Árboles de decisión y Bagging para la detección de spam

Adolfo Sánchez Burón

Algoritmos empleados: Árboles de decisión y Bagging

Para un breve resumen del algoritmo de árboles de decisión mirar el post Árboles de decisión. Detección de churn (https://www.ml2projects.com/post/churn-ardec)

Bagging (Bootstrap Aggregating) es un algortimo de “ensemble” (ensamblaje), esto es, básicamente consiste en unir algoritmos simples para obtener otros más potentes. Los más usados son los de **bagging** y **boosting** (ver otro post (https://www.ml2projects.com/post/gbm-oov-cv)).

Modelos bagging

Los primeros tienen como objetivo la reducción de la varianza: sobre diferentes muestras aleatorias del set de entrenamiento se obtienen modelos predictivos simples diferentes. Por último, se ensamblan todos ellos en un único modelo, que es el promedio de los anteriores (en casos de regresión) o el más votado (en casos de clasificación).

Este algoritmo es especialmente recomendable en modelos con alta varianza (en caso contrario no mejora significativamente los resultados). Además, reduce el sobreaprendizaje ya que cada modelo simple es independiente del anterior, con muestras aleatorias diferentes.

Random forest (ver otro post (https://www.ml2projects.com/post/churn-randomf)) es una técnica que emplea el bagging, pero además de emplear muestras aleatorias en cada paso, también utiliza diferentes variables.

Modelos boosting

Los modelos boosting siguen un procedimiento secuencial: cada modelo está relacionado con el anterior, teniendo en cuenta sus errores, es decir, cada modelo mejora al anterior.

El objetivo de estos es reducir el sesgo. Finalmente, en casos de clasificación se selecciona el modelo predictivo final mediante votación, mientras que en casos de regresión se emplea una suma ponderada.

Son recomendables estas lecturas:

Amat Rodrigo, J. Árboles de decisión, random forest, gradient boosting y C5.0. (https://rpubs.com/Joaquin_AR/255596)

aporras. What is the difference between Bagging and Boosting? (https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/)

Brownlee, J. Bagging and Random Forest Ensemble Algorithms for Machine Learning. (https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/)

Características del caso

Se emplea el data set incluido en la librería de “kernlab” de R. Para ver una descripción puede consultarse ÙCI (http://archive.ics.uci.edu/ml/datasets/Spambase/). Básicamente consta de 4601 observaciones con 47 variables, una de las cuales indica si el mensaje era efectivamente spam. El resto de variables informa si una palabra determinada estaba incluida en una mensaje.

El objetivo del caso es predecir la probabilidad de que un determinado mail sea spam o no lo sea, en función de las palabras y caracteres que incluye el mensaje.

Proceso

1. Entorno

El primer punto tratará sobre la preparación del entorno, donde se mostrará la descarga de las librerías empleadas y la importación de datos.

2. Análisis descriptivo

Se mostrarán y explicarán las funciones empleadas en este paso, dividiéndolas en tres grupos: Análisis inicial, Tipología de datos y Análisis descriptivo (gráficos).

3. Preparación de la modelización

Donde se tratará

a. Preselección de variables predictoras

b. Funciones para la modelización

c. Particiones del dataset en dos grupos: training (70%) y test (30%)

4. Modelización

Por motivos didácticos, se dividirá la modelización de los dos algoritmos en una sucesión de pasos.

---

1. Entorno

1.1. Instalar librerías

```
library(dplyr)
library(knitr)       # For Dynamic Report Generation in R
library(ROCR)        # Model Performance and ROC curve
library(caret)       # Classification and Regression Training -  for any machine learn
ing algorithms
library(kernlab)     #para utilizar el dataset de "spam"
library(randomForest) # para determinar el poder predictivo de las variables
library(DataExplorer) #para realizar el análisis descriptivo con gráficos
library(rpart)       # Crear arboles de decisión
library(rpart.plot)  # Gráfico del árbol
#install.packages("ipred")
library(ipred)       # for bagged decision trees
```

```
options(scipen=999)
#Desactiva la notación científica
```

1.2. Importar datos

Se importan el dataset directamente de la librería “kernlab”.

```
#library(kernlab)
data(spam)
```

---

2. Análisis descriptivo

2.1. Análisis inicial

Se comienza viendo cómo es la estructura y características del dataset mediante “head” y “str”.

```
head(spam)
```

```
##   make address  all num3d  our over remove internet order mail receive will
## 1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00  0.00 0.00    0.00 0.64
## 2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07  0.00 0.94    0.21 0.79
## 3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12  0.64 0.25    0.38 0.45
## 4 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31
## 5 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31
## 6 0.00    0.00 0.00     0 1.85 0.00   0.00     1.85  0.00 0.00    0.00 0.00
##   people report addresses free business email  you credit your font num000
## 1   0.00   0.00      0.00 0.32     0.00  1.29 1.93   0.00 0.96    0   0.00
## 2   0.65   0.21      0.14 0.14     0.07  0.28 3.47   0.00 1.59    0   0.43
## 3   0.12   0.00      1.75 0.06     0.06  1.03 1.36   0.32 0.51    0   1.16
## 4   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 5   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 6   0.00   0.00      0.00 0.00     0.00  0.00 0.00   0.00 0.00    0   0.00
##   money hp hpl george num650 lab labs telnet num857 data num415 num85
## 1  0.00  0   0      0      0   0    0      0      0    0      0     0
## 2  0.43  0   0      0      0   0    0      0      0    0      0     0
## 3  0.06  0   0      0      0   0    0      0      0    0      0     0
## 4  0.00  0   0      0      0   0    0      0      0    0      0     0
## 5  0.00  0   0      0      0   0    0      0      0    0      0     0
## 6  0.00  0   0      0      0   0    0      0      0    0      0     0
##   technology num1999 parts pm direct cs meeting original project   re  edu
## 1          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 2          0    0.07     0  0   0.00  0       0     0.00       0 0.00 0.00
## 3          0    0.00     0  0   0.06  0       0     0.12       0 0.06 0.06
## 4          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 5          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 6          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
##   table conference charSemicolon charRoundbracket charSquarebracket
## 1     0          0          0.00            0.000                 0
## 2     0          0          0.00            0.132                 0
## 3     0          0          0.01            0.143                 0
## 4     0          0          0.00            0.137                 0
## 5     0          0          0.00            0.135                 0
## 6     0          0          0.00            0.223                 0
##   charExclamation charDollar charHash capitalAve capitalLong capitalTotal type
## 1           0.778      0.000    0.000      3.756          61          278 spam
## 2           0.372      0.180    0.048      5.114         101         1028 spam
## 3           0.276      0.184    0.010      9.821         485         2259 spam
## 4           0.137      0.000    0.000      3.537          40          191 spam
## 5           0.135      0.000    0.000      3.537          40          191 spam
## 6           0.000      0.000    0.000      3.000          15           54 spam
```

```
str(spam)
```

```
## 'data.frame':    4601 obs. of  58 variables:
##  $ make            : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ address         : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
##  $ all             : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
##  $ num3d           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ our             : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
##  $ over            : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
##  $ remove          : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
##  $ internet        : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
##  $ order           : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
##  $ mail            : num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
##  $ receive         : num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
##  $ will            : num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
##  $ people          : num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
##  $ report          : num  0 0.21 0 0 0 0 0 0 0 0 ...
##  $ addresses       : num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
##  $ free            : num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
##  $ business        : num  0 0.07 0.06 0 0 0 0 0 0 0 ...
##  $ email           : num  1.29 0.28 1.03 0 0 0 0.32 0 0.15 0.12 ...
##  $ you             : num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
##  $ credit          : num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
##  $ your            : num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
##  $ font            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num000          : num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
##  $ money           : num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
##  $ hp              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ hpl             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ george          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num650          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ lab             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ labs            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ telnet          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num857          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ data            : num  0 0 0 0 0 0 0 0 0.15 0 ...
##  $ num415          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num85           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ technology      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num1999         : num  0 0.07 0 0 0 0 0 0 0 0 ...
##  $ parts           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pm              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ direct          : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ cs              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ meeting         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ original        : num  0 0 0.12 0 0 0 0 0 0.3 0 ...
##  $ project         : num  0 0 0 0 0 0 0 0 0 0.06 ...
##  $ re              : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ edu             : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ table           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ conference      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charSemicolon   : num  0 0 0.01 0 0 0 0 0 0 0.04 ...
##  $ charRoundbracket : num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 0.271 0.03
```

```
...
##  $ charSquarebracket: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charExclamation  : num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0.244 ...
##  $ charDollar       : num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
##  $ charHash         : num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
##  $ capitalAve       : num  3.76 5.11 9.82 3.54 3.54 ...
##  $ capitalLong      : num  61 101 485 40 40 15 4 11 445 43 ...
##  $ capitalTotal     : num  278 1028 2259 191 191 ...
##  $ type             : Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

Cambiamos el nombre de la variable “type” a “target” (la variable dependiente que queremos predecir) y el dataset lo renombramos de “spam” a “df”.

```
spam$target <- as.factor(spam$type)
df <- select(spam,-one_of(c('type')))

