Laboratorio 3

Bruno Olivera

4/21/2019

Práctico 3

Ejercicio 6

From the dataset spam of the kernlab library:

a) Compute and draw the default tree T provided by **rpart** and the decision stump. Look at **T$frame** and examine it.

b) 1) Compute and draw the optimal tree T_1 with associate **cp** parameter given by cross-validation error.
   2) Compute and draw the optimal tree T_2 with associate **cp** parameter given by the **1-SE** rule.
   3) Compare T, T_{max}, T_1 and T_2 in learning and in test samples.

c) Apply Bagging and Random Forest (default) and compare the prediction errors with a single tree.

d) Study the evolution of the **OOB** error with respect to **ntree** using **do.trace**.

e) Calculate the variable importance of the spam variables for Random Forest (default).

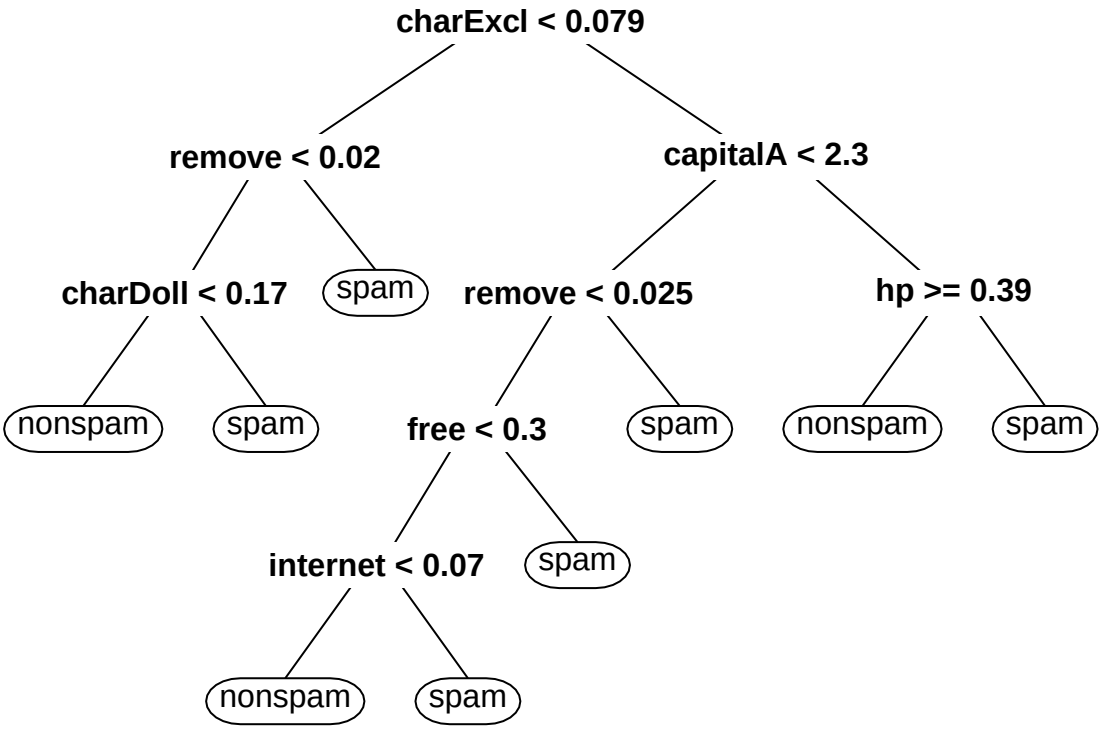
f) Calculate the importance of spam variables for stumps Random Forest.

g) Illustrate the influence of the **mtry** parameter on the **OOB** error and on the variable importance.

h) Train a CART, BAGGING, RANDOM FOREST and SVM model inside a loop(50 iterations) splitting the dataset into train/test in each iteration. Compute test error for each model in each iteration. Determine best model calculating the mean error for each model. Train the best model over the entire dataset.

a) Generamos un arbol de decisión y un decision stump y computamos el error de test.

