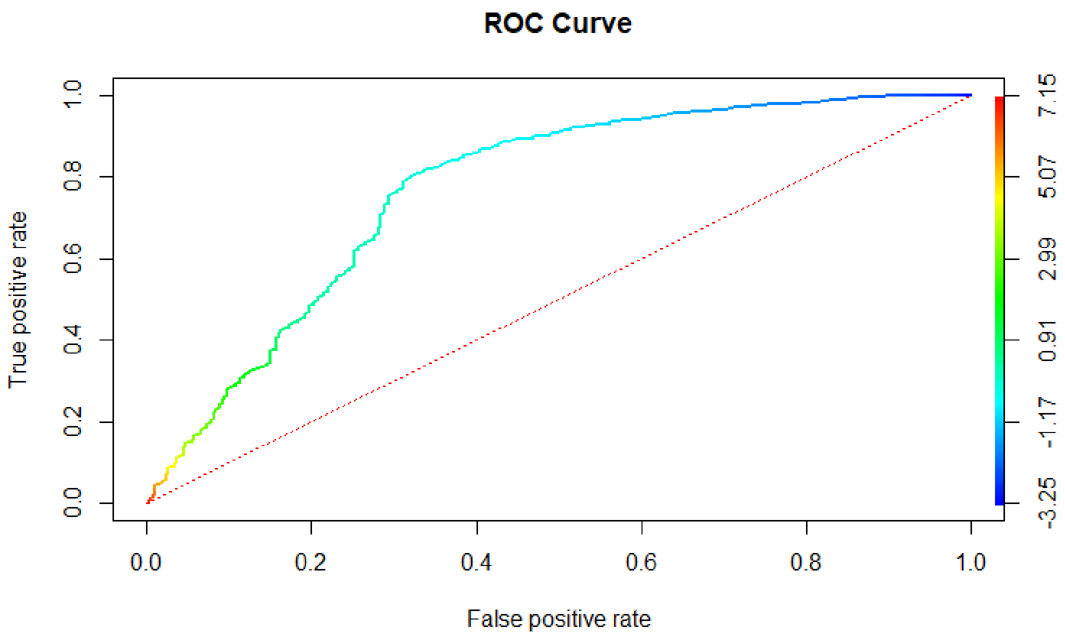
We note that the best cost corresponds to 0.01

Once the optimal cost is known, we create a model using the following code:

```
m06=svm(churn~., data=train, kernel="linear", cost=bestcost06,scale=FALSE)

m06pred=predict(m06,newdata=test,decision.values = T)

cm06=confusionMatrix(table(test$churn,m06pred),positive = "Yes")

auc06=auc.svm(m06pred,test$churn)
```



```
     m06pred
       No Yes
  No  720 305
  Yes 108 273

               Accuracy : 0.7063
                 95% CI : (0.6817, 0.73)
    No Information Rate : 0.5889
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3604

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.4723
            Specificity : 0.8696
         Pos Pred Value : 0.7165
         Neg Pred Value : 0.7024
             Prevalence : 0.4111
         Detection Rate : 0.1942
   Detection Prevalence : 0.2710
      Balanced Accuracy : 0.6709


> auc06
[1] 0.7666
```



Support Vector Machine radial

The simplest way to perform the separation is by means of a straight line, a straight map or an N-dimensional hyperplane.

Unfortunately, the universes to be studied are not usually presented in idyllic two-dimensional cases as in the previous example, but an SVM algorithm must deal with non-linear separation curves since the sets cannot be linearly separated. Due to the computational limitations of linear learning machines, they cannot be used in most real-world applications.

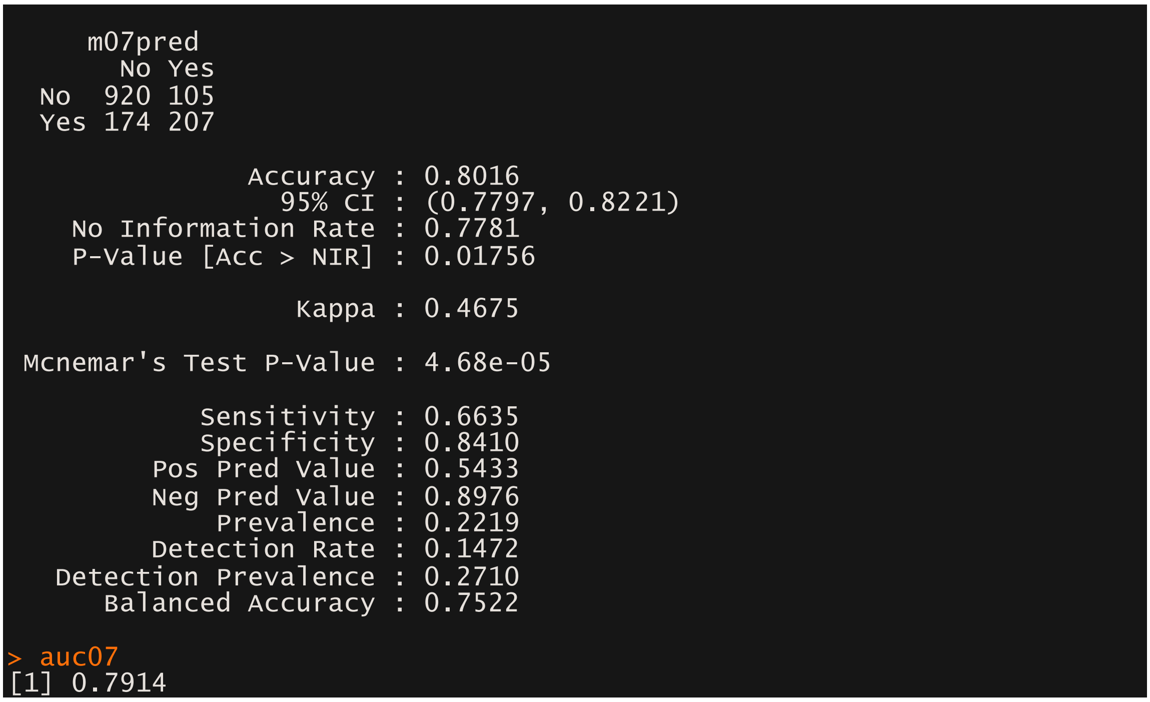
Kernel functions offer a solution to this problem by projecting the information into a larger feature space which increases the computational capacity of the linear learning machine.