#Observamos la estructura
str(df)
```

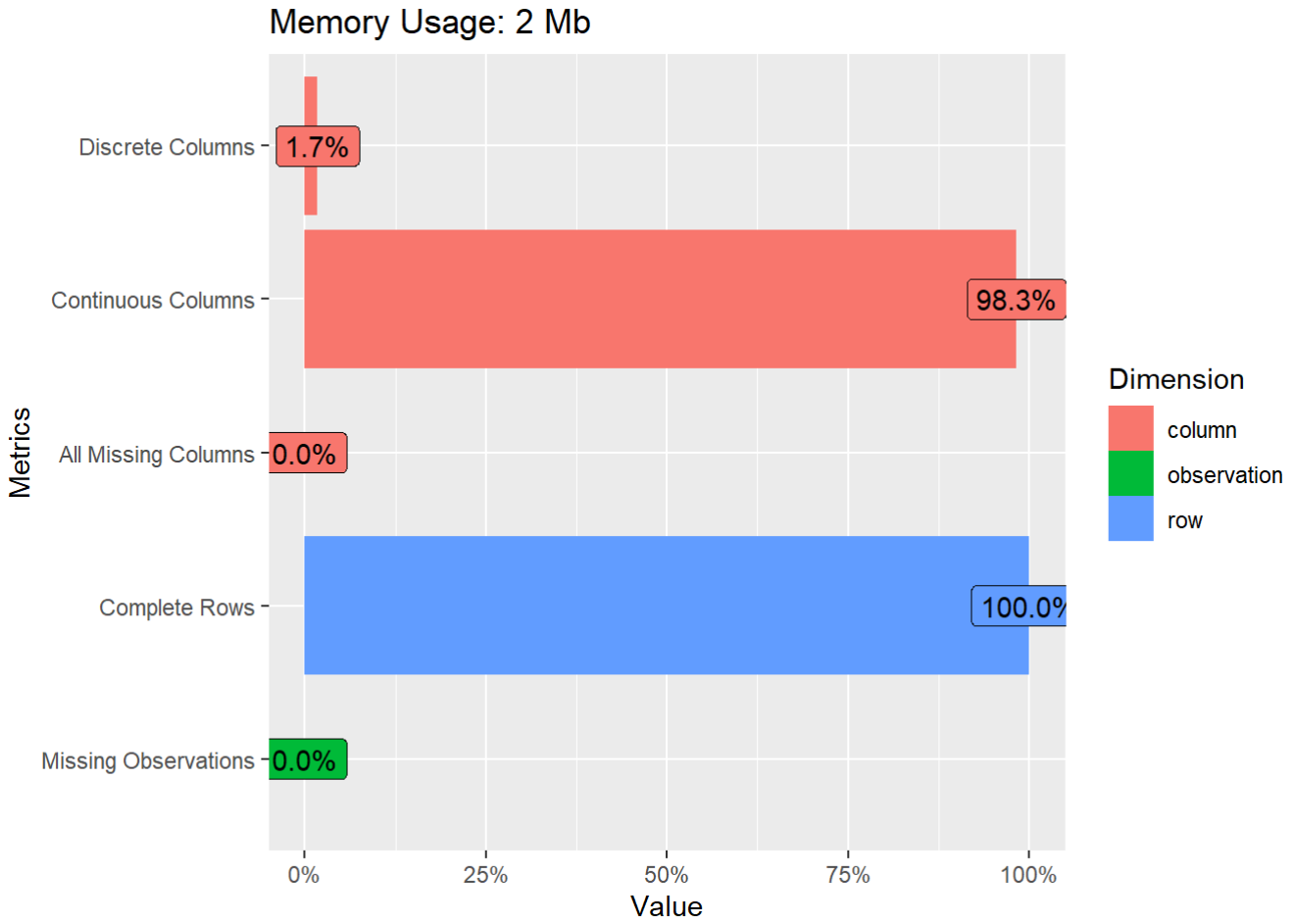
```
## 'data.frame':    4601 obs. of  58 variables:
##  $ make            : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ address         : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
##  $ all             : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
##  $ num3d           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ our             : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
##  $ over            : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
##  $ remove          : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
##  $ internet        : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
##  $ order           : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
##  $ mail            : num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
##  $ receive         : num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
##  $ will            : num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
##  $ people          : num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
##  $ report          : num  0 0.21 0 0 0 0 0 0 0 0 ...
##  $ addresses       : num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
##  $ free            : num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
##  $ business        : num  0 0.07 0.06 0 0 0 0 0 0 0 ...
##  $ email           : num  1.29 0.28 1.03 0 0 0 0.32 0 0.15 0.12 ...
##  $ you             : num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
##  $ credit          : num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
##  $ your            : num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
##  $ font            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num000          : num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
##  $ money           : num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
##  $ hp              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ hpl             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ george          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num650          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ lab             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ labs            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ telnet          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num857          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ data            : num  0 0 0 0 0 0 0 0 0.15 0 ...
##  $ num415          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num85           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ technology      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num1999         : num  0 0.07 0 0 0 0 0 0 0 0 ...
##  $ parts           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pm              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ direct          : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ cs              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ meeting         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ original        : num  0 0 0.12 0 0 0 0 0 0.3 0 ...
##  $ project         : num  0 0 0 0 0 0 0 0 0 0.06 ...
##  $ re              : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ edu             : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ table           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ conference      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charSemicolon   : num  0 0 0.01 0 0 0 0 0 0 0.04 ...
##  $ charRoundbracket : num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 0.271 0.03
```

```
...
##  $ charSquarebracket: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charExclamation  : num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0.244 ...
##  $ charDollar       : num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
##  $ charHash         : num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
##  $ capitalAve       : num  3.76 5.11 9.82 3.54 3.54 ...
##  $ capitalLong      : num  61 101 485 40 40 15 4 11 445 43 ...
##  $ capitalTotal     : num  278 1028 2259 191 191 ...
##  $ target           : Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

2.2. Análisis descriptivo (gráficos)

Reakizamos el análisis descriptivo mediante gráficas.

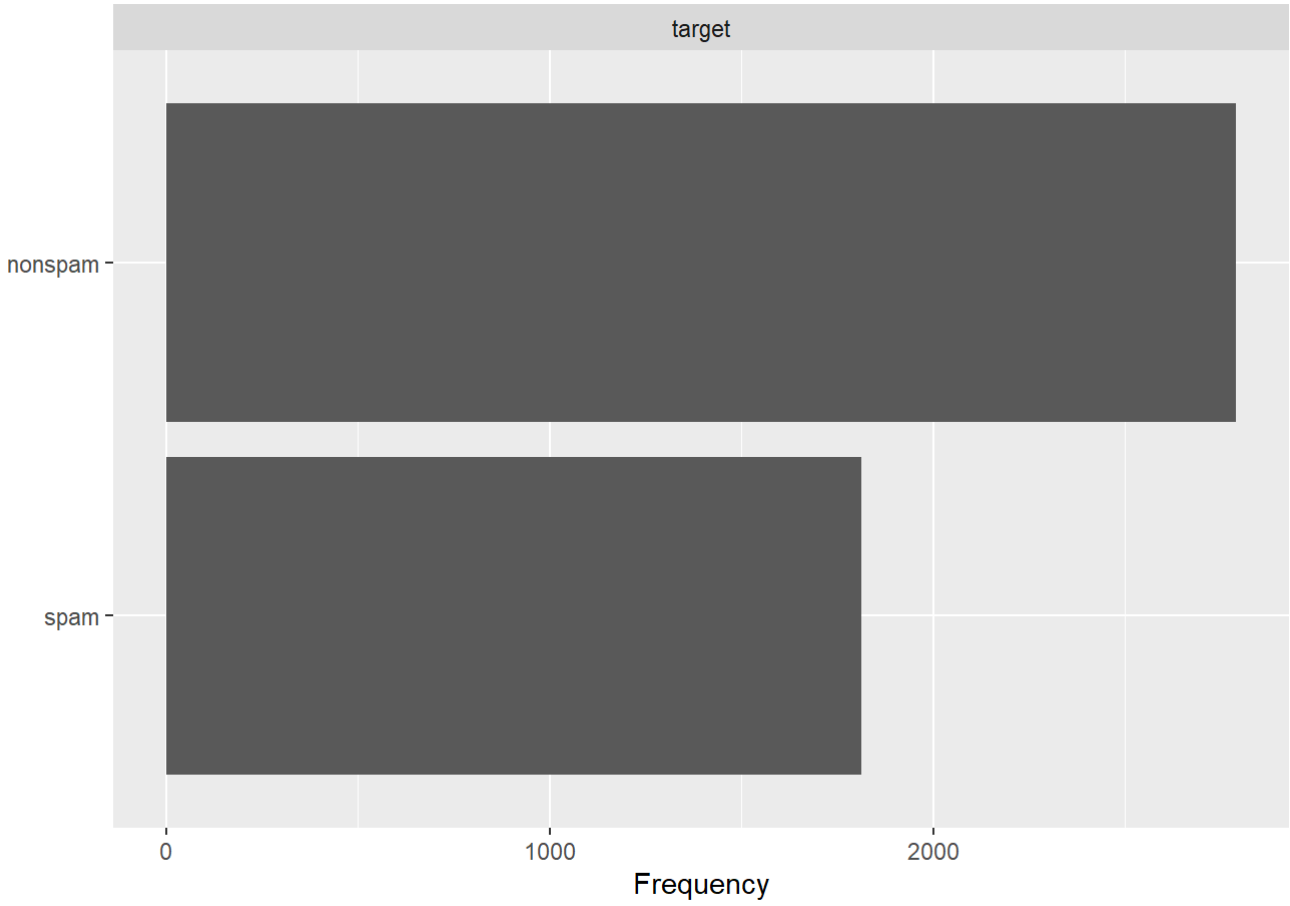
```
plot_intro(df) #gráfico para observar la distribución de variables y los casos missing por columnas, observaciones y filas
```



Como se ha trabajado previamente, no existen casos missing, por lo que podemos seguir el análisis descriptivo.

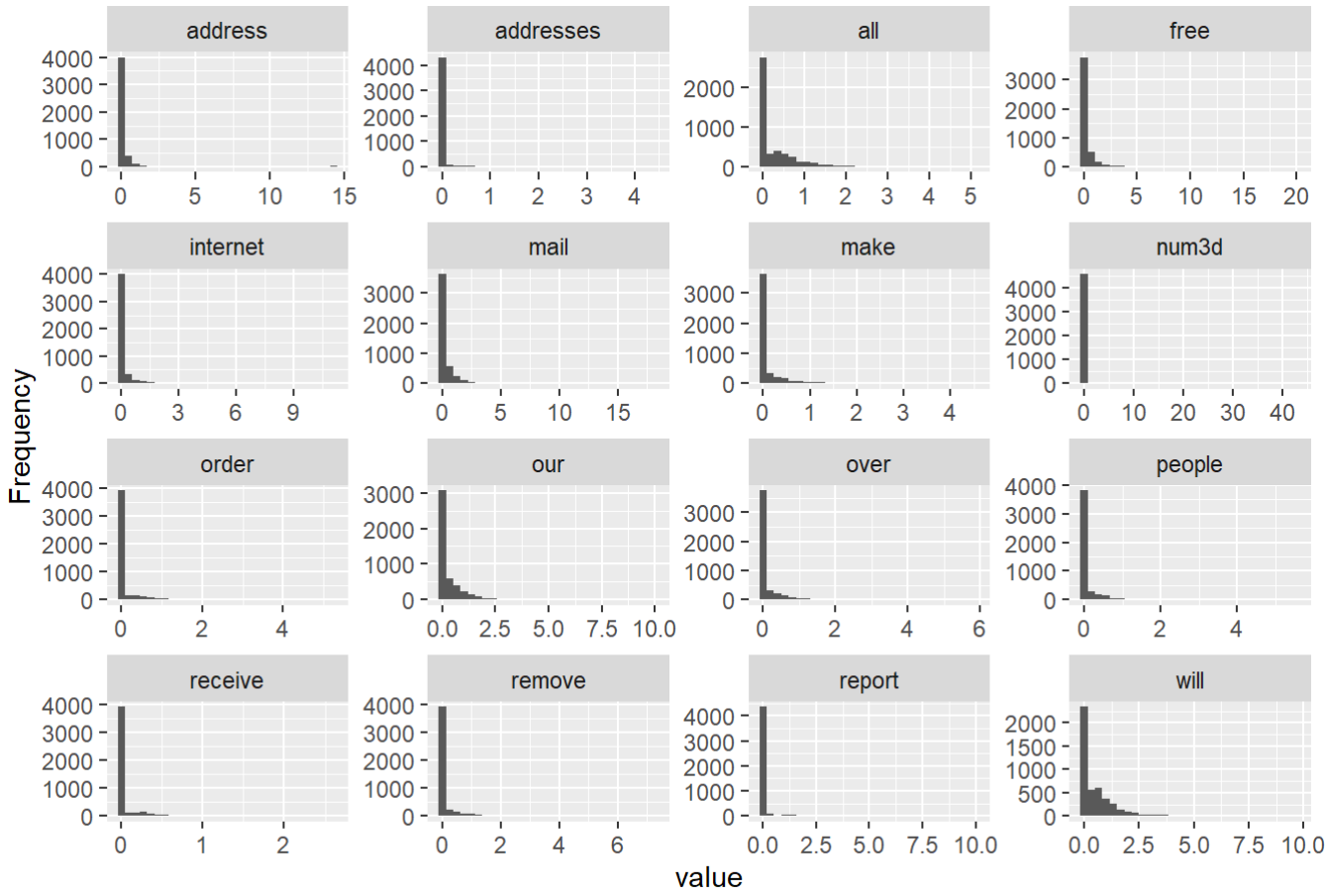
Sacamos un plot_bar para para variables categóricas.