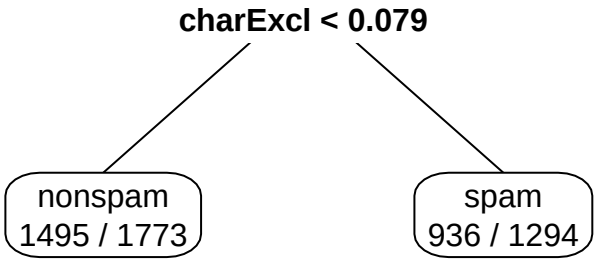
```
library(rpart)
library(rpart.plot)
library(partykit)
library(kernlab)
set.seed(2019)

data(spam)

# separamos los datos en train/test
smp_size = floor(2/3 * nrow(spam))
train_ind = sample(seq_len(nrow(spam)), size=smp_size)
train = spam[train_ind,]
test = spam[-train_ind,]

# generamos y graficamos el arbol por defecto
T=rpart(type~.,data=train)
prp(T,yesno=0)
```




```
# computamos el error en test del arbol T
T_pred=predict(T,type="class",test)
T_error=sum(T_pred!=test[,58])/dim(test)[1]
T_error
```

```
## [1] 0.1016949
```

```
# generamos y graficamos el decision stump
DS=rpart(type~.,data=train,maxdepth=1)
prp(DS,extra=2,yesno=0)
```




```
# computamos el error en test del decision stump
DS_pred=predict(DS,type="class",test)
DS_error=sum(DS_pred!=test[,58])/dim(test)[1]
DS_error
```

```
## [1] 0.2118644
```

```
# Analizamos T$frame
T$frame
```

```
##                var    n   wt  dev yval  complexity ncompete nsurrogate
## 1  charExclamation 3067 3067 1214    1 0.476112026        4          5
## 2           remove 1773 1773  278    1 0.063426689        4          3
```

```
## 4       charDollar 1648 1648  177    1 0.020593081        4          4
## 8           <leaf> 1597 1597  139    1 0.006589786        0          0
## 9           <leaf>   51   51   13    2 0.004118616        0          0
## 5           <leaf>  125  125   24    2 0.009060956        0          0
## 3       capitalAve 1294 1294  358    2 0.091433278        4          5
## 6           remove  431  431  160    1 0.052718287        4          5
## 12            free  363  363   94    1 0.017298188        4          5
## 24        internet  292  292   48    1 0.011532125        4          4
## 48          <leaf>  270  270   30    1 0.004118616        0          0
## 49          <leaf>   22   22    4    2 0.010000000        0          0
## 25          <leaf>   71   71   25    2 0.000000000        0          0
## 13          <leaf>   68   68    2    2 0.010000000        0          0
## 7               hp  863  863   87    2 0.022240527        4          5
## 14          <leaf>   37   37    5    1 0.010000000        0          0
## 15          <leaf>  826  826   55    2 0.007413509        0          0
##        yval2.V1     yval2.V2     yval2.V3     yval2.V4     yval2.V5
## 1  1.000000e+00 1.853000e+03 1.214000e+03 6.041735e-01 3.958265e-01
## 2  1.000000e+00 1.495000e+03 2.780000e+02 8.432036e-01 1.567964e-01
## 4  1.000000e+00 1.471000e+03 1.770000e+02 8.925971e-01 1.074029e-01
## 8  1.000000e+00 1.458000e+03 1.390000e+02 9.129618e-01 8.703820e-02
## 9  2.000000e+00 1.300000e+01 3.800000e+01 2.549020e-01 7.450980e-01
## 5  2.000000e+00 2.400000e+01 1.010000e+02 1.920000e-01 8.080000e-01
## 3  2.000000e+00 3.580000e+02 9.360000e+02 2.766615e-01 7.233385e-01
## 6  1.000000e+00 2.710000e+02 1.600000e+02 6.287703e-01 3.712297e-01
## 12 1.000000e+00 2.690000e+02 9.400000e+01 7.410468e-01 2.589532e-01
## 24 1.000000e+00 2.440000e+02 4.800000e+01 8.356164e-01 1.643836e-01
## 48 1.000000e+00 2.400000e+02 3.000000e+01 8.888889e-01 1.111111e-01
## 49 2.000000e+00 4.000000e+00 1.800000e+01 1.818182e-01 8.181818e-01
## 25 2.000000e+00 2.500000e+01 4.600000e+01 3.521127e-01 6.478873e-01
## 13 2.000000e+00 2.000000e+00 6.600000e+01 2.941176e-02 9.705882e-01
## 7  2.000000e+00 8.700000e+01 7.760000e+02 1.008111e-01 8.991889e-01
## 14 1.000000e+00 3.200000e+01 5.000000e+00 8.648649e-01 1.351351e-01
## 15 2.000000e+00 5.500000e+01 7.710000e+02 6.658596e-02 9.334140e-01
##    yval2.nodeprob
## 1    1.000000e+00
## 2    5.780893e-01
## 4    5.373329e-01
## 8    5.207043e-01
## 9    1.662863e-02
## 5    4.075644e-02
## 3    4.219107e-01
## 6    1.405282e-01
## 12   1.183567e-01
## 24   9.520704e-02
## 48   8.803391e-02
## 49   7.173133e-03
## 25   2.314966e-02
## 13   2.217150e-02
## 7    2.813825e-01
## 14   1.206391e-02
## 15   2.693186e-01
```

En T$frame para cada nodo tenemos la siguiente información:

- **var**: nombre de la variable usada en el split del nodo o "<leaf>" en caso de ser un nodo hoja

- **n**: número de observaciones que llegan al nodo
- **wt**: la suma de los pesos para las observaciones que llegan al nodo (en este caso no estamos usando pesos por lo que **wt** coincide con **n**).
- **dev**: la desviación del nodo, es decir la cantidad de observaciones mal clasificadas
- **yval**: la clase asignada al nodo
- **complexity**: el parámetro de complejidad (**cp**) que haría que el nodo colapse. Es decir que si entrenamos el modelo con un **cp** mayor a este valor, el nodo no va a existir en este nuevo modelo.
- **ncompete**: el número de splits competidores que se almacenan. Suele ser de interés saber no solamente cuál es la mejor variable para hacer el split, sino también cuál salió segunda, tercera, etc.
- **nsurrogate**: el número de splits surrogativos que se almacenan.
- **yval2**: una matriz con la siguiente información adicional:
  - **V1**: la clase asignada al nodo (coincide con **yval**)
  - **V2**: la cantidad de muestras de la clase 1 en el nodo (igual a **n-dev** si **yval**=1 o a **dev** si **yval**=2)
  - **V3**: la cantidad de muestras de la clase 2 en el nodo (igual a **dev** si **yval**=1 o a **n-dev** si **yval**=2)
  - **V4**: probabilidad de la clase 1 en el nodo (coincide con **V2/n**)
  - **V5**: probabilidad de la clase 2 en el nodo (coincide con **V3/n**)
  - **nodeprob**: la probabilidad del nodo. Vale 1 para la raíz, y debe sumar 1 entre las probabilidades de las dos hojas de un nodo. Por ejemplo en este caso se puede ver que la **nodeprob** del nodo 2) es 5.78 y la del nodo 3) es 4.22 lo cual suma 1.