In this case a radial kernel, in which the model will have to be learned having previously stipulated two parameters, one of cost as in the previous example and another one that we call gamma.

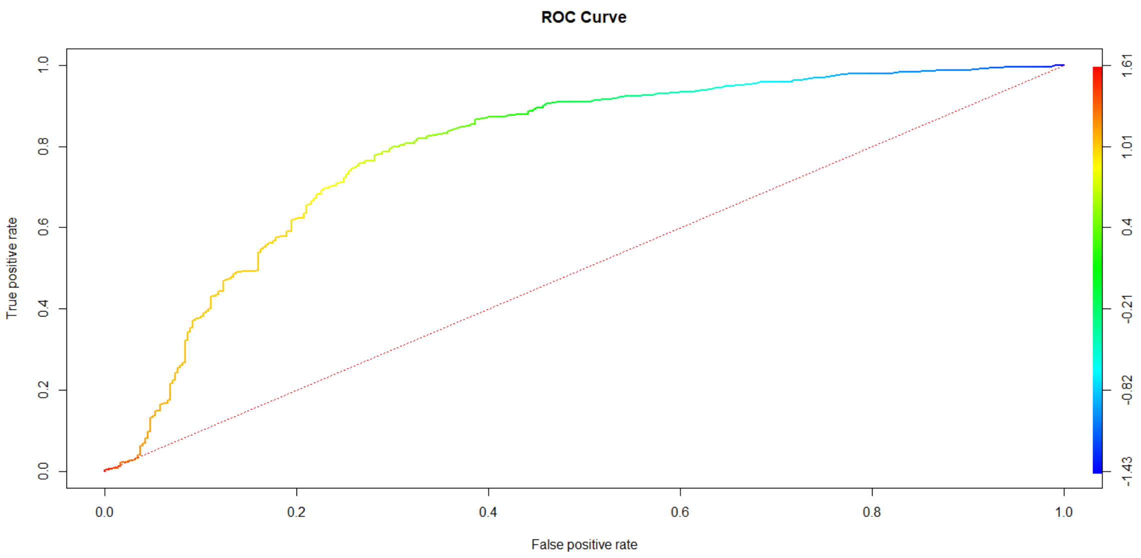
To know its optimal value, we use again the tune() function.

```
tune.out07=tune(svm, churn~., data=train, kernel="radial",
ranges=list(cost=c(0.1,1,10),gamma=c(0.5,1,2,3,4)))

bestmod07=tune.out07$best.model

bestcost07=tune.out07$best.model$cost

bestgamma07=tune.out07$best.model$gamma
```

```
m07=svm(churn~., data=train, kernel="radial",  gamma=bestgamma07, cost=bestcost07)

m07pred=predict(m07,newdata=test,decision.values = T)

cm07=confusionMatrix(table(test$churn,m07pred),positive = "Yes")

auc07=auc.svm(m07pred,test$churn)
```



```
     m07pred
       No Yes
  No   920 105
  Yes 174 207

               Accuracy : 0.8016
                 95% CI : (0.7797, 0.8221)
    No Information Rate : 0.7781
    P-Value [Acc > NIR] : 0.01756

                  Kappa : 0.4675

 Mcnemar's Test P-Value : 4.68e-05

            Sensitivity : 0.6635
            Specificity : 0.8410
         Pos Pred Value : 0.5433
         Neg Pred Value : 0.8976
             Prevalence : 0.2219
         Detection Rate : 0.1472
   Detection Prevalence : 0.2710
      Balanced Accuracy : 0.7522

> auc07
[1] 0.7914
```

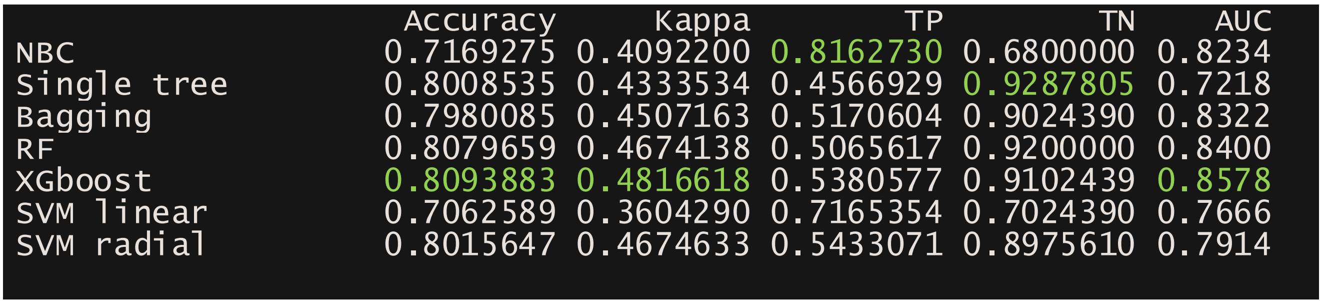


We clearly see that almost all indicators are favoured by this change to the radial kernel.

Once the indicators of the models presented have been calculated, it is time to compare them. To do so, we have put them all together in a comparative table. We select the indicators that are most relevant to our research.