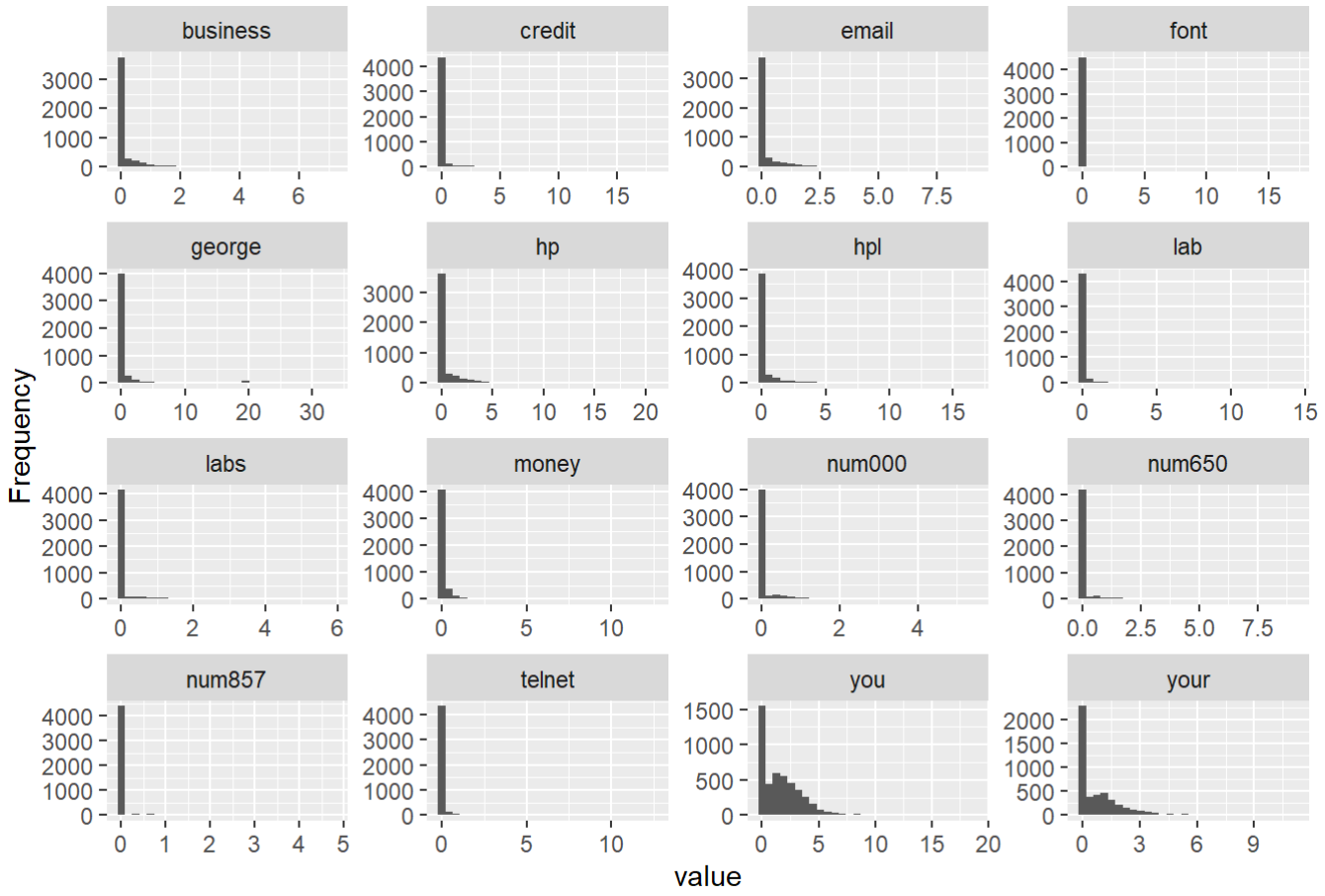
```
plot_bar(df) #gráfico para observar la distribución de frecuencias en variables categóricas
```

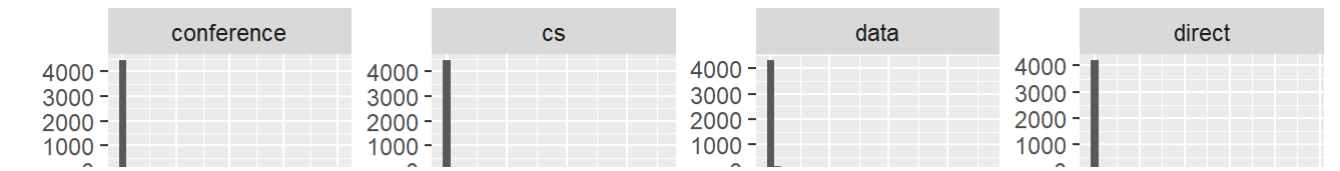


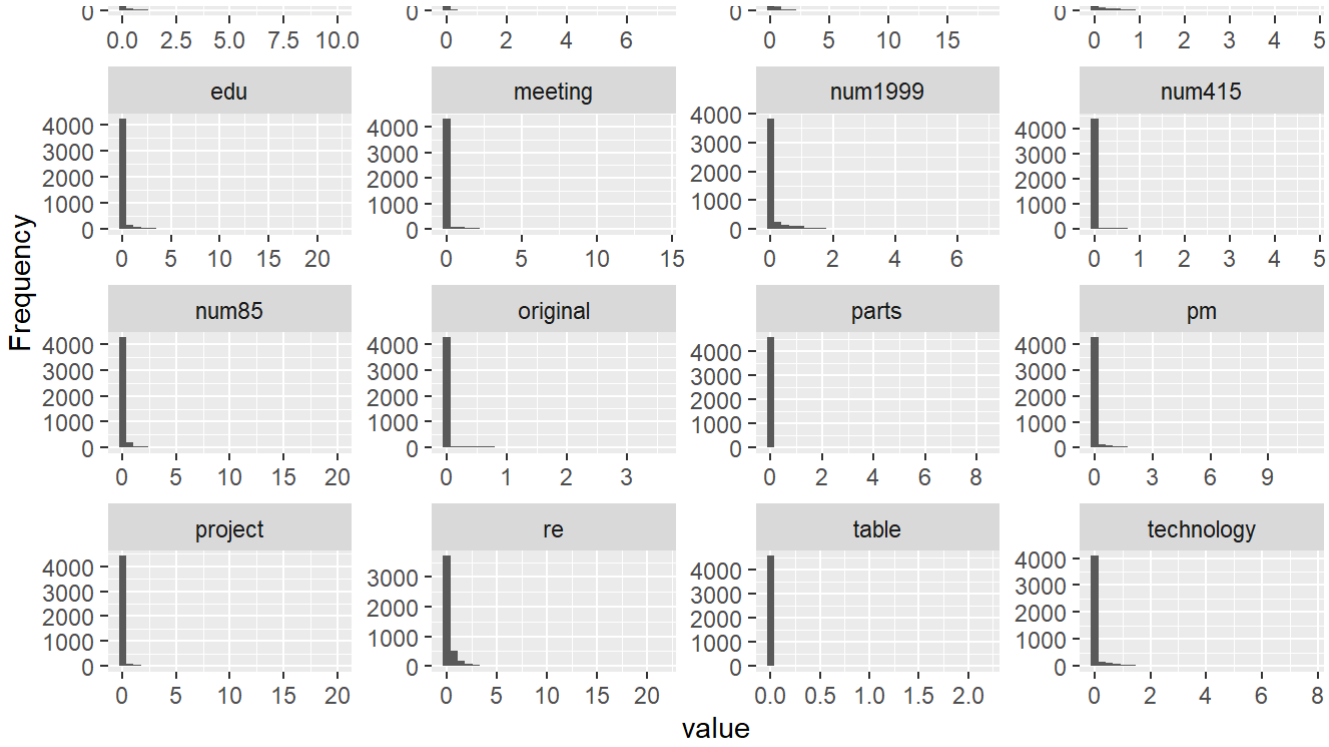
Sacamos un plot_histogram (para observar la distibución de las variables cuantitativas y un plot_correlation (para ver la relación entre ellas).

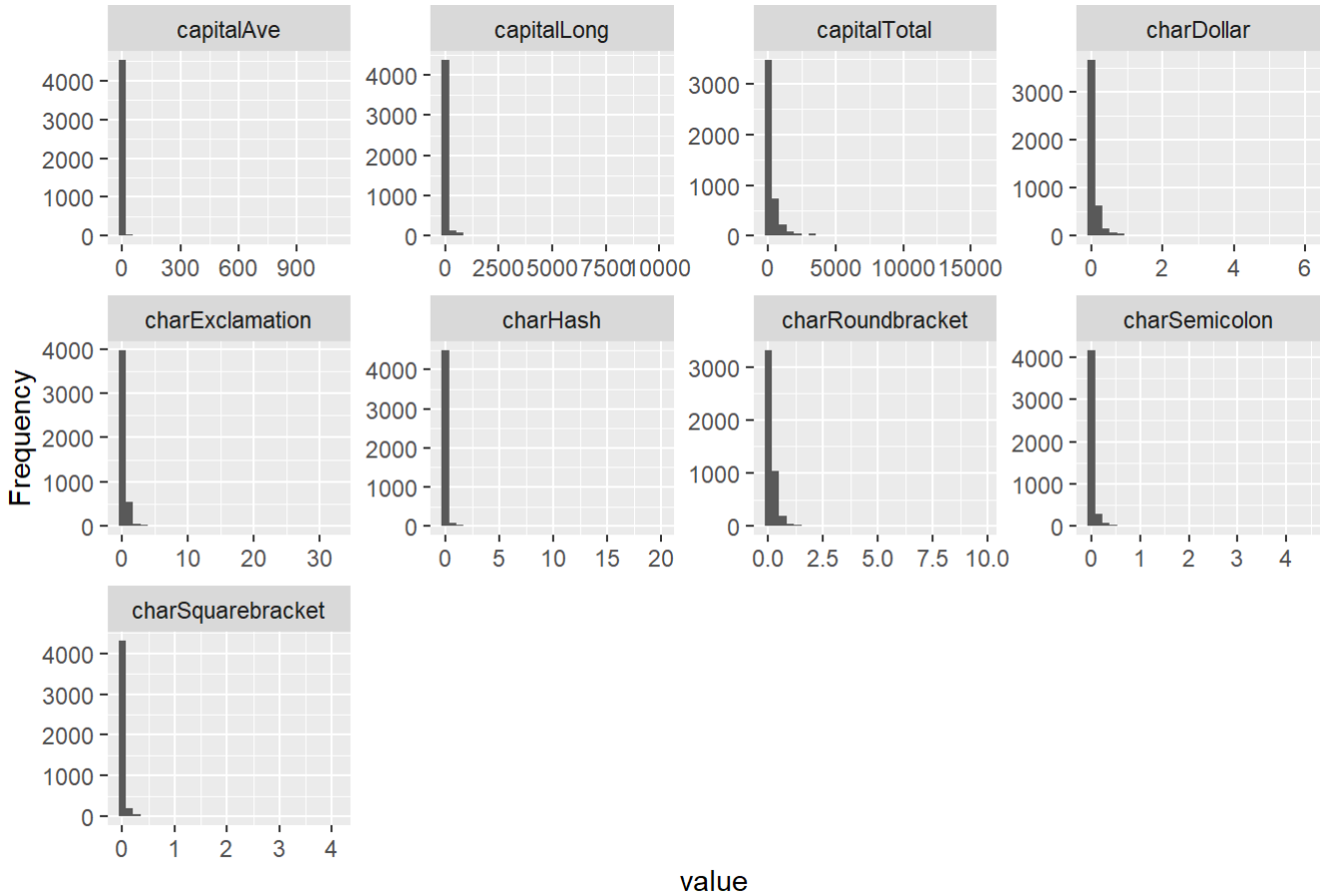
```
plot_histogram(df) #gráfico para observar la distribución de frecuencias en variables
 categóricas
```


address
addresses
all
free
internet
mail
make
num3d
order
our
over
people
receive
remove
report
will
Frequency
value