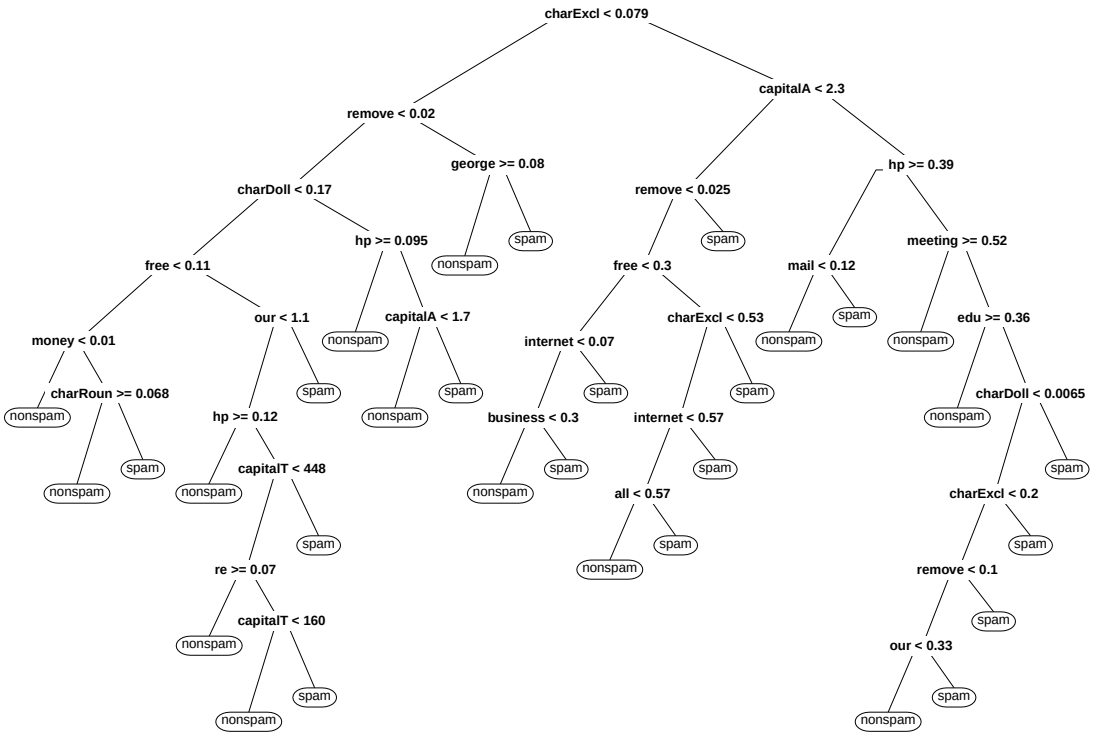
b)

1) Computamos el arbol óptimo para cross-validation. Para esto nos fijamos en la cptable del arbol maximal y nos quedamos con el cp del arbol con menor error de cross-validation.

```
# generamos el arbol maximal para luego calcular el
# cp de cross-validation y el cp de la regla 1-SE
Tmax=rpart(type~.,data=train,cp=0)

# calculamos el cp dado por el error de cross-validation
CP_cross_validation = Tmax$cptable[which.min(Tmax$cptable[,"xerror"]),"CP"]
CP_cross_validation
```

```
## [1] 0.002059308
```

```
# computamos y graficamos el arbol óptimo para cross-validation
T1=rpart(type~.,data=train,cp=CP_cross_validation)
prp(T1,yesno=0)
```