In this case they are the following:

```
# ACCURACY

Accuracy=c(cm01$overall[1],cm02$overall[1],cm03$overall[1],cm04$overall[1],cm05$overall[1],cm06$overall[1],cm07$overall[1])

# KAPPA

Kappa=c(cm01$overall[2],cm02$overall[2],cm03$overall[2],cm04$overall[2],cm05$overall[2],cm06$overall[2],cm07$overall[2])

# TRUEPOSIVES

TP=c(cm01$byClass[3],cm02$byClass[3],cm03$byClass[3],cm04$byClass[3],cm05$byClass[3],cm06$byClass[3],cm07$byClass[3])

# TRUE NEGATIVES

TN=c(cm01$byClass[4],cm02$byClass[4],cm03$byClass[4],cm04$byClass[4],cm05$byClass[4],cm06$byClass[4],cm07$byClass[4])

#AREA UNDER THE CURVE

AUC=c(auc01,auc02,auc03,auc04,auc05,auc06,auc07)

# EVALUATION MODELS MATRIX

Evaluation1=data.frame(cbind(Accuracy, Kappa, TP, TN, AUC))

row.names(Evaluation1)=c("Naive Bayes classifier","Single tree", "Bagging", "Random Forest","XGboost", "SVM linear","SVM radial")

Evaluation1
```



```
                   Accuracy     Kappa        TP        TN    AUC
NBC               0.7169275 0.4092200 0.8162730 0.6800000 0.8234
Single tree       0.8008535 0.4333534 0.4566929 0.9287805 0.7218
Bagging           0.7980085 0.4507163 0.5170604 0.9024390 0.8322
RF                0.8079659 0.4674138 0.5065617 0.9200000 0.8400
XGboost           0.8093883 0.4816618 0.5380577 0.9102439 0.8578
SVM linear        0.7062589 0.3604290 0.7165354 0.7024390 0.7666
SVM radial        0.8015647 0.4674633 0.5433071 0.8975610 0.7914
```

We note that there has been one model that has stood out from the rest and that is the XGboost. It is easy to reach these conclusions since it presents the best results in almost each of the measures. The Random Forest algorithm has also obtained very good results, since all its measurements show values very similar to those of the Xgboost, but always a few hundredths below.

As a third option we have the model corresponding to the radial SVM that also obtains decent results in most of the indicators.

We also can note that NBC is the one which detects more positives but does not do a good detection of the negatives.

Cost sensitive classification

The indicators presented in the previous cases can give us an idea of the quality of the model in general. But one of the key elements to optimize our model as much as possible is to establish (or know) the cost of false predictions.

Let's take an example. We are trying to build a model that will detect early in time if a patient is going to have cancer. It would make no sense to build a model that considers a false positive and a false negative to be equally important. Common sense tells us that we will have to favour the number of detections of patients with cancer, even if this means reducing the values of the indicators described above.

Therefore, we will have to work on quantifying the average cost of each of the options. Obviously, if the cost of retaining a client is higher than the cost of attracting a new one, there will be no economic incentive to detect customer leakage.

In our case, there will be no cost associated with identifying a client that has been correctly classified, since the model is simply describing reality, whether it is good or bad.

However, it will generate a high cost for not being able to detect if a client is going to leave the company. The client would not receive any offer to retain him and the income he has brought will be gone.

There is also a cost in predicting that a client is going to leave the company when in fact he would not, because conditions would have been improved when we could have kept the client with the same conditions.

In the case of the telecommunications sector, the cost associated with acquiring a new customer is much higher than the cost of retaining that customer. It is estimated that it is between 5 and 15 times more expensive to acquire a customer than to retain one. This means that the cost of an FN is between 5 and 15 times more expensive than that of a FP. To simplify we will take the value of 5 and with it we will build the cost matrix.

The cost matrix is similar to the confusion matrix. The objective is to penalize errors (false positives and false negatives) versus successes (true negatives and true positives).

	Prediction		
		Negative	Positive
Real	Negative	0	C(FP)
	Positive	C(FN)	0

In this case our cost matrix will be as follows:

$$cost\ matrix = \begin{pmatrix} 0 & 1 \\ 5 & 0 \end{pmatrix}$$

The formula that will allow us to calculate the cost will be the following:

$$cost = 5C(FN) + C(FP)$$

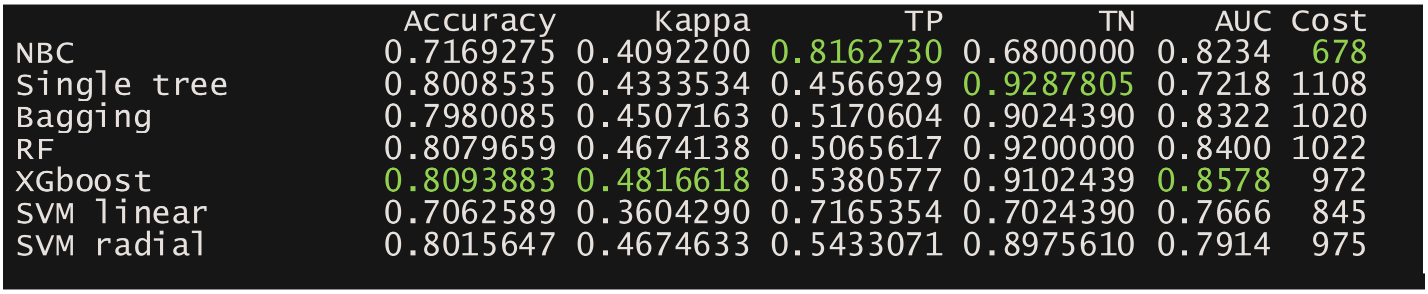
This cost is a relative cost, the value that represents only an indicator to compare the models. If we wanted the total cost to have a real meaning, we could weigh the customers with a value. To calculate this cost, we would have to analyse each false negative and false positive individually and weigh the cost with the value of each client.

We are content to consider that all customers have the same value.

To calculate the cost in R from the confusion matrix we create the following function:

```
cost=function(cm){

  mat=cm$table

  c=(5*mat[2,1]+mat[1,2])

  return(c)

}
```

Applying the function to the previous models we obtain:



```
                    Accuracy     Kappa        TP        TN    AUC Cost
NBC                0.7169275 0.4092200 0.8162730 0.6800000 0.8234  678
Single tree        0.8008535 0.4333534 0.4566929 0.9287805 0.7218 1108
Bagging            0.7980085 0.4507163 0.5170604 0.9024390 0.8322 1020
RF                 0.8079659 0.4674138 0.5065617 0.9200000 0.8400 1022
XGboost            0.8093883 0.4816618 0.5380577 0.9102439 0.8578  972
SVM linear         0.7062589 0.3604290 0.7165354 0.7024390 0.7666  845
SVM radial         0.8015647 0.4674633 0.5433071 0.8975610 0.7914  975
```

We realize that the model that would have maximized the company's profits would no longer have been the XGBoost, but the Naive Bayes Classifier.