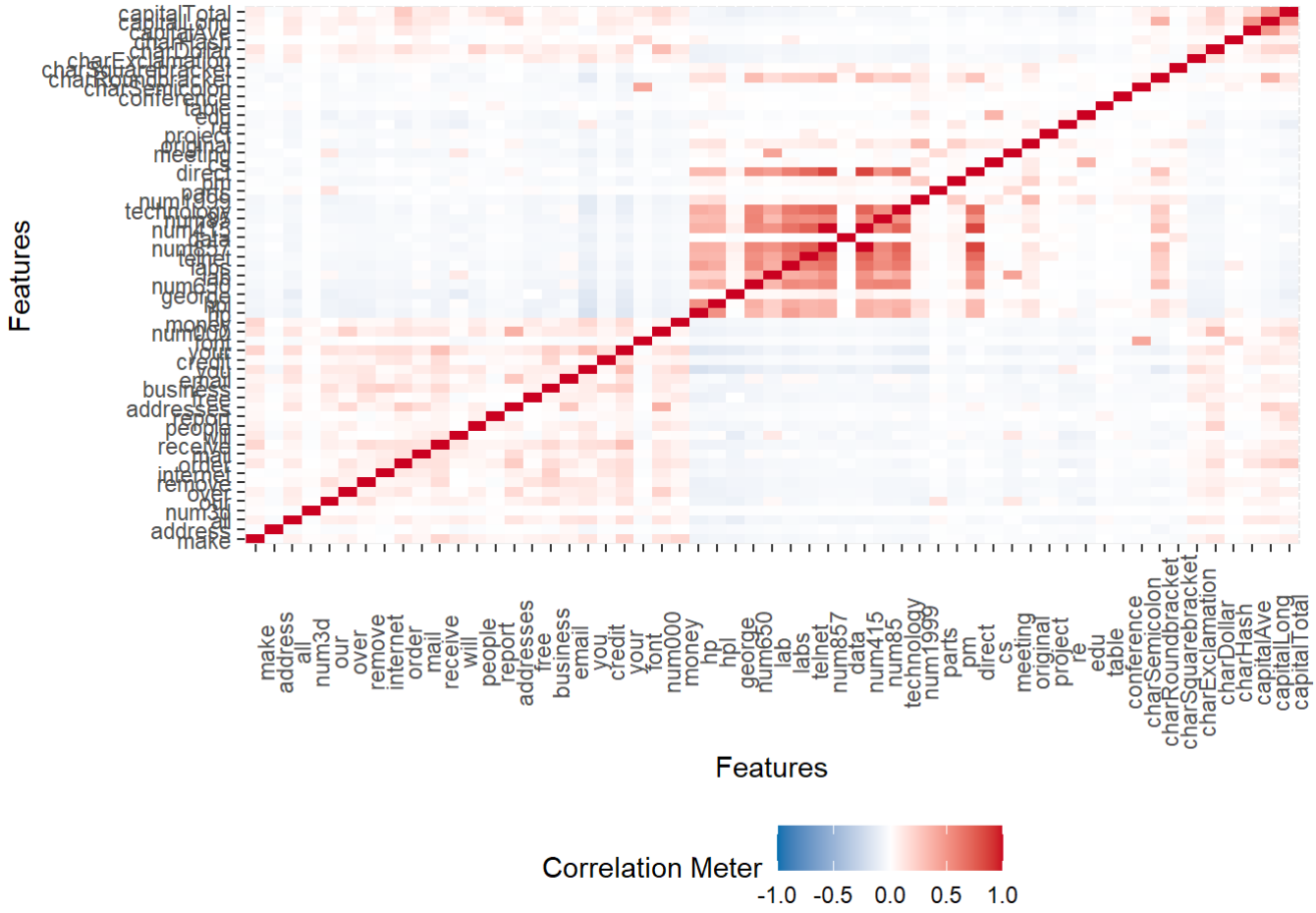

business
credit
email
font
george
hp
hpl
lab
labs
money
num000
num650
num857
telnet
you
your
Frequency
value


conference
cs
data
direct






```
plot_correlation(df[,-58])
```



3. Preparación de la modelización

3.1. Preselección de variables predictoras con Random Forest

Observamos el peso de las variables predictoras sobre nuestra target mediante Random forest.

```
set.seed(123)

pre_rf <- randomForest(formula = target ~., data= df, mtry=2,ntree=50, importance = T)

#CON LAS DOS SIGUIENTES INSTRUCCIONES SACAMOS SOLO EL ÍNDICE DE GINI (PERO SIN ORDENA
R)
imp_rf <- importance(pre_rf)[,4]
imp_rf
```

```
##             make          address              all             num3d
##        14.5629667       19.8490557       38.9004484        2.7719346
##              our             over           remove         internet
##        82.0037911       28.3572518      132.4836740       42.6552084
##            order             mail          receive             will
##        18.7327932       27.6393219       25.5413985       23.7294134
##           people           report        addresses             free
##         9.1835437        6.7876676        9.5457726       80.9486875
##         business            email              you           credit
##        40.3443240       21.9355778       38.8331970       21.1279211
##             your             font           num000            money
##        67.6786913        6.5260690       53.3779278       42.0822122
##               hp              hpl           george           num650
##        67.2746445       34.3836022       37.7667103       17.1827811
##              lab             labs           telnet           num857
##        18.4863353       11.5578928       11.0437187        2.3599920
##             data           num415            num85       technology
##        14.1179108        2.5742147       10.5171761       12.1713075
##           num1999            parts               pm           direct
##        33.5780682        1.2615477       10.6041689        4.5537482
##               cs          meeting          original          project
##         3.7776412       16.5781849        5.2818841        8.1193937
##               re              edu             table       conference
##        21.2665875       26.1674890        0.9705227        5.9174352
##     charSemicolon  charRoundbracket charSquarebracket   charExclamation
##         8.8882359       22.1587802        6.0578116      158.3537162
##        charDollar          charHash       capitalAve       capitalLong
##        88.7103547       18.5259367       88.1347395      103.4277952
##      capitalTotal
##        72.8824987
```

```
#A CONTINUACIÓN, VOY A ORDENAR LA SALIDA DE ÍNDICE DE GINI Y ORDENADO
imp_rf <- data.frame(VARIABLE = names(imp_rf), IMP_RF = imp_rf) #lo transformamos a da
taframe
imp_rf <- imp_rf %>% arrange(desc(IMP_RF)) %>% mutate(RANKING_RF = 1:nrow(imp_rf)) #cr
eamos el ranking
imp_rf
```

```
##             VARIABLE      IMP_RF RANKING_RF
## 1    charExclamation 158.3537162          1
## 2             remove 132.4836740          2
## 3        capitalLong 103.4277952          3
## 4         charDollar  88.7103547          4
## 5         capitalAve  88.1347395          5
## 6                our  82.0037911          6
## 7               free  80.9486875          7
## 8       capitalTotal  72.8824987          8
## 9               your  67.6786913          9
## 10                hp  67.2746445         10
## 11            num000  53.3779278         11
## 12          internet  42.6552084         12
## 13             money  42.0822122         13
## 14          business  40.3443240         14
## 15               all  38.9004484         15
## 16               you  38.8331970         16
## 17            george  37.7667103         17
## 18               hpl  34.3836022         18
## 19           num1999  33.5780682         19
## 20              over  28.3572518         20
## 21              mail  27.6393219         21
## 22               edu  26.1674890         22
## 23           receive  25.5413985         23
## 24              will  23.7294134         24
## 25  charRoundbracket  22.1587802         25
## 26             email  21.9355778         26
## 27                re  21.2665875         27
## 28            credit  21.1279211         28
## 29           address  19.8490557         29
## 30             order  18.7327932         30
## 31          charHash  18.5259367         31
## 32               lab  18.4863353         32
## 33            num650  17.1827811         33
## 34           meeting  16.5781849         34
## 35              make  14.5629667         35
## 36              data  14.1179108         36
## 37        technology  12.1713075         37
## 38              labs  11.5578928         38
## 39            telnet  11.0437187         39
## 40                pm  10.6041689         40
## 41             num85  10.5171761         41
## 42         addresses   9.5457726         42
## 43            people   9.1835437         43
## 44     charSemicolon   8.8882359         44
## 45           project   8.1193937         45
## 46            report   6.7876676         46
## 47              font   6.5260690         47
## 48 charSquarebracket   6.0578116         48
## 49        conference   5.9174352         49
## 50          original   5.2818841         50
```

```
## 51            direct   4.5537482         51
## 52                cs   3.7776412         52
## 53             num3d   2.7719346         53
## 54            num415   2.5742147         54
## 55            num857   2.3599920         55
## 56             parts   1.2615477         56
## 57             table   0.9705227         57
```

Se opta por dejar todas las variables para el análisis posterior.

3.2. Preparar funciones

Tomadas del curso de Machine Learning Predictivo (https://www.datascience4business.com/08_mlc-salespage-b) de DS4B) :

- Matriz de confusión
- Métricas
- Umbrales

Función para la matriz de confusión

En esta función se prepara la matriz de confusión (ver en otro post), donde se observa qué casos coinciden entre la puntuación real (obtenida por cada sujeto) y la puntuación predicha (“scoring”) por el modelo, estableciendo previmente un límite (“umbral”) para ello.

```
confusion<-function(real,scoring,umbral){
  conf<-table(real,scoring>=umbral)
  if(ncol(conf)==2) return(conf) else return(NULL)
}
```

Funcion para métricas de los modelos

Los indicadores a observar serán:

- Acierto (accuracy) = (TRUE POSITIVE + TRUE NEGATIVE) / TODA LA POBLACIÓN
- Precisión = TRUE POSITIVE / (TRUE POSITIVE + FALSE POSITIVE)
- Cobertura (recall, sensitivity) = TRUE POSITIVE / (TRUE POSITIVE + FALSE NEGATIVE)
- F1 = 2* (precisión * cobertura) (precisión + cobertura)

```
metricas<-function(matriz_conf){
  acierto <- (matriz_conf[1,1] + matriz_conf[2,2]) / sum(matriz_conf) *100
  precision <- matriz_conf[2,2] / (matriz_conf[2,2] + matriz_conf[1,2]) *100
  cobertura <- matriz_conf[2,2] / (matriz_conf[2,2] + matriz_conf[2,1]) *100
  F1 <- 2*precision*cobertura/(precision+cobertura)
  salida<-c(acierto,precision,cobertura,F1)
  return(salida)
}
```

Función para probar distintos umbrales

Con esta función se analiza el efecto que tienen distintos umbrales sobre los indicadores de la matriz de confusión (precisión y cobertura). Lo que buscaremnos será aquél que maximice la relación entre cobertura y precisión (F1).

```
umbrales<-function(real,scoring){
  umbrales<-data.frame(umbral=rep(0,times=19),acierto=rep(0,times=19),precision=rep(0,
times=19),cobertura=rep(0,times=19),F1=rep(0,times=19))
  cont <- 1
  for (cada in seq(0.05,0.95,by = 0.05)){
    datos<-metricas(confusion(real,scoring,cada))
    registro<-c(cada,datos)
    umbrales[cont,]<-registro
    cont <- cont + 1
  }
  return(umbrales)
}
```

3.3. Particiones de training (70%) y test (30%)

Se segmenta la muestra en dos partes (train y test) empleando el programa Caret.