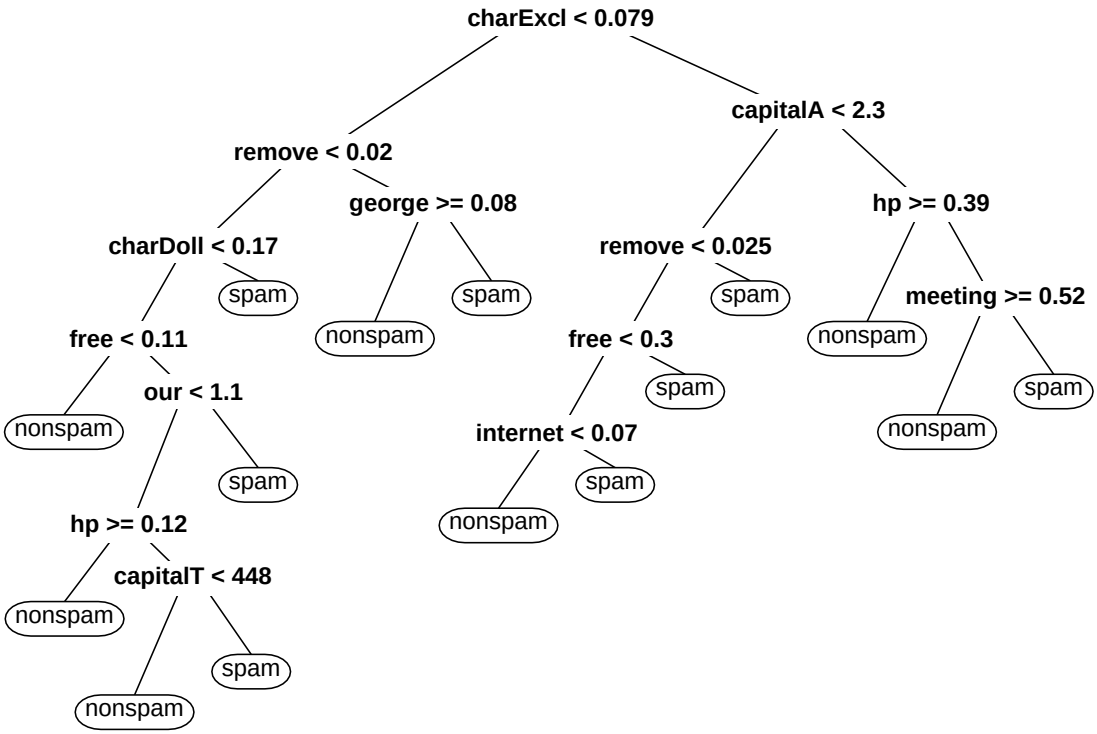


2) Computamos el arbol óptimo para la regla 1-SE. Para esto hallamos el arbol cuyo error de cross-validation es el mínimo, a ese valor le sumamos la desviación, y luego hallamos el arbol más simple (menos nodos) cuyo error de cross-validation sea menor que esta suma.

```
# calculamos el cp dado por la regla 1-SE
xerror = Tmax$cptable[,4]
xstd   = Tmax$cptable[,5]
t.opt  = min(seq(along = xerror)[xerror <= min(xerror) + xstd[which.min(xerror)]])
CP_1_SE = Tmax$cptable[t.opt,1]
CP_1_SE
```

```
## [1] 0.004118616
```

```
# computamos y graficamos el arbol óptimo para la regla 1-SE
T2=rpart(type~.,data=train,cp=CP_1_SE)
prp(T2,yesno=0)
```

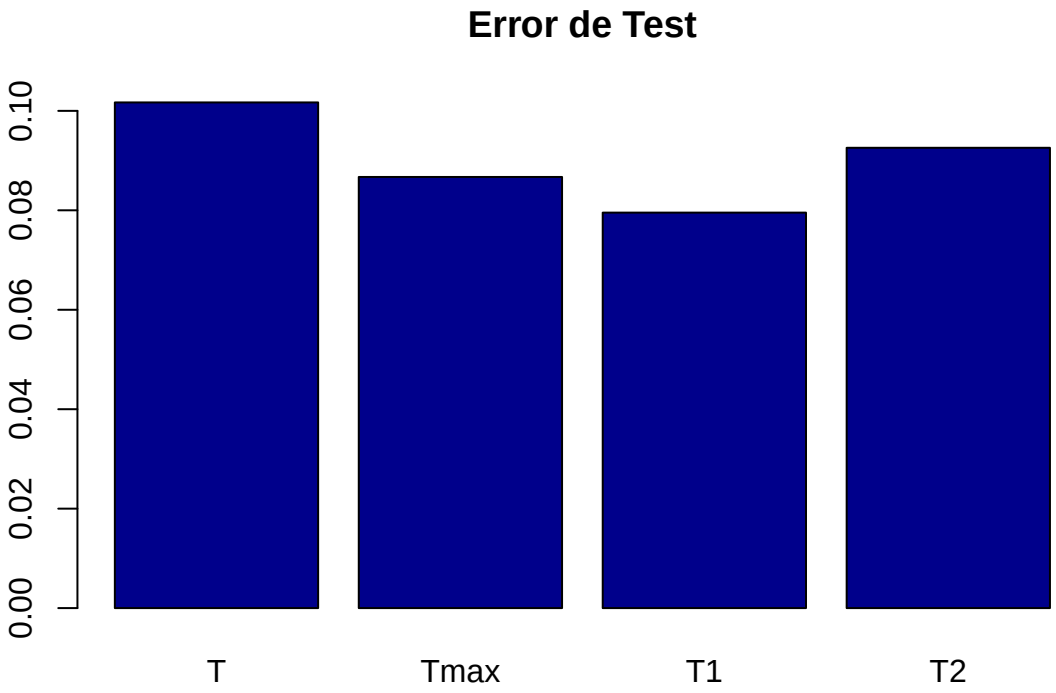


3) Comparamos los errores de test para los árboles T,T_{max},T_1,T_2. El error para T ya lo tenemos, falta calcular los otros tres.

```
# computamos el error de test para Tmax
Tmax_pred=predict(Tmax,type="class",test)
Tmax_error=sum(Tmax_pred!=test[,58])/dim(test)[1]

# computamos el error de test para T1
T1_pred=predict(T1,type="class",test)
T1_error=sum(T1_pred!=test[,58])/dim(test)[1]

# computamos el error de test para T2
T2_pred=predict(T2,type="class",test)
T2_error=sum(T2_pred!=test[,58])/dim(test)[1]

# comparamos los errores de test para T,Tmax,T1,T2
errors=rbind("T"=T_error,"Tmax"=Tmax_error,"T1"=T1_error,"T2"=T2_error)
colnames(errors)="test_error"
errors
```

```
##      test_error
## T    0.10169492
## Tmax 0.08670143
## T1   0.07953064
## T2   0.09256845
```

```
# graficamos los errores de test
barplot(t(errors),col="darkblue",main="Error de Test")
```