As we discussed above, tree-based models also offer us a probabilistic output. In the following sections we will try to get better results at cost levels by modifying the threshold.

Random Forest

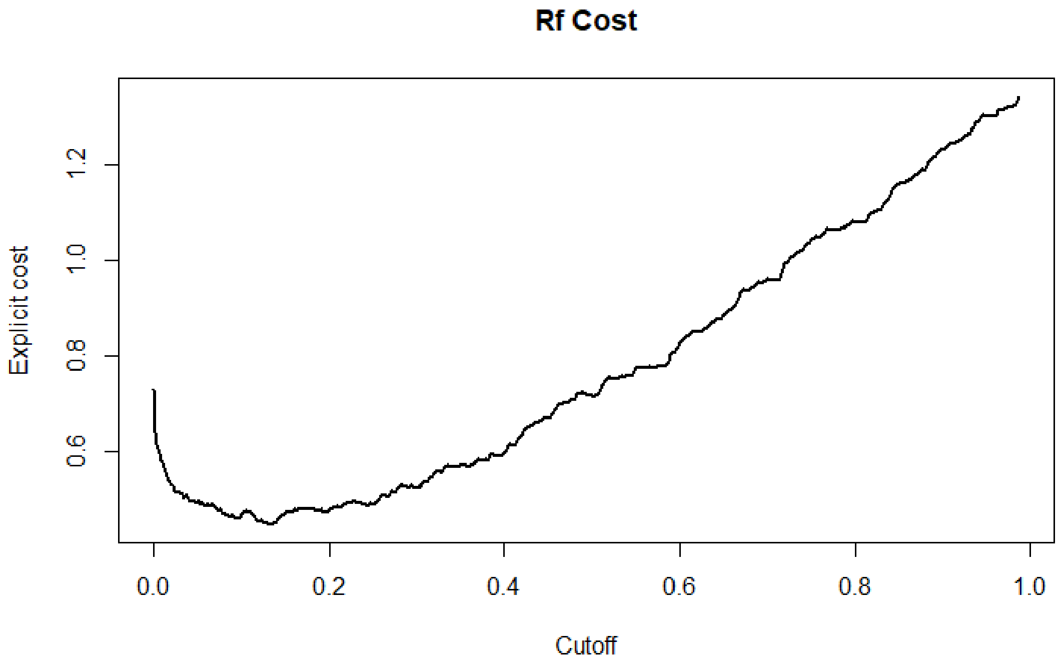
The objective therefore is to find the threshold that optimizes the results in terms of costs. To do this, we draw the evolution of the cost according to the threshold. It should be noted that the information from the test data is being graphed.

First, we create the model as before. We can actually recycle the one obtained in the previous section.

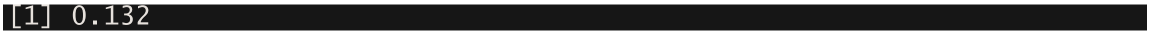
```
m09=train(churn~., data=train, method="rf",tuneGrid=expand.grid(.mtry=4),
trControl=trainctrl)

m09pred=predict(m09, type = "prob", test)[,2]
```

Now, we contrast cost and threshold, also called cuttoff:

```
m09_pred<-prediction(m09pred, test$churn)

m09_perf=performance(m09_pred,"cost","cutoff",cost.fp=1,cost.fn=5)

plot(m09_perf, lwd=2, main="Rf Cost")
```



```
df=data.frame(cut = m09_perf@x.values[[1]], cost = m09_perf@y.values[[1]])

df[which.min(df$cost), "cut"]
```



```
[1] 0.132
```

We see that the threshold value for which the cost would have been lower is 0.132.

We cannot know this value in advance, since we are looking at it in the test data that we don't have available when we build the model.

To calculate the optimal threshold based on the learning data, we create a function that, through a cross validation process, evaluates the optimal threshold in the different data packages and returns the average value.

```
optimalt.rf=function (training){

 yourdata=training[sample(nrow(training)),]

 folds <- cut(seq(1,nrow(yourdata)),breaks=5,labels=FALSE)

 i=0

 t=0

 for(i in 1:5){

  testIndexes=which(folds==i,arr.ind=TRUE)

  testData=yourdata[testIndexes, ]
```

```
    trainData=yourdata[-testIndexes, ]
    model=train(churn~., data=trainData, method="rf",tuneGrid=expand.grid(.mtry=4))
    model_pred=predict(model, type = "prob", testData)[,2]
    m_pred=prediction(model_pred, testData$churn)
    m_perf=performance(m_pred,"cost","cutoff",cost.fp=1,cost.fn=5)
    df=data.frame(cut = m_perf@x.values[[1]], cost = m_perf@y.values[[1]])
    t=t+df[which.min(df$cost), "cut"]
  }
  return (t/5)
}
```

Once this function is defined, we continue with the code that allows us to visualize the results of the new model.

```
t.rf=optimalt.rf(train)
m09pred_F=ifelse(m09pred < t.rf ,"No","Yes")
cm09=confusionMatrix(table(test$churn,m09pred_F),positive = "Yes")
auc09=auc.trees(m09pred,test$churn)
cost09=cost(cm09)
```

In this section we will represent the results directly in the comparison table.