1. Training o entrenamiento (70% de la muestra): servirá para entrenar al modelo de clasificación.
2. Test (30%): servirá para validar el modelo. La característica fundamental es que esta muestra no debe haber tenido contacto previamente con el funcionamiento del modelo.

```
set.seed(100)  # Para reproducir los mismos resultados
partition <- createDataPartition(y = df$target, p = 0.7, list = FALSE)
train <- df[partition,]
test <- df[-partition,]
```

```
#Distribución de la variable TARGET
table(train$target)
```

```
##
## nonspam    spam
##    1952    1270
```

```
table(test$target)
```

```
##
## nonspam    spam
##     836     543
```

4. Modelización con Árboles de decisión

Paso 1. Primer modelo

```
mod_dt<-rpart(target~., train,
          method = 'class',
          parms = list(  split = "information"),
          control = rpart.control(cp = 0.00001))
#Lanzamos la función "rpart2" para hallar el AD, que incluye>:
#El modelo a entrenar: target ~.
#El df: "train"
#method = 'class'. Método de clasificación
#split: criterio de corte. Se tienen dos opciones: 'information' o 'gini'
#cp: criterio de complejidad. Se comienza emplenado un cp muy pequeño (esto es, dejamo
s que el árbol sea muy profundo), y posteriormente, se irán añadiendo condiciones más
 restrictivas (siendo el árbol menos profundo y más interpretable).

# Imprimimos los resultados
printcp(mod_dt)
```

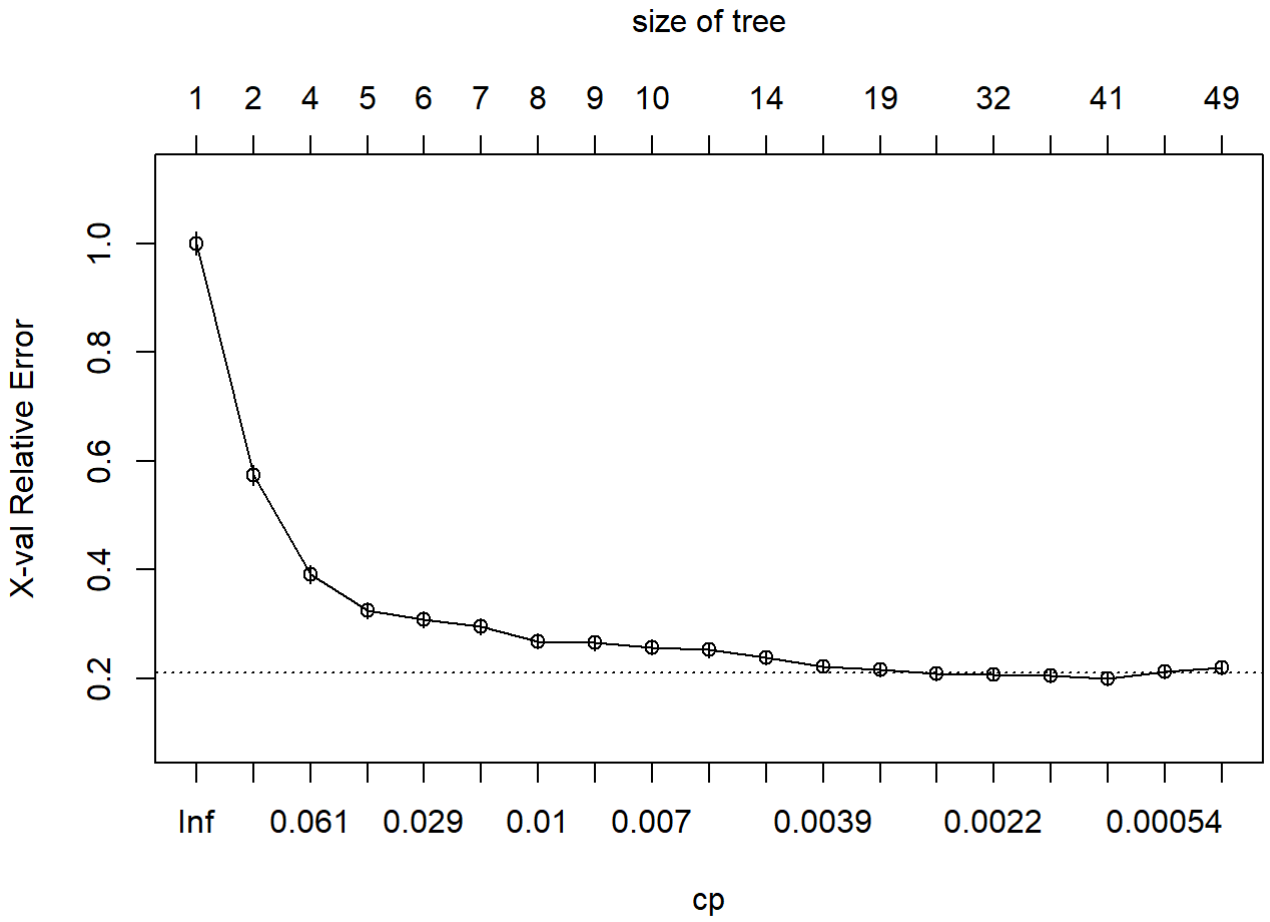
```
##
## Classification tree:
## rpart(formula = target ~ ., data = train, method = "class", parms = list(split = "i
nformation"),
##     control = rpart.control(cp = 0.00001))
##
## Variables actually used in tree construction:
##  [1] all              business         capitalAve       capitalLong
##  [5] capitalTotal     charDollar       charExclamation  charRoundbracket
##  [9] charSemicolon    edu              free             george
## [13] hp               internet         money            num1999
## [17] num650           our              re               remove
## [21] will             you              your
##
## Root node error: 1270/3222 = 0.39417
##
## n= 3222
##
##            CP nsplit rel error  xerror     xstd
## 1  0.47716535      0   1.00000 1.00000 0.021841
## 2  0.08267717      1   0.52283 0.57323 0.018692
## 3  0.04566929      3   0.35748 0.39134 0.016143
## 4  0.03228346      4   0.31181 0.32520 0.014941
## 5  0.02677165      5   0.27953 0.30787 0.014595
## 6  0.01181102      6   0.25276 0.29606 0.014350
## 7  0.00866142      7   0.24094 0.26850 0.013749
## 8  0.00787402      8   0.23228 0.26535 0.013678
## 9  0.00629921      9   0.22441 0.25748 0.013497
## 10 0.00551181     11   0.21181 0.25276 0.013386
## 11 0.00492126     13   0.20079 0.23780 0.013027
## 12 0.00314961     17   0.18110 0.22283 0.012651
## 13 0.00262467     18   0.17795 0.21575 0.012467
## 14 0.00251969     21   0.17008 0.20866 0.012280
## 15 0.00196850     31   0.14016 0.20787 0.012258
## 16 0.00157480     33   0.13622 0.20472 0.012173
## 17 0.00110236     40   0.12520 0.19921 0.012023
## 18 0.00026247     45   0.11969 0.21260 0.012384
## 19 0.00001000     48   0.11890 0.21969 0.012570
```

Interpretación de la tabla:

a. Observamos la columna xerror (error de validación cruzada), la cual se va reduciendo hasta que empieza a crecer.

b. Tomamos el datos de CP (criterior de complejidad) de ese nivel para incluirlo posteriormente al modelo.

```
plotcp(mod_dt)
```



Se observa que el error de validación cruzada (xerror = 0.19921) minimiza en un cp = 0.00110236 de complejidad.Por tanto, los dos siguientes pasos son:

a. Generamos un nuevo árbol con ese parámetro, esto es más restrictivo.

b. Además, le añadimos otro parámetro de restricción: el número de niveles, que no tenga mas de 7 niveles de profundidad (maxdepth = 7), aunque no sabemos dónde va a parar antes, por el cp o por el número de niveles.

Paso 2. Segundo modelo

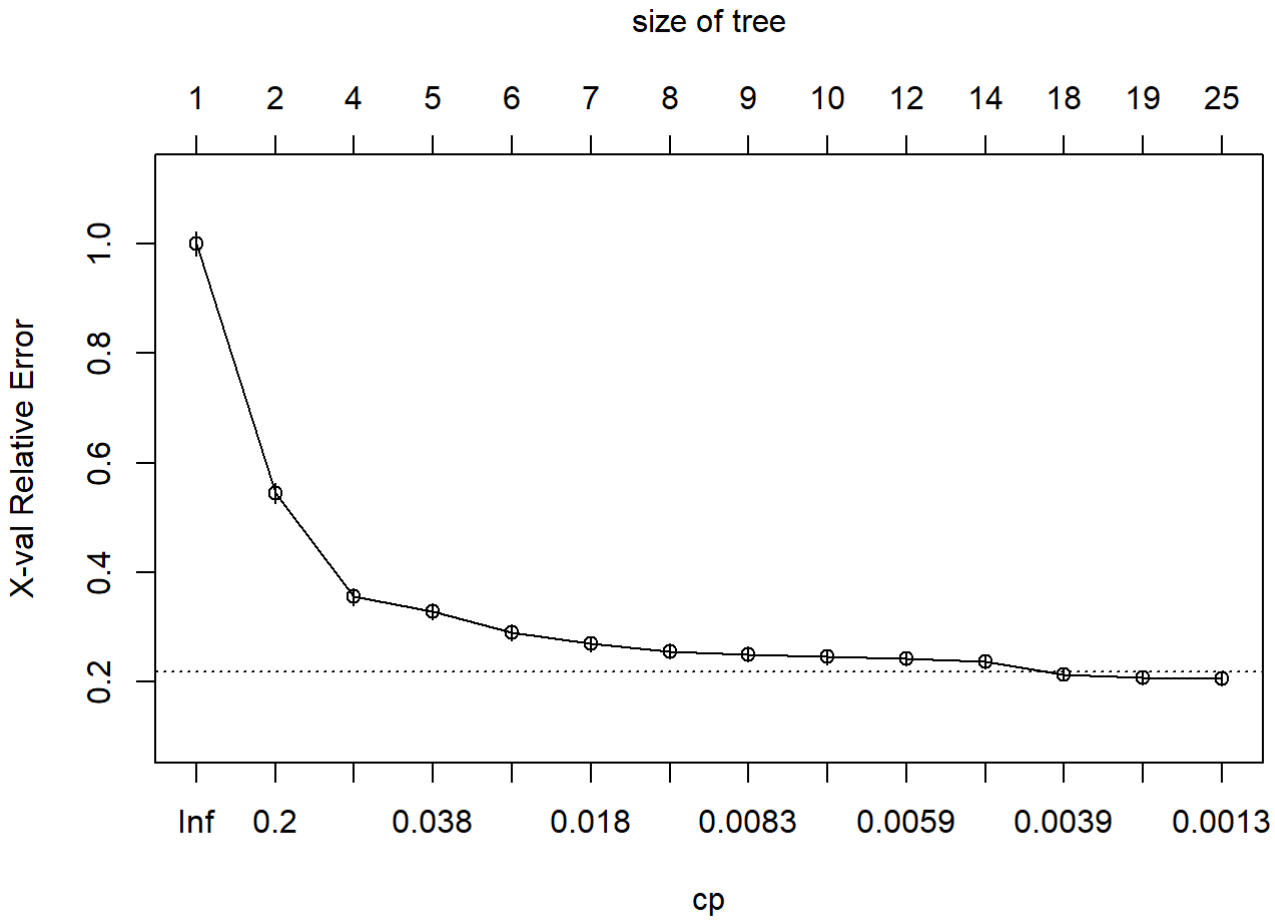
```
mod_dt2<-rpart(target~., train,
          method = 'class',
          parms = list(  split = "information"),
          control = rpart.control(cp = 0.00110236,
                                  maxdepth = 7))

printcp(mod_dt2)
```

```
##
## Classification tree:
## rpart(formula = target ~ ., data = train, method = "class", parms = list(split = "i
nformation"),
##     control = rpart.control(cp = 0.00110236, maxdepth = 7))
##
## Variables actually used in tree construction:
##  [1] business        capitalAve      capitalLong     charDollar
##  [5] charExclamation edu             free            george
##  [9] hp              num1999         our             remove
## [13] you             your
##
## Root node error: 1270/3222 = 0.39417
##
## n= 3222
##
##           CP nsplit rel error  xerror     xstd
## 1  0.4771654      0   1.00000 1.00000 0.021841
## 2  0.0826772      1   0.52283 0.54409 0.018345
## 3  0.0456693      3   0.35748 0.35591 0.015522
## 4  0.0322835      4   0.31181 0.32835 0.015003
## 5  0.0267717      5   0.27953 0.28976 0.014216
## 6  0.0118110      6   0.25276 0.26929 0.013767
## 7  0.0086614      7   0.24094 0.25591 0.013460
## 8  0.0078740      8   0.23228 0.25039 0.013330
## 9  0.0062992      9   0.22441 0.24567 0.013218
## 10 0.0055118     11   0.21181 0.24252 0.013142
## 11 0.0049213     13   0.20079 0.23701 0.013007
## 12 0.0031496     17   0.18110 0.21417 0.012426
## 13 0.0015748     18   0.17795 0.20709 0.012237
## 14 0.0011024     24   0.16850 0.20630 0.012216
```