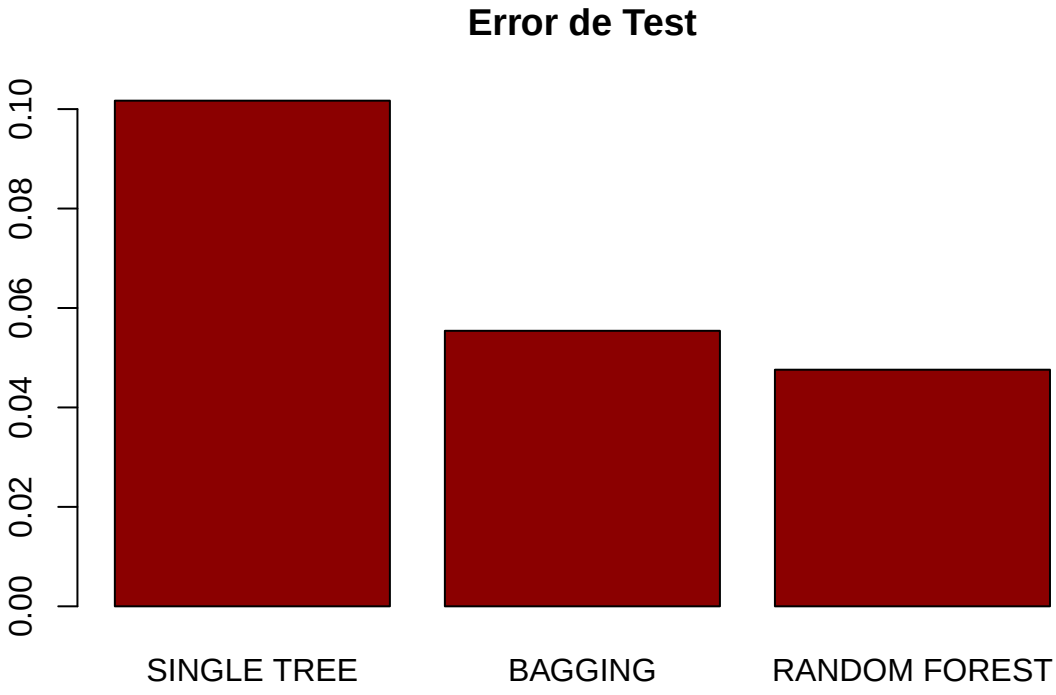

Error de Test
0.00
0.02
0.04
0.06
0.08
0.10
T
Tmax
T1
T2

c) Generamos modelos BAGGING y RANDOM FOREST y comparamos los errores de predicción con el modelo de un solo arbol

```
library(ipred)
library(randomForest)

# entrenamos un modelo bagging y calculamos su error de test
bag=bagging(type~.,data=train,cp=0.01,coob=TRUE)
bag_pred=predict(bag,type="class",test)
bag_error=sum(bag_pred!=test[,58])/dim(test)[1]

# entrenamos un modelo random forest y calculamos su error de test
rf=randomForest(type~.,data=train,cp=0.01,coob=TRUE)
rf_pred=predict(rf,type="class",test)
rf_error=sum(rf_pred!=test[,58])/dim(test)[1]

# comparamos los errores de test para bag,rf y T
errors=rbind("SINGLE TREE"=T_error,"BAGGING"=bag_error,"RANDOM FOREST"=rf_error)
colnames(errors)="test_error"
errors
```

```
##               test_error
## SINGLE TREE   0.10169492
## BAGGING       0.05541069
## RANDOM FOREST 0.04758801
```

```
# graficamos los errores de test
barplot(t(errors),col="darkred",main="Error de Test")
```

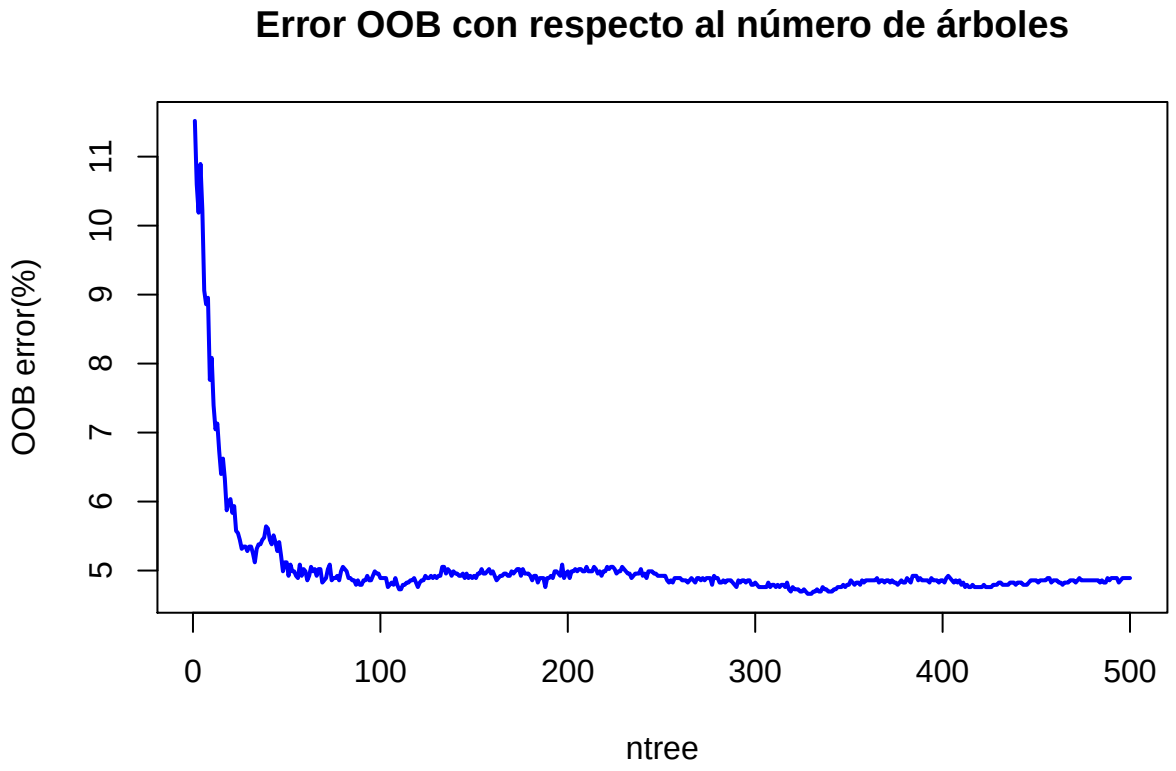


d) Estudiamos la evolución del error OOB según **ntree** con el parámetro **do.trace**. En cada fila vemos el error OOB y el error de clasificación para cada clase.

```
# computamos un randomForest con la opción do.trace (lo hacemos cada 50 árboles
# para no imprimir las 500 líneas, más abajo hacemos una gráfica teniendo en
# cuenta los 500 valores).
rf_trace=randomForest(type~.,data=train,cp=0.01,coob=TRUE,do.trace=50)
```