Decision tree

The procedure for the single tree is very similar to the previous one. What we do is to simply change the learning method of the model.

```
optimalt.tree=function (training){
  yourdata=training[sample(nrow(training)),]
  folds <- cut(seq(1,nrow(yourdata)),breaks=5,labels=FALSE)
  i=0
  t=0
  for(i in 1:5){
    testIndexes=which(folds==i,arr.ind=TRUE)
    testData=yourdata[testIndexes, ]

    trainData=yourdata[-testIndexes, ]

    model=train(churn~., data=trainData, method="rpart", trControl=trainctrl)

    model_pred=predict(model, type = "prob", testData)[,2]

    m_pred<-prediction(model_pred, testData$churn)

    m_perf=performance(m_pred,"cost","cutoff",cost.fp=1,cost.fn=5)

    df <- data.frame(cut = m_perf@x.values[[1]], cost = m_perf@y.values[[1]])

    t=t+df[which.min(df$cost), "cut"]

      }

  return (t/5)

}
```

To obtain the results, we enter the corresponding code with a slight modification. This time we establish a minimum threshold value that will correspond to the minimum probability offered by the tree. This is done because it is likely that the threshold stays below the last value of probabilities offered by the tree. In this case the model will only predict one class and it will be a failure.

```
m08=train(churn~., data=train, method="rpart", trControl=trainctrl)

m08pred=predict(m08,type = "prob", test)[,2]

t.tree=optimalt.tree(train)

if(min(m08pred)>t.tree){t.tree=min(m18pred)}

m08pred_F=ifelse(m08pred <=t.tree ,"No","Yes")

cm08=confusionMatrix(table(test$churn, m08pred_F),positive = "Yes")

auc08=auc.trees(m08pred,test$churn)

cost08=cost(cm08)
```

Xgboost

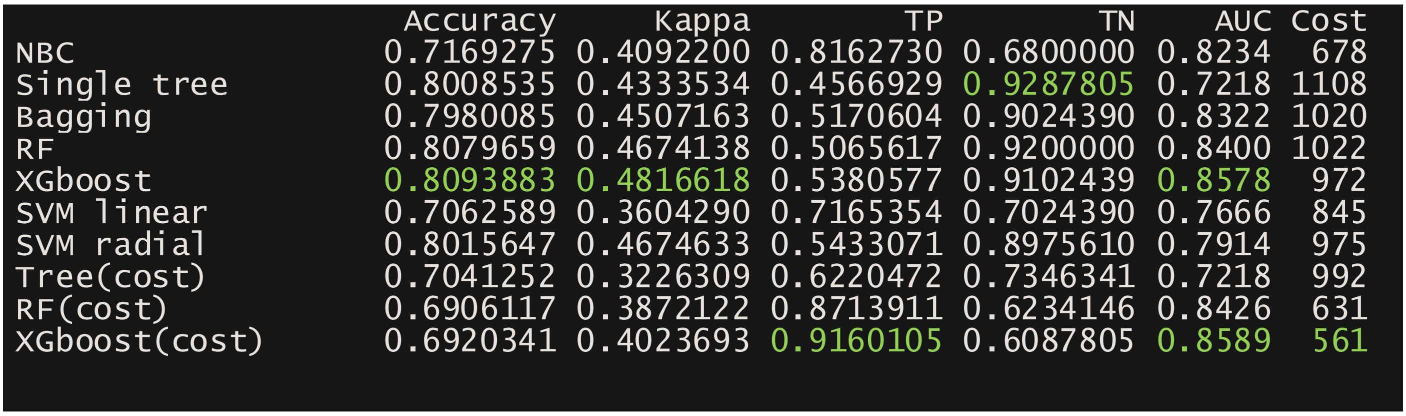
We performed the same process with this algorithm, which had previously given us the best results.

```
optimalt.Xgb=function (training){

  yourdata=training[sample(nrow(training)),]
  folds <- cut(seq(1,nrow(yourdata)),breaks=5,labels=FALSE)
  i=0
  t=0
  for(i in 1:5){
    testIndexes=which(folds==i,arr.ind=TRUE)
    testData=yourdata[testIndexes, ]
    trainData=yourdata[-testIndexes, ]
    model=train(churn~., data=trainData, method="xgbTree",trControl=trainctrl_2,verbose=T)
    model_pred=predict(model, type = "prob", testData)[,2]
    m_pred<-prediction(model_pred, testData$churn)
    m_perf=performance(m_pred,"cost","cutoff",cost.fp=1,cost.fn=5)
    df <- data.frame(cut = m_perf@x.values[[1]], cost = m_perf@y.values[[1]])
    t=t+df[which.min(df$cost), "cut"]
  }
  return (t/5)
}
m10=train(churn~., data=train, method="xgbTree",trControl=trainctrl_2,verbose=T)
m10pred=predict(m10, type = "prob", test)[,2]
t.xgb=optimalt.Xgb(train)
m10pred_F=ifelse(m10pred < t.xgb ,"No","Yes")
cm10=confusionMatrix(table(test$churn,m10pred_F),positive = "Yes")
auc10=auc.trees(m10pred,test$churn)
cost10=cost(cm10)
```