Se observa que xerror no asciende.

```
#Observamos gráficamente el resultado anterior
plotcp(mod_dt2)
```



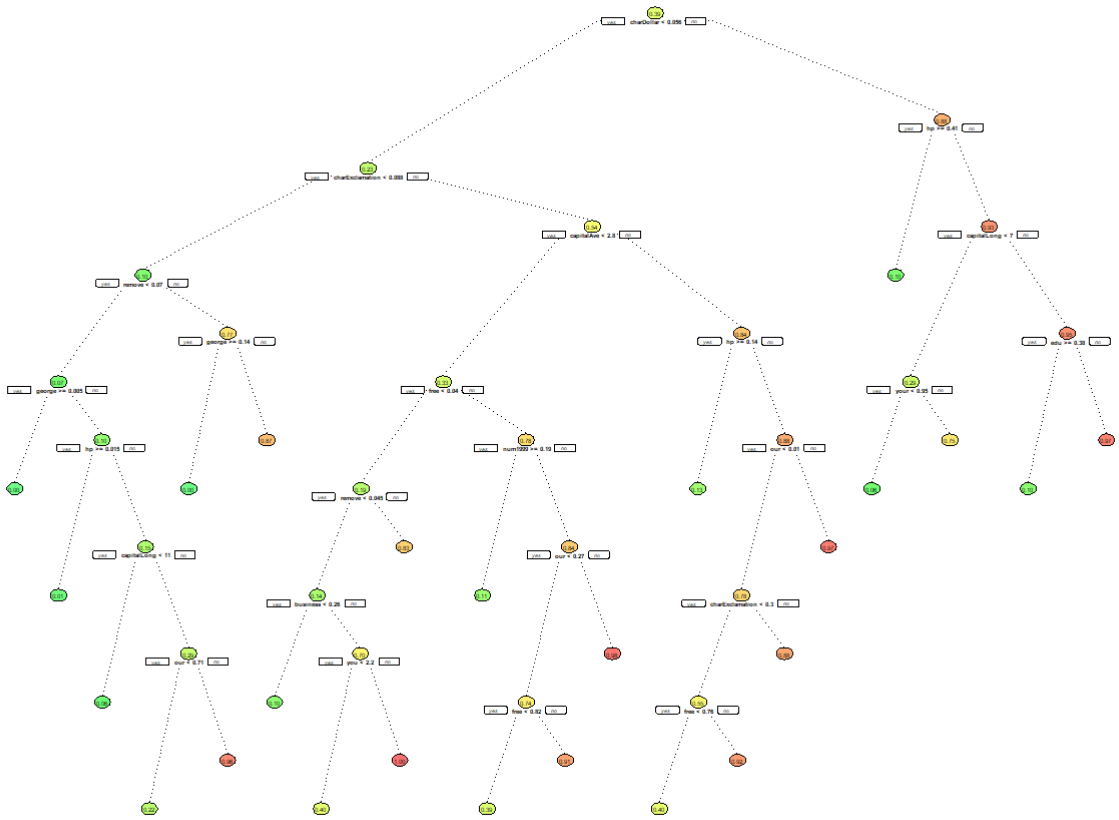
Se observa que xerror no asciende. Parece que el árbol es bastante estable, por lo que pasamos a interpretarlo.

Paso 3. Interpretación del árbol

Seguimos tres pasos:

A. Creación del gráfico del árbol

```
rpart.plot(mod_dt2,type=2,extra = 7, under = TRUE,under.cex = 0.7,fallen.leaves=F,gap
 = 0,cex=0.2, yesno = 2,box.palette = "GnYlRd",branch.lty = 3)
```



En muchas ocasiones el árbol no es fácilmente visible si no se amplía considerablemente, lo cual podemos hacerlo exportándolo a pdf o en un power point.

B. Reglas del árbol

Sacamos las reglas de división del árbol, necesarias para hacer una implantación de negocio posterior

```
rpart.rules(mod_dt2,style = 'tall',cover = T)
```

```
## target is 0.00 with cover 15% when
##     charDollar < 0.056
##     charExclamation < 0.088
##     remove < 0.070
##     george >= 0.005
##
## target is 0.00 with cover 0% when
##     charDollar < 0.056
##     charExclamation < 0.088
##     remove >= 0.070
##     george >= 0.140
##
## target is 0.01 with cover 12% when
##     charDollar < 0.056
##     charExclamation < 0.088
##     hp >= 0.015
##     remove < 0.070
##     george < 0.005
##
## target is 0.06 with cover 15% when
##     charDollar < 0.056
##     charExclamation < 0.088
##     hp < 0.015
##     remove < 0.070
##     george < 0.005
##     capitalLong < 11
##
## target is 0.06 with cover 0% when
##     charDollar >= 0.056
##     hp < 0.405
##     capitalLong < 7
##     your < 0.95
##
## target is 0.10 with cover 2% when
##     charDollar >= 0.056
##     hp >= 0.405
##
## target is 0.10 with cover 0% when
##     charDollar >= 0.056
##     hp < 0.405
##     capitalLong >= 7
##     edu >= 0.38
##
## target is 0.10 with cover 9% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     remove < 0.045
##     free < 0.04
##     business < 0.26
##
```

```
## target is 0.11 with cover 0% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     free >= 0.04
##     num1999 >= 0.19
##
## target is 0.13 with cover 0% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     hp >= 0.135
##     capitalAve >= 2.8
##
## target is 0.22 with cover 8% when
##     charDollar < 0.056
##     charExclamation < 0.088
##     hp < 0.015
##     remove < 0.070
##     our < 0.71
##     george < 0.005
##     capitalLong >= 11
##
## target is 0.39 with cover 1% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     free is 0.04 to 0.82
##     our < 0.27
##     num1999 < 0.19
##
## target is 0.40 with cover 0% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     remove < 0.045
##     free < 0.04
##     business >= 0.26
##     you < 2.2
##
## target is 0.40 with cover 1% when
##     charDollar < 0.056
##     charExclamation is 0.088 to 0.297
##     hp < 0.135
##     capitalAve >= 2.8
##     free < 0.76
##     our < 0.01
##
## target is 0.75 with cover 0% when
##     charDollar >= 0.056
##     hp < 0.405
##     capitalLong < 7
##     your >= 0.95
```

```
##
## target is 0.83 with cover 1% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     remove >= 0.045
##     free < 0.04
##
## target is 0.87 with cover 2% when
##     charDollar < 0.056
##     charExclamation < 0.088
##     remove >= 0.070
##     george < 0.140
##
## target is 0.88 with cover 3% when
##     charDollar < 0.056
##     charExclamation >= 0.297
##     hp < 0.135
##     capitalAve >= 2.8
##     our < 0.01
##
## target is 0.91 with cover 1% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     free >= 0.82
##     our < 0.27
##     num1999 < 0.19
##
## target is 0.92 with cover 0% when
##     charDollar < 0.056
##     charExclamation is 0.088 to 0.297
##     hp < 0.135
##     capitalAve >= 2.8
##     free >= 0.76
##     our < 0.01
##
## target is 0.96 with cover 1% when
##     charDollar < 0.056
##     charExclamation < 0.088
##     hp < 0.015
##     remove < 0.070
##     our >= 0.71
##     george < 0.005
##     capitalLong >= 11
##
## target is 0.97 with cover 22% when
##     charDollar >= 0.056
##     hp < 0.405
##     capitalLong >= 7
##     edu < 0.38
##
```

```
## target is 0.97 with cover 5% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     hp < 0.135
##     capitalAve >= 2.8
##     our >= 0.01
##
## target is 0.98 with cover 1% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     free >= 0.04
##     our >= 0.27
##     num1999 < 0.19
##
## target is 1.00 with cover 0% when
##     charDollar < 0.056
##     charExclamation >= 0.088
##     capitalAve < 2.8
##     remove < 0.045
##     free < 0.04
##     business >= 0.26
##     you >= 2.2
```

```
#style sirve para que la salida sea mas legible y cover añade el % de casos e los que
 aplica la regla
```

C. Introducir datos en un df

Llevamos el nodo final de cada cliente a un data.frame para poder hacer una explotacion posterior (por ejemplo para saber las características de cada nodo, como edad, etc.)

```
#Se usa el predict específico de rpart y con el parámetro nn
ar2_numnodos<-rpart.predict(mod_dt2, test, nn = T)

head(ar2_numnodos)
```

```
##       nonspam      spam nn
## 1  0.03225806 0.9677419 47
## 11 0.17391304 0.8260870 41
## 12 0.17391304 0.8260870 41
## 15 0.02380952 0.9761905 87
## 16 0.03366059 0.9663394 31
## 24 0.02380952 0.9761905 87
```

INTERPRETACIÓN

El mail 1 tiene una probabilidad de ser spam del 96,77% y cae en el nodo 47.

El mail 11 tiene una probabilidad de ser spam del 82,26% y cae en el nodo 41.

Paso 4. Predict y matriz de confusión

```
dt_score_class <- predict(mod_dt2, test, type="class")

MC_dt <- confusionMatrix(dt_score_class, test$target , positive = 'nonspam')
MC_dt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction nonspam spam
##    nonspam     799   81
##    spam         37  462
##
##                Accuracy : 0.9144
##                  95% CI : (0.8984, 0.9287)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.8182
##
##  Mcnemar's Test P-Value : 0.00007543
##
##             Sensitivity : 0.9557
##             Specificity : 0.8508
##          Pos Pred Value : 0.9080
##          Neg Pred Value : 0.9259
##              Prevalence : 0.6062
##          Detection Rate : 0.5794
##    Detection Prevalence : 0.6381
##       Balanced Accuracy : 0.9033
##
##        'Positive' Class : nonspam
##
```

Se observa que el modelo se ajusta muy bien a los datos, obteniendo unos indicaodres muy altos, por ejemplo el Accuracy = 0.9144.

Paso 5. Predict con probabilidades y umbrales

1º) En este paso sacamos la probabilidad de cada mail a ser spam o no spam.