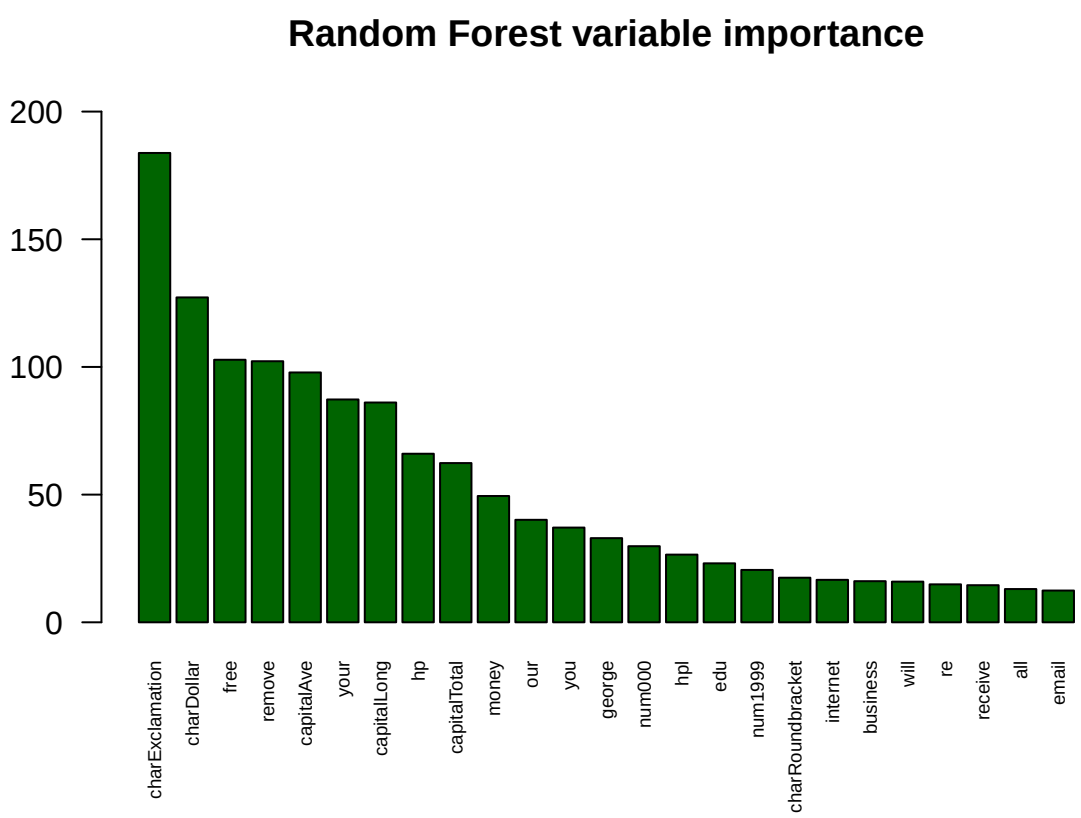
```
## ntree      OOB      1      2
##    50:   5.12%  2.91%  8.48%
##   100:   4.89%  2.75%  8.15%
##   150:   4.92%  2.81%  8.15%
##   200:   4.99%  2.86%  8.24%
##   250:   4.92%  2.75%  8.24%
##   300:   4.83%  2.70%  8.07%
##   350:   4.79%  2.70%  7.99%
##   400:   4.86%  2.81%  7.99%
##   450:   4.86%  2.81%  7.99%
##   500:   4.89%  2.81%  8.07%
```

```
# para un análisis más a fondo graficamos la evolución del error
plot(100*rf_trace$err.rate[,1], main="Error OOB con respecto al número de árboles",
     type="l",ylab="OOB error(%)",xlab="ntree",lwd=2,col="blue")
```