Once this has been completed, we can begin to evaluate the results of the new models.

```
Accuracy=c(cm01$overall[1],cm02$overall[1],cm03$overall[1],cm04$overall[1],cm05$overall[1],cm06$overall[1],cm07$overall[1],cm08$overall[1],cm09$overall[1],cm10$overall[1])

Kappa=c(cm01$overall[2],cm02$overall[2],cm03$overall[2],cm04$overall[2],cm05$overall[2],cm06$overall[2],cm07$overall[2],cm08$overall[2],cm09$overall[2],cm10$overall[2])

TP=c(cm01$byClass[3],cm02$byClass[3],cm03$byClass[3],cm04$byClass[3],cm05$byClass[3],cm06$byClass[3],cm07$byClass[3],cm08$byClass[3],cm09$byClass[3],cm10$byClass[3])
```

```
TN=c(cm01$byClass[4],cm02$byClass[4],cm03$byClass[4],cm04$byClass[4],cm05$byClass[4],cm06$byClass[4],cm07$byClass[4],cm08$byClass[4],cm09$byClass[4],cm10$byClass[4])

AUC=c(auc01,auc02,auc03,auc04,auc05,auc06,auc07,auc08,auc09,auc10)

Cost=c(cost01,cost02,cost03,cost04,cost05,cost06,cost07,cost08,cost09,cost10)

Evaluation2=data.frame(cbind(Accuracy, Kappa, TP, TN, AUC,Cost))

row.names(Evaluation2)=c("NBC","Single tree", "Bagging", "RF","XGboost", "SVM lin","SVM rad","Tree(cost)","RF(cost)","XGboost(cost)")

Evaluation2
```



```
                  Accuracy     Kappa        TP        TN    AUC Cost
NBC              0.7169275 0.4092200 0.8162730 0.6800000 0.8234  678
Single tree      0.8008535 0.4333534 0.4566929 0.9287805 0.7218 1108
Bagging          0.7980085 0.4507163 0.5170604 0.9024390 0.8322 1020
RF               0.8079659 0.4674138 0.5065617 0.9200000 0.8400 1022
XGboost          0.8093883 0.4816618 0.5380577 0.9102439 0.8578  972
SVM linear       0.7062589 0.3604290 0.7165354 0.7024390 0.7666  845
SVM radial       0.8015647 0.4674633 0.5433071 0.8975610 0.7914  975
Tree(cost)       0.7041252 0.3226309 0.6220472 0.7346341 0.7218  992
RF(cost)         0.6906117 0.3872122 0.8713911 0.6234146 0.8426  631
XGboost(cost)    0.6920341 0.4023693 0.9160105 0.6087805 0.8589  561
```

We see that the algorithms have managed to reduce the cost of the NBC model significantly. This has been done reducing the TN rate in order to favour the TP rate. Is easy to understand that the cost will be reduce because FP are more expensive than FN.

We see that the AUC increases slightly but this is due to the randomness of the new model. If we had not generated other models the AUC would have not change since we have simply modified the threshold.

Classification in synthetic balanced data

When you have a dependent variable with two categories where one is much larger than the other, the data is said to be unbalanced. As we have commented before this makes the algorithms work worse tending to favour the majority class. This is due to different reasons:

1. The use of measures of overall performance to drive the learning process, such as accuracy rate.

2. Classification rules that predict that the minority class is often highly specialized, and their coverage is very low, therefore they are discarded in favour of more general rules. If modelling is intended to avoid overfitting, it will tend to generalise.

3. Very small groupings of the minority class can be identified as noise, and therefore are often wrongly discarded by the classifier.

To avoid the problem of unbalance, different methods are available to balance the data. The initial idea would be to modify the size of the original database to obtain a new training set where the dependent variable is better balanced.

There are three main categories to solve the problem of unbalanced data:

1. Cost-sensitive learning (already used): Initially, all errors have the same weighting, but with this approach, different costs are assigned to the different errors, modifying the objective of the modelling. The classification error is no longer the variable to be minimized, but the cost associated with the classification. In the previous section we have used it with real costs, but it is also possible to establish a cost that compensates the imbalance of the classes.

2. Sampling of the data: This solution aims to balance the classes of a variable by adding or removing instances through a predetermined procedure. In turn, two subcategories can be differentiated:

a. Subsampling and Oversampling: remove or add instances to balance the classes.

b. Generation of synthetic data: create, based on the available data, instances of the minority class that are very similar to the data, but different.

3. Modification of the algorithm: This procedure would be oriented to adapt the learning method so that it is better tuned to the unbalanced class. This solution would imply the internal modification of the algorithms to set a target other than the general accuracy of the model. This methodology is not developed and applied in this paper.

We will use a technique called SMOTE (Synthetic Minority Oversampling Technique)

With this technique, the data set is balanced by generating artificial data, so it would be a form of oversampling but with better conditions. This technique generates a random set of minority class observations.

Bootstrapping and KNN (Nearest K-Neighbours algorithm) are used to generate the random set.

For more information on the technique used and other similar techniques, please refer to the following page:

https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family

We are therefore preparing to enlarge the set of training data available to us.

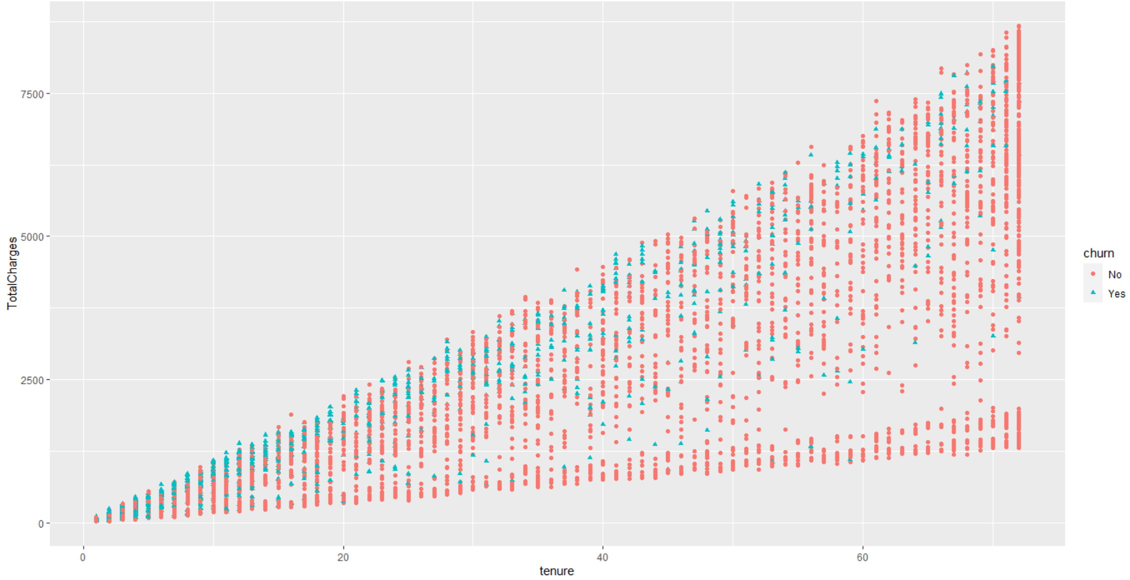
```
library(DMwR)

library(grid)

train_smote=SMOTE(churn~., train, perc.over=400, perc.under=120)

table(train_smote$churn)
```

```
  No  Yes 
7142 7440 
```