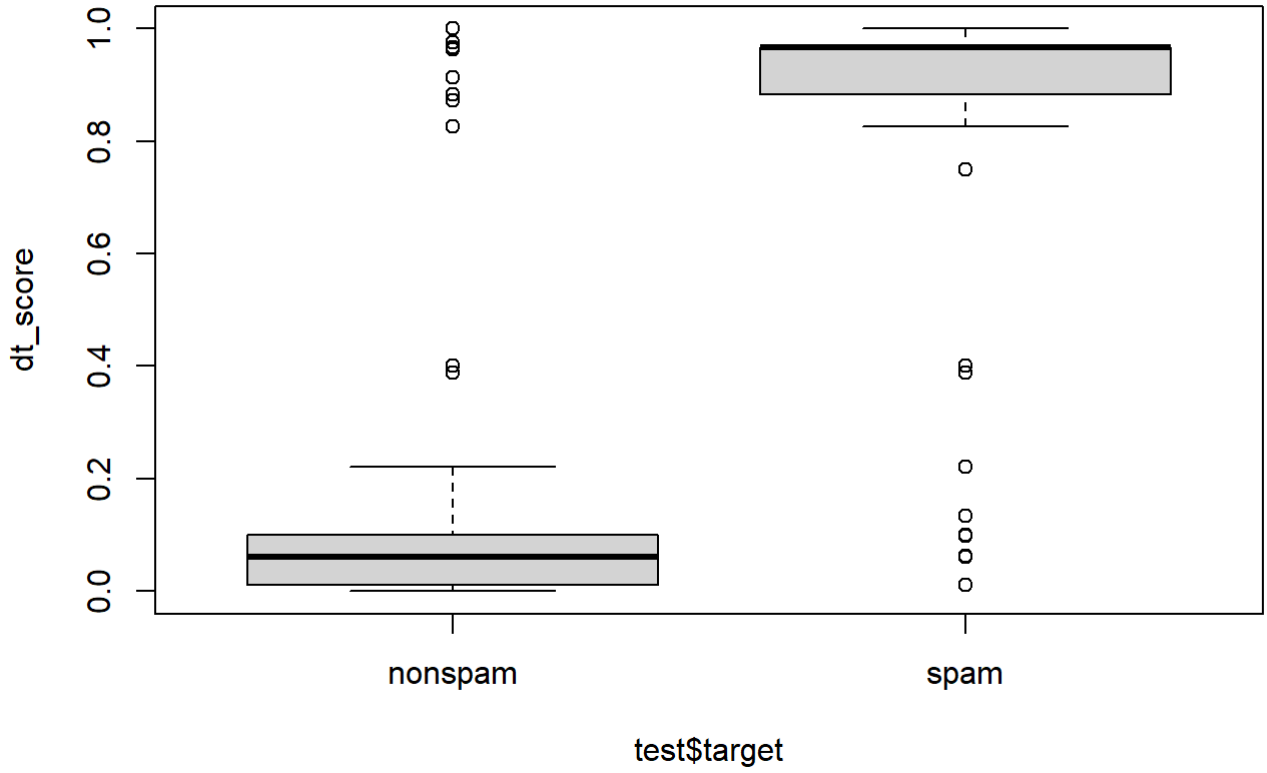
```
dt_score<-predict(mod_dt2,test,type = 'prob')[,2]
#Samos el predict para el modelo "ar2":
#Con el data frame "test"
#Se utiliza "type=prob" (que reporta la probabilidad para cada caso)
#Se le incluye [,2], es decir, la segunda columna, ya que interesa la probabilidad de
 que sea spam.

#Sacamos los 6 primeros casos
head(dt_score)
```

```
##         1        11        12        15        16        24
## 0.9677419 0.8260870 0.8260870 0.9761905 0.9663394 0.9761905
```

Vemos la capacidad de discriminación entre los dos niveles de la TARGET. Lógicamente, después de observar los indicadores de la matriz de cnfusión, la capacidad discriminativa del modelo es muy elevada.

```
plot(dt_score ~ test$target)
```



2º) Ahora transformamos la probabilidad obtenida en una decisión binaria de Sí es spam, o No es spam.

Con la función umbrales probamos diferentes cortes

```
umb_dt<-umbrales(test$target,dt_score)
umb_dt
```

```
##    umbral  acierto precision cobertura       F1
## 1    0.05 64.17694  52.37633  99.44751 68.61499
## 2    0.10 79.98550  67.22581  95.94843 79.05918
## 3    0.15 87.45468  78.99687  92.81768 85.35140
## 4    0.20 87.45468  78.99687  92.81768 85.35140
## 5    0.25 91.80566  91.03053  87.84530 89.40956
## 6    0.30 91.80566  91.03053  87.84530 89.40956
## 7    0.35 91.80566  91.03053  87.84530 89.40956
## 8    0.40 91.44307  92.58517  85.08287 88.67562
## 9    0.45 91.44307  92.58517  85.08287 88.67562
## 10   0.50 91.44307  92.58517  85.08287 88.67562
## 11   0.55 91.44307  92.58517  85.08287 88.67562
## 12   0.60 91.44307  92.58517  85.08287 88.67562
## 13   0.65 91.44307  92.58517  85.08287 88.67562
## 14   0.70 91.44307  92.58517  85.08287 88.67562
## 15   0.75 91.37056  92.57028  84.89871 88.56868
## 16   0.80 91.37056  92.57028  84.89871 88.56868
## 17   0.85 90.21030  92.50000  81.76796 86.80352
## 18   0.90 87.09210  93.97590  71.82320 81.41962
## 19   0.95 86.87455  94.58128  70.71823 80.92729
```

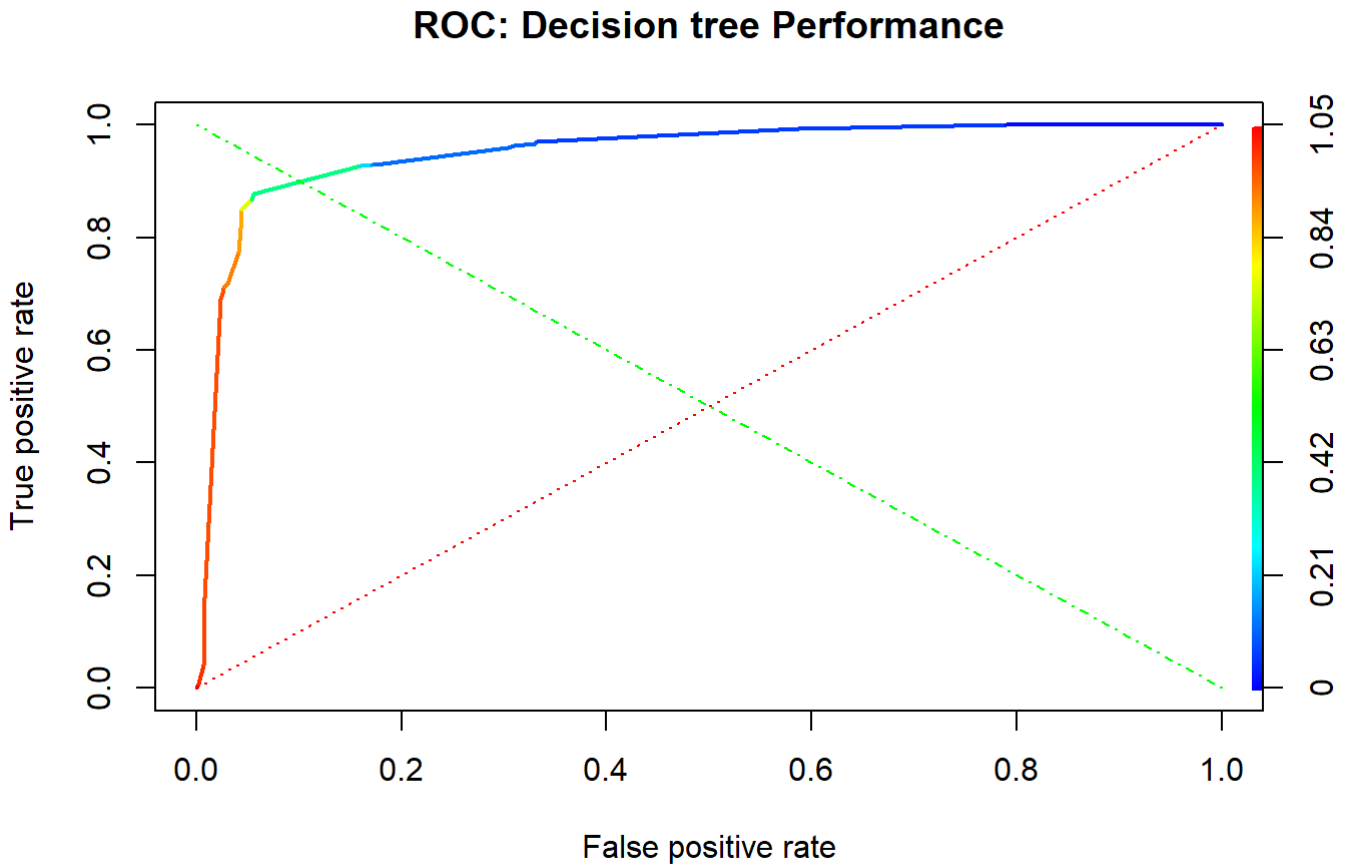
Seleccionamos el umbral que maximiza la F1 (cuando empieza a decaer)

```
umbfinal_dt<-umb_dt[which.max(umb_dt$F1),1]
umbfinal_dt
```

```
## [1] 0.25
```

Paso 6. Curva ROC

```
pred_dt <- prediction(dt_score, test$target)
perf_dt <- performance(pred_dt,"tpr","fpr")
#library(ROCR)
plot(perf_dt, lwd=2, colorize=TRUE, main="ROC: Decision tree Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



En la curva ROC, la línea diagonal que divide el gráfico en dos partes iguales indica que el modelo no tiene ninguna capacidad predictiva. Todo el área que está por encima de esa diagonal hasta la curva, indica la capacidad predictiva del modelo.

Paso 7. Métricas definitivas

Sacamos las métricas definitivas incluyendo el AUC

```
#Matriz de confusión con umbral final
score <- ifelse(dt_score > umbfinal_dt, "spam", "nonspam")
MC <- table(test$target, score)
dt_Acc <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
dt_Sen <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
dt_Pr <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
dt_F1 <- round(2*dt_Pr*dt_Sen/(dt_Pr+dt_Sen), 2)
dt_AUROC <- round(performance(pred_dt, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acierto_ad: ",dt_Acc,"\tSensibilidad_ad: ", dt_Sen, "\tPrecision_ad:", dt_Pr, "\t
F1_ad:", dt_F1, "\tAUROC_ad: ",dt_AUROC)
```

```
## Acierto_ad:  91.81   Sensibilidad_ad:  91.03     Precision_ad: 87.85     F1_ad: 89.
41    AUROC_ad:  95.19
```

Obtenemos las métricas definitivas añadiendo la métrica AUC, que indica el porcentaje de predicción del modelo, un 95.19%, lo que indica que es un modelo extremadamente bueno.

5. Modelización con bagged trees

Paso 1. Entrenamiento del modelo

```
set.seed(123)

# Train a bagged model
mod_bt <- bagging(target ~ ., train,
                          coob = TRUE)

# Print the model
print(mod_bt)
```

```
##
## Bagging classification trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = target ~ ., data = train, coob = TRUE)
##
## Out-of-bag estimate of misclassification error:  0.0618
```

Vamos a emplear el método OOB “out-of-bag” (que lleva menos tiempo que con CARET mediante validación cruzada), para ello se emplea el parámetro “coob = TRUE”.

Paso 2. Predict y matriz de confusión

```
bt_score_class <- predict(mod_bt, test, type="class") # return classification labels