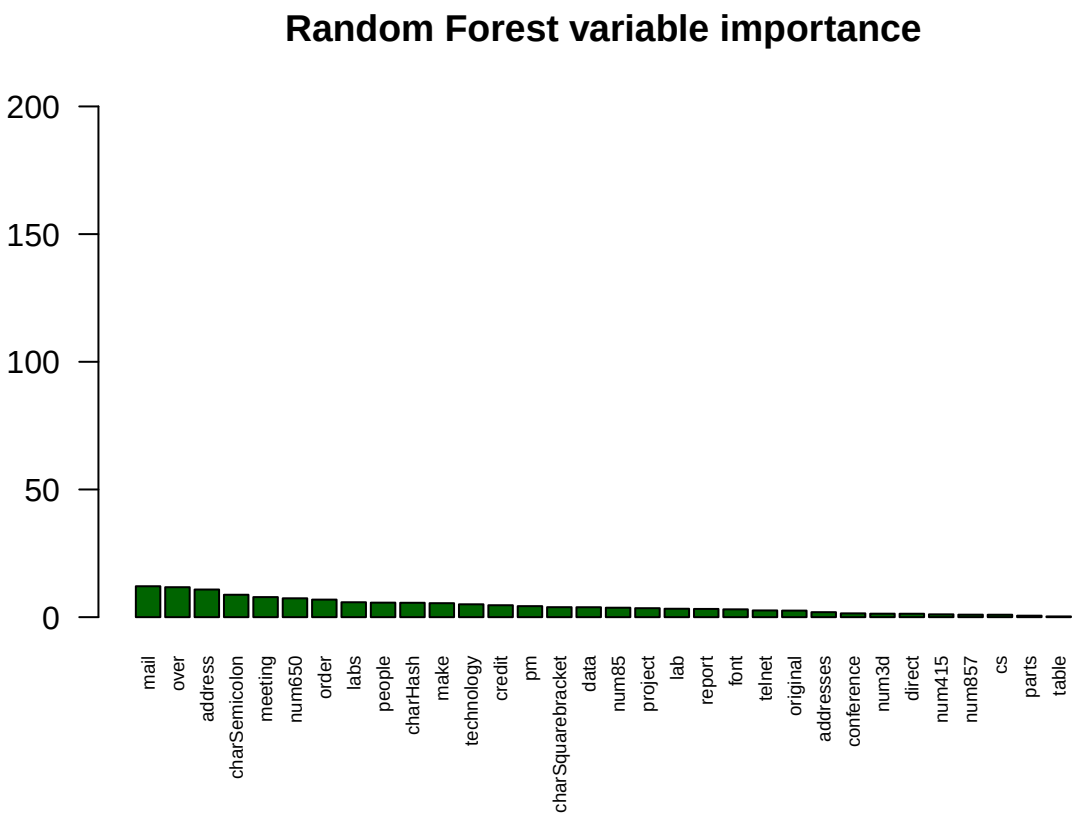


e) Analizamos la importancia de las variables para el modelo Random Forest.

```
# obtenemos la importancia de las variables y las ordenamos decrecientemente
rf_importance=importance(rf)[order(importance(rf),decreasing=TRUE),]

# graficamos la importancia de las variables
barplot(rf_importance[1:25], main="Random Forest variable importance",
        las=2,cex.names=0.6,ylim=c(0,200),margin=1,col="darkgreen")
```

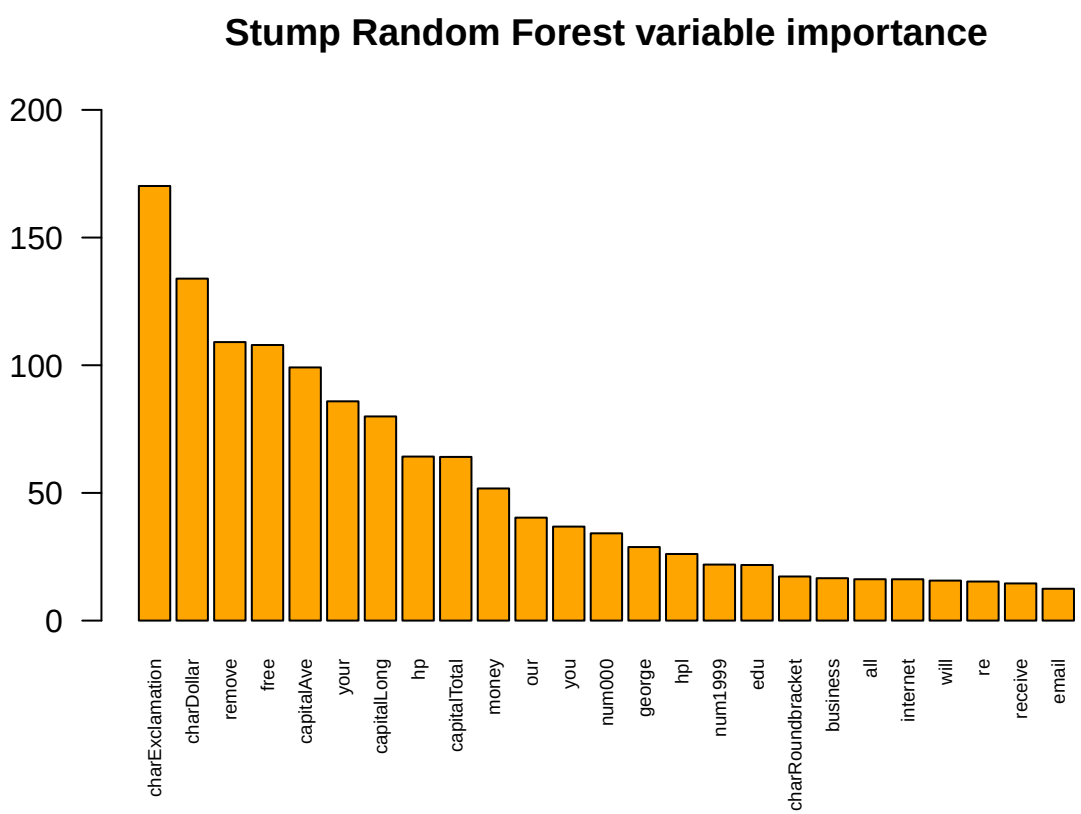



```
# graficamos la importancia de las variables
barplot(rf_importance[26:57], main="Random Forest variable importance",
        las=2,cex.names=0.6,ylim=c(0,200),margin=1,col="darkgreen")
```