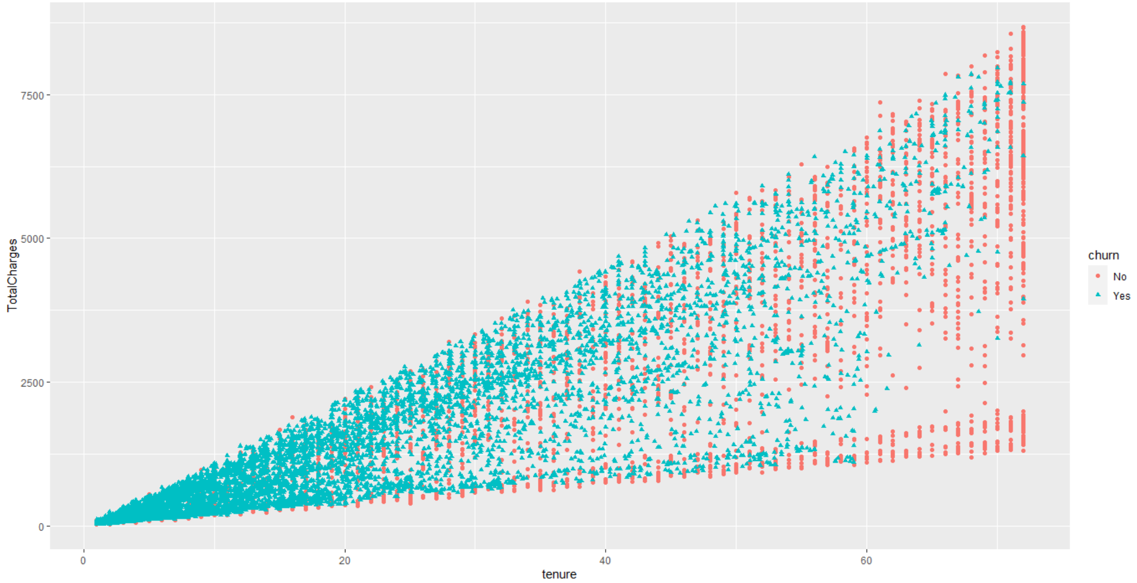
We checked on one side the number of instances generated and on the other side that the classes were correctly balanced.

We can also make a comparison of the generated observations by keeping an eye on the numerical variables, which is where we can best see the difference.

```
library(ggplot2)

ggplot(train, aes(tenure, TotalCharges)) + geom_point(aes(shape = churn, color = churn))
```

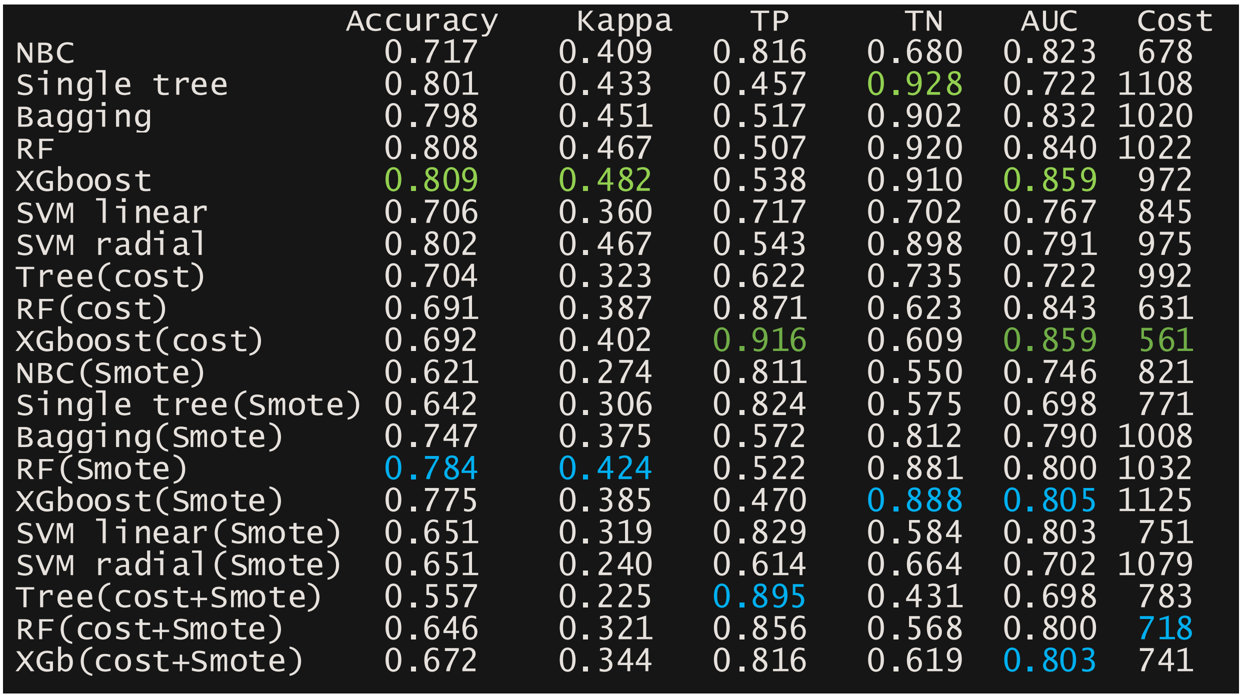


```
ggplot(train_smote, aes(tenure, TotalCharges)) + geom_point(aes(shape = churn, color = churn))
```



Once we have the new data set created, we are going to use exactly the same code just varying the input data. We will train the new models with different data, but we will asses them with same test data.