MC_bt <- confusionMatrix(bt_score_class, test$target , positive = 'nonspam')
MC_bt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction nonspam spam
##    nonspam     806   51
##    spam         30  492
##
##                Accuracy : 0.9413
##                  95% CI : (0.9275, 0.9531)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.8761
##
##  Mcnemar's Test P-Value : 0.02627
##
##             Sensitivity : 0.9641
##             Specificity : 0.9061
##          Pos Pred Value : 0.9405
##          Neg Pred Value : 0.9425
##              Prevalence : 0.6062
##          Detection Rate : 0.5845
##    Detection Prevalence : 0.6215
##       Balanced Accuracy : 0.9351
##
##        'Positive' Class : nonspam
##
```

Como en el caso de árboles de decisión, los resultados son muy buenos. Por ejemplo, Accuracy = 0.9413.

Paso 3. Predict con probabilidades y umbrales

1º) En este paso sacamos la probabilidad de cada cliente de devolver el crédito.

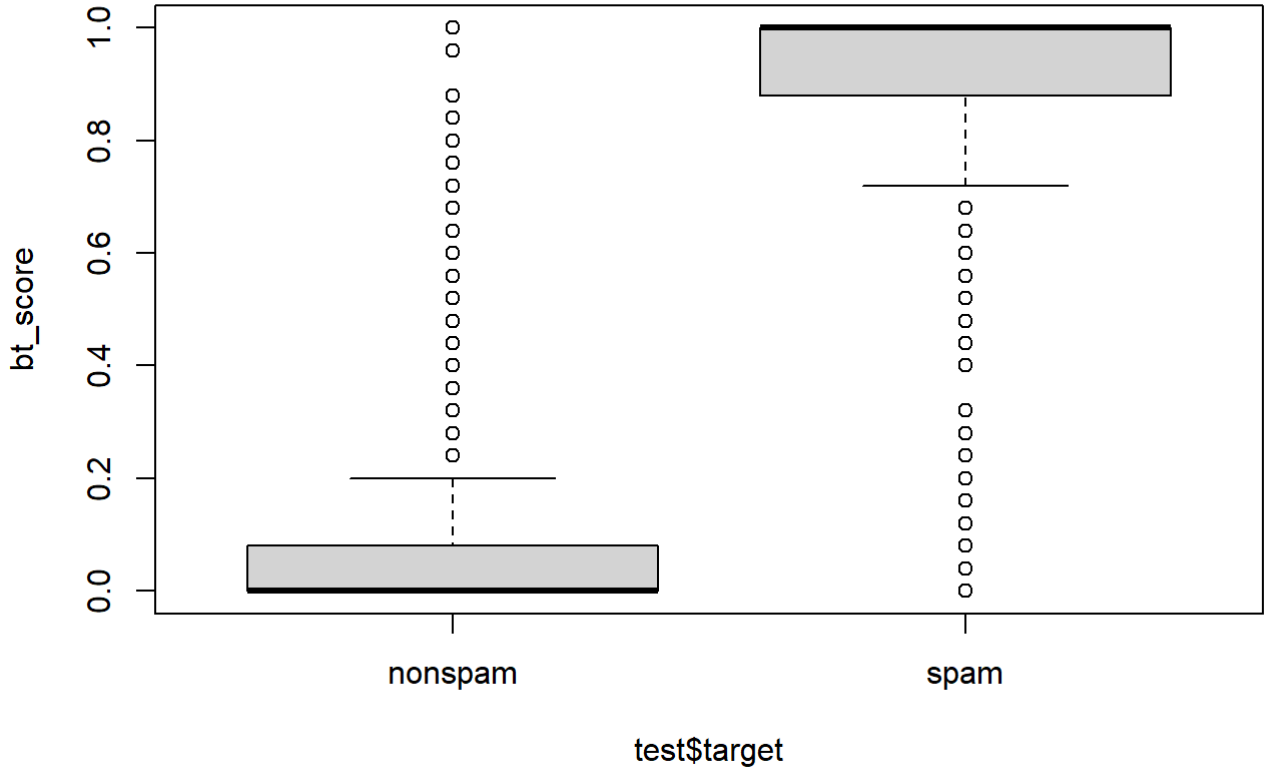
```
bt_score<-predict(mod_bt,test,type = 'prob')[,2]
#Sacamos el predict con el data frame "test"
#Se utiliza "type=prob" (que reporta la probabilidad para cada caso)
#Se le incluye [,2], es decir, la segunda columna, ya que interesa la probabilidad de
 que sea spam

#Sacamos los 6 primeros casos
head(bt_score)
```

```
## [1] 1.00 0.88 0.84 0.96 0.96 0.88
```

Vemos la capacidad de discriminación entre los dos niveles de la TARGET, que es muy elevada.

```
plot(bt_score ~ test$target)
```



2º) Ahora transformamos la probabilidad obtenida en una decisión binaria de Sí es spam o No es spam.

Con la función umbrales probamos diferentes cortes

```
umb_bt<-umbrales(test$target,bt_score)
umb_bt
```

```
##    umbral  acierto precision cobertura       F1
## 1    0.05 81.00073  67.98976  97.79006 80.21148
## 2    0.10 83.82886  71.85792  96.86924 82.50980
## 3    0.15 86.72951  76.23907  96.31676 85.10985
## 4    0.20 88.25236  78.99543  95.58011 86.50000
## 5    0.25 90.35533  83.71711  93.73849 88.44483
## 6    0.30 91.44307  85.95601  93.55433 89.59436
## 7    0.35 92.74837  88.92794  93.18600 91.00719
## 8    0.40 93.61856  90.84381  93.18600 92.00000
## 9    0.45 93.83611  92.88390  91.34438 92.10771
## 10   0.50 94.12618  94.25287  90.60773 92.39437
## 11   0.55 93.90863  94.73684  89.50276 92.04545
## 12   0.60 93.61856  96.14604  87.29282 91.50579
## 13   0.65 93.32850  96.88150  85.81952 91.01562
## 14   0.70 93.03843  97.65458  84.34622 90.51383
## 15   0.75 92.74837  97.84017  83.42541 90.05964
## 16   0.80 92.02321  98.21826  81.21547 88.91129
## 17   0.85 89.84772  98.78935  75.13812 85.35565
## 18   0.90 88.46991  98.97959  71.45488 82.99465
## 19   0.95 84.77157  98.82698  62.06262 76.24434
```

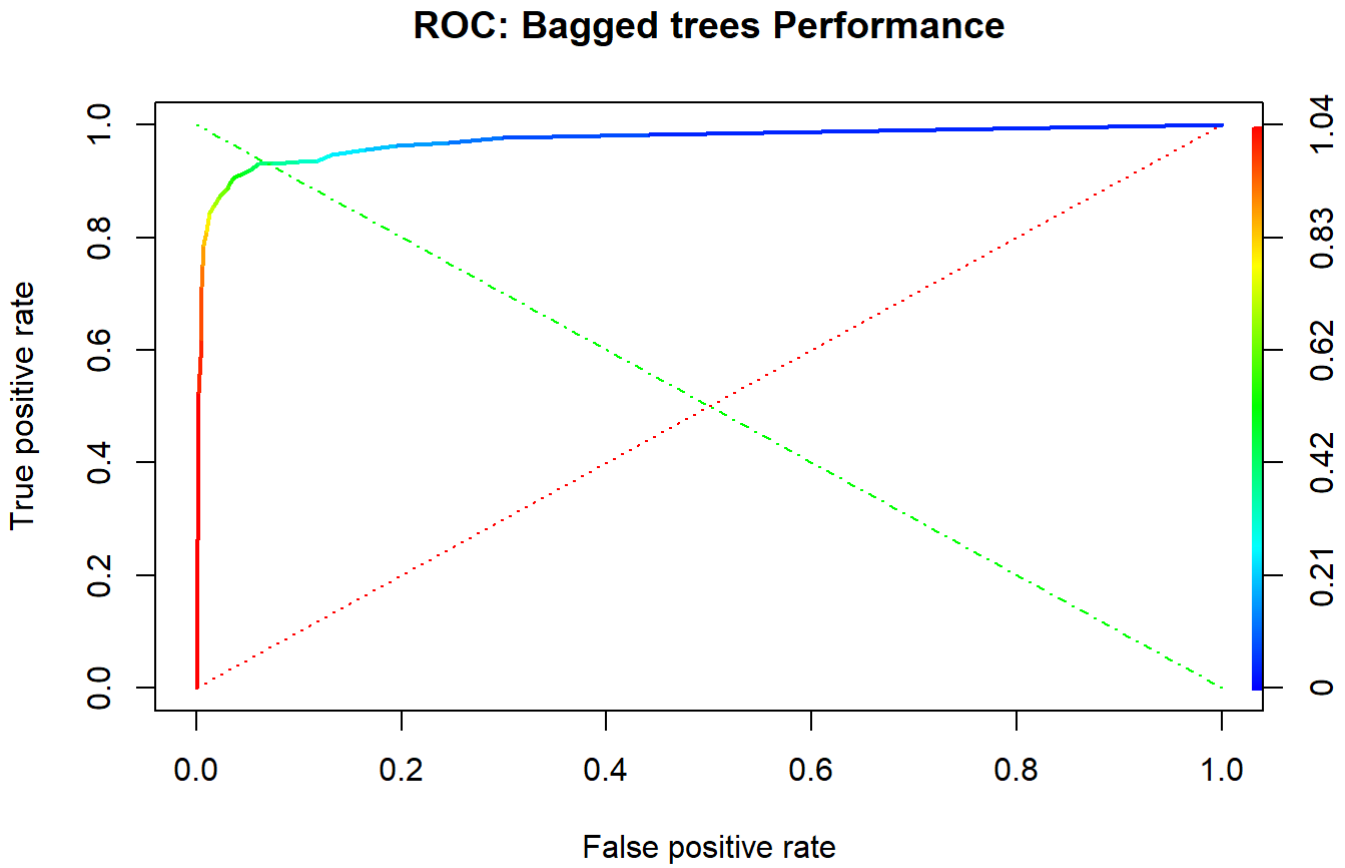
Seleccionamos el umbral que maximiza la F1 (cuando empieza a decaer)

```
umbfinal_bt<-umb_bt[which.max(umb_bt$F1),1]
umbfinal_bt
```

```
## [1] 0.5
```

Paso 4. Curva ROC

```
pred_bt <- prediction(bt_score, test$target)
perf_bt <- performance(pred_bt,"tpr","fpr")
#library(ROCR)
plot(perf_bt, lwd=2, colorize=TRUE, main="ROC: Bagged trees Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



Paso 5. Métricas definitivas

```
#Matriz de confusión con umbral final
score <- ifelse(bt_score > umbfinal_bt, "spam", "nonspam")
MC <- table(test$target, score)
Acc_bt <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
Sen_bt <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
Pr_bt <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
F1_bt <- round(2*Pr_bt *Sen_bt /(Pr_bt +Sen_bt), 2)

#AUC
AUROC_bt <- round(performance(pred_bt, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acc_bt: ", Acc_bt,"\tSen_bt: ", Sen_bt, "\tPr_bt:", Pr_bt, "\tF1_bt:", F1_bt, "\t
AUROC_bt: ", AUROC_bt)
```

```
## Acc_bt:  94.13   Sen_bt:  94.25  Pr_bt: 90.61    F1_bt: 92.39    AUROC_bt:  97.27
```

Se obtiene una AUC de 97.27, lo que indica un modelo con uina gran capacidad para predecir la clasificación buscada.

5. Comparación de los dos modelos

```
# Etiquetas de filas
models <- c('Árboles Decisión', 'Bagged trees')

#Accuracy
models_Acc <- c(dt_Acc, Acc_bt)

#Sensibilidad
models_Sen <- c(dt_Sen, Sen_bt)

#Precisión
models_Pr <- c(dt_Pr, Pr_bt)

#F1
models_F1 <- c(dt_F1, F1_bt)

# AUC
models_AUC <- c(dt_AUROC, AUROC_bt)
```

```
# Combinar métricas
metricas <- as.data.frame(cbind(models, models_Acc, models_Sen, models_Pr, models_F1,
 models_AUC))
```

```
# Colnames
colnames(metricas) <- c("Model", "Acc", "Sen", "Pr", "F1", "AUC")
```

```
# Tabla final de métricas
kable(metricas, caption ="Comparision of Model Performances")
```

Comparision of Model Performances

Model	Acc	Sen	Pr	F1	AUC
Árboles Decisión	91.81	91.03	87.85	89.41	95.19
Bagged trees	94.13	94.25	90.61	92.39	97.27

Se observa que el modelo con Bagging tree mejora obtiene un mejor resultado (AUC = 97.27) con respecto al obtenido con árboles de decisión (AUC = 95.27), mejorando todos los indicadores.