After some computing time, we visualize the results to compare them with the previous models:



```
                    Accuracy    Kappa     TP      TN     AUC   Cost
NBC                    0.717    0.409  0.816   0.680   0.823    678
Single tree            0.801    0.433  0.457   0.928   0.722   1108
Bagging                0.798    0.451  0.517   0.902   0.832   1020
RF                     0.808    0.467  0.507   0.920   0.840   1022
XGboost                0.809    0.482  0.538   0.910   0.859    972
SVM linear             0.706    0.360  0.717   0.702   0.767    845
SVM radial             0.802    0.467  0.543   0.898   0.791    975
Tree(cost)             0.704    0.323  0.622   0.735   0.722    992
RF(cost)               0.691    0.387  0.871   0.623   0.843    631
XGboost(cost)          0.692    0.402  0.916   0.609   0.859    561
NBC(Smote)             0.621    0.274  0.811   0.550   0.746    821
Single tree(Smote)     0.642    0.306  0.824   0.575   0.698    771
Bagging(Smote)         0.747    0.375  0.572   0.812   0.790   1008
RF(Smote)              0.784    0.424  0.522   0.881   0.800   1032
XGboost(Smote)         0.775    0.385  0.470   0.888   0.805   1125
SVM linear(Smote)      0.651    0.319  0.829   0.584   0.803    751
SVM radial(Smote)      0.651    0.240  0.614   0.664   0.702   1079
Tree(cost+Smote)       0.557    0.225  0.895   0.431   0.698    783
RF(cost+Smote)         0.646    0.321  0.856   0.568   0.800    718
XGb(cost+Smote)        0.672    0.344  0.816   0.619   0.803    741
```

It is easy to see that the big majority of the classifiers have performed worse results than the ones coming from the original data. The best model in terms of cost still being the XGboost.

We can also observe that this time the model which has performed better results if we only look the SMOTE models is the Random Forest but still far from the previous RF model. Another curious thing that we can observe is that the single tree has obtained a better result in terms of cost in comparation with the previous tree model.

Apart from this we can conclude that having created some synthetic data has been useless in this case.

Comparations and Conclusions

Actually, we have to know that concluding with just one iteration can be dangerous.

In some cases, when we create our model, we can have three different sources of randomness.

For example, in the case of random forest these are sources of randomness:

1) When we separate between learning data and test data

2) When we fill the bags through the bootstrap

3) When we hide randomly the variables to learn the trees

In order to not take false conclusions due to random reasons a good way to proceed is to calculate the mean and the variance of some iterations modifying the learning and test data.

It can be done with a cross validation procedure but in our case, we are just going to vary the random seed when we do the segregation between learning and test data.

```
set.seed(X)

tr=sample(1:nrow(telco), round(nrow(telco)*0.8))

train.telco=telco[tr,]
```

```
test.telco=telco[-tr,]

train=train.telco[,imp.var]

test=test.telco[,imp.var]
```

We will do 5 iterations where X will take all the entire values between 1 and 5. Of course, after each segregation we will compute all the code previously presented all along the document.

Then we will compute the mean and the variance of the most important indicators through a dynamic table in excel.

	Accuracy		AUC		Cost	
	mean	variance	mean	variance	mean	variance
NBC	0,7282	0,0130	0,8288	0,0105	665,4	47,99
Single tree	0,7935	0,0074	0,5943	0,2923	1153,6	101,34
Bagging	0,7925	0,0063	0,8293	0,0101	1030,2	46,60
RF	0,8065	0,0056	0,6916	0,3396	992,8	36,68
XGboost	0,8117	0,0018	0,8571	0,0074	952,8	15,39
SVM lin	0,7778	0,0402	0,7913	0,0295	1019,6	112,12
SVM rad	0,7997	0,0066	0,7924	0,0160	1009,6	41,23
Tree(cost)	0,7195	0,0291	0,5931	0,2917	955,2	42,74
RF(cost)	0,6752	0,0101	0,8439	0,0093	635,8	42,19
XGboost(cost)	0,6973	0,0086	0,8559	0,0078	577,6	29,86
NBC(Smote)	0,6200	0,0023	0,7411	0,0049	846,3	21,96
Single tree(Smote)	0,6195	0,0364	0,6908	0,0244	769,7	57,01
Bagging(Smote)	0,7475	0,0019	0,7888	0,0120	1003,0	17,06
RF(Smote)	0,7719	0,0104	0,7931	0,0145	1088,7	64,59
XGboost(Smote)	0,7779	0,0046	0,8052	0,0087	1113,7	28,73
SVM lin(Smote)	0,6316	0,0285	0,7783	0,0223	784,7	87,50
SVM rad(Smote)	0,6536	0,0082	0,7062	0,0108	1079,0	17,00
Tree(cost+Smote)	0,5055	0,1075	0,6908	0,0244	844,7	72,82
RF(cost+Smote)	0,6441	0,0076	0,7931	0,0145	729,7	49,54
XGb(cost+Smote)	0,6776	0,0101	0,8012	0,0045	712,0	43,49

Now we can conclude that in our case the best classifier is finally Xgboost. In the parameters of accuracy, AUC and cost the mean of the results is the highest. The variance value is also smaller than its first competitor which is Random Forest. That means that the results are not only better in terms of average but also in terms of stability.

We can also note that if we have a limited computational capacity the Naïve Bayes Classifier can also be a good model with quite good results and with just a few seconds of calculation time. Xgboost is taking a few minutes to finally create the model, most of this time is taken by finding the best threshold.

In conclusion, during this work we have been able to create a model which is able to predict successfully (in 81,17% of the cases) if a customer will leave the company. We have also been able to modify it in order to reduce as much as possible the cost of the churn problem. After doing that, we are able to detect 9 over 10 clients who are going to leave the company.

From these predictions, we will identify in an individual way the risk clients and we will be able to offer some improvements in their contracts to finally get our objective, keep them with us.

A good and practice way to manage the relation with the determined risk clients would be in an automatic way. We could segregate those clients in different groups depending on their needs and take different commercial offers for each group. Unsupervised learning would be the way to get it, but we leave for another work.

Bibliography

- Presentations of ML course, Eric Medvet
- An introduction to statistical learning, Gareth James
- https://machinelearningmastery.com/
- https://www.rdocumentation.org/
- https://www.datasciencecentral.com/
- https://www.wikipedia.org/