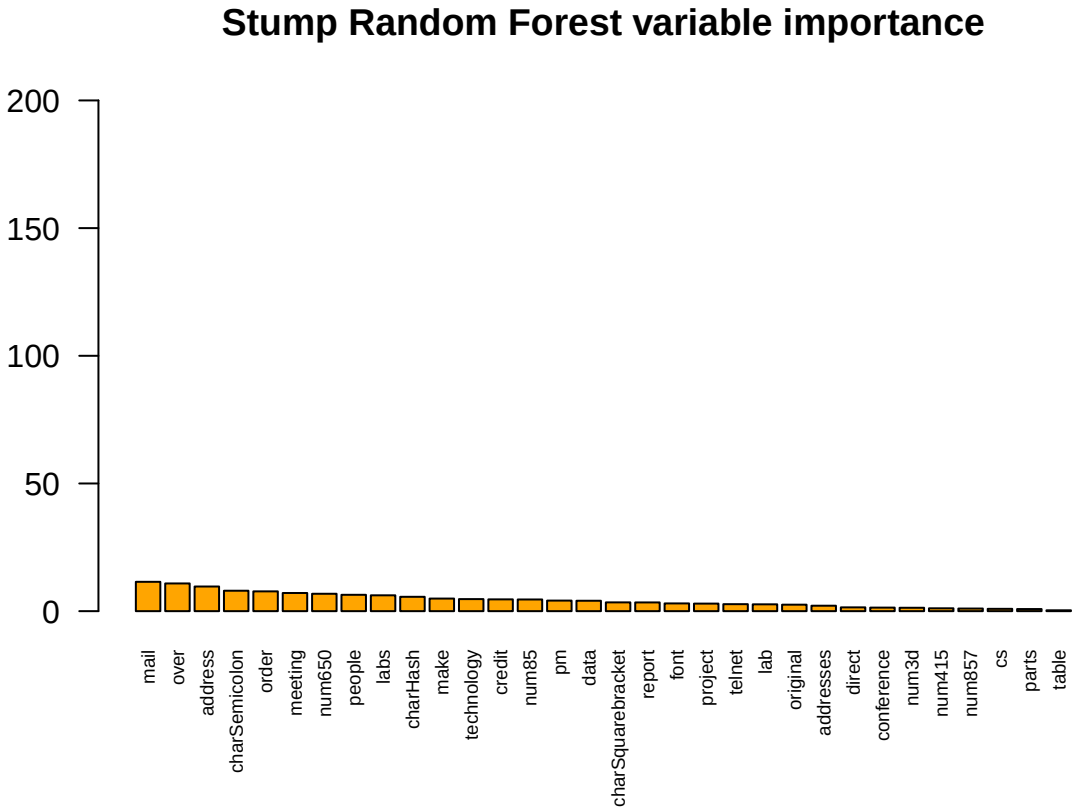

Random Forest variable importance
200
150
100
50
0
mail
over
address
charSemicolon
meeting
num650
order
labs
people
charHash
make
technology
credit
pm
charSquarebracket
data
num85
project
lab
report
font
telnet
original
addresses
conference
num3d
direct
num415
num857
cs
parts
table

f) Analizamos la importancia de las variables para el modelo stump Random Forest.

```
rf_stump=randomForest(type~.,data=train,cp=0.01,maxdep=1,coob=TRUE)

# obtenemos la importancia de las variables y las ordenamos decrecientemente
rf_stump_importance=importance(rf_stump)[order(importance(rf_stump),decreasing=TRUE),]

# graficamos la importancia de las variables
barplot(rf_stump_importance[1:25], main="Stump Random Forest variable importance",
        las=2,cex.names=0.6,ylim=c(0,200),margin=1,col="orange")
```



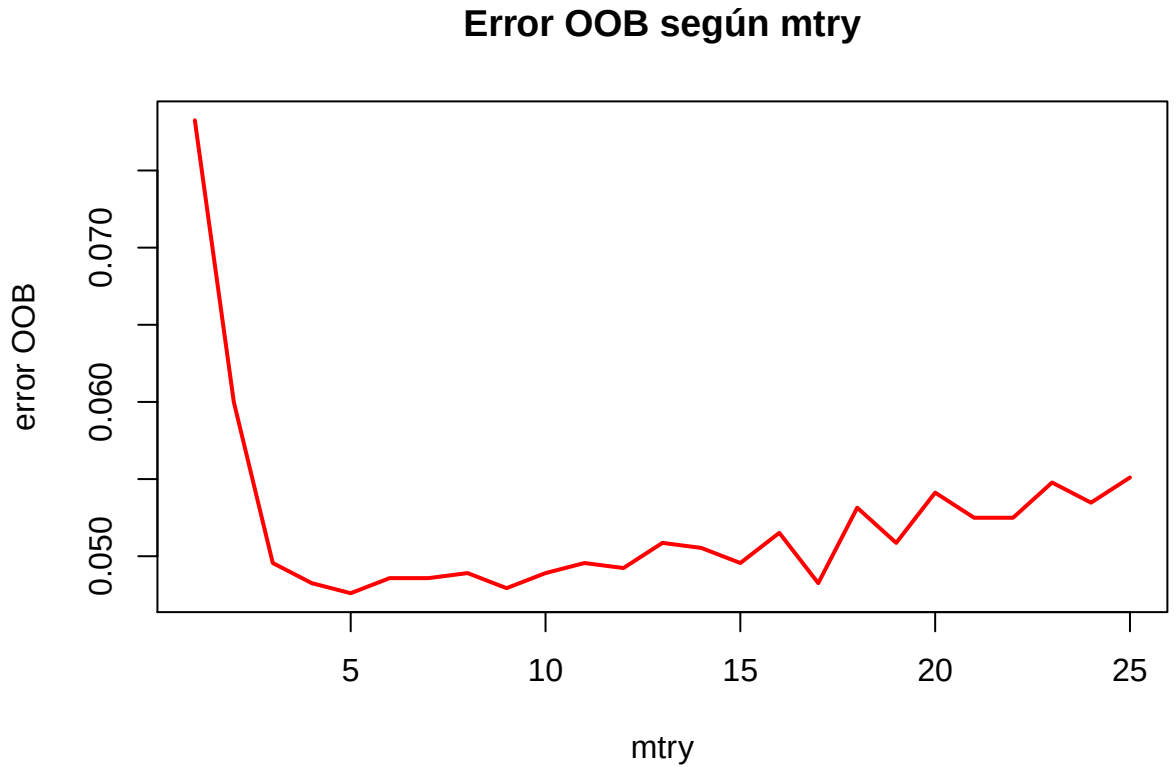

```
# graficamos la importancia de las variables
barplot(rf_stump_importance[26:57], main="Stump Random Forest variable importance",
        las=2,cex.names=0.6,ylim=c(0,200),margin=1,col="orange")
```


Stump Random Forest variable importance
200
150
100
50
0
mail
over
address
charSemicolon
order
meeting
num650
people
labs
charHash
make
technology
credit
num85
pm
data
charSquarebracket
report
font
project
telnet
lab
original
addresses
direct
conference
num3d
num415
num857
cs
parts
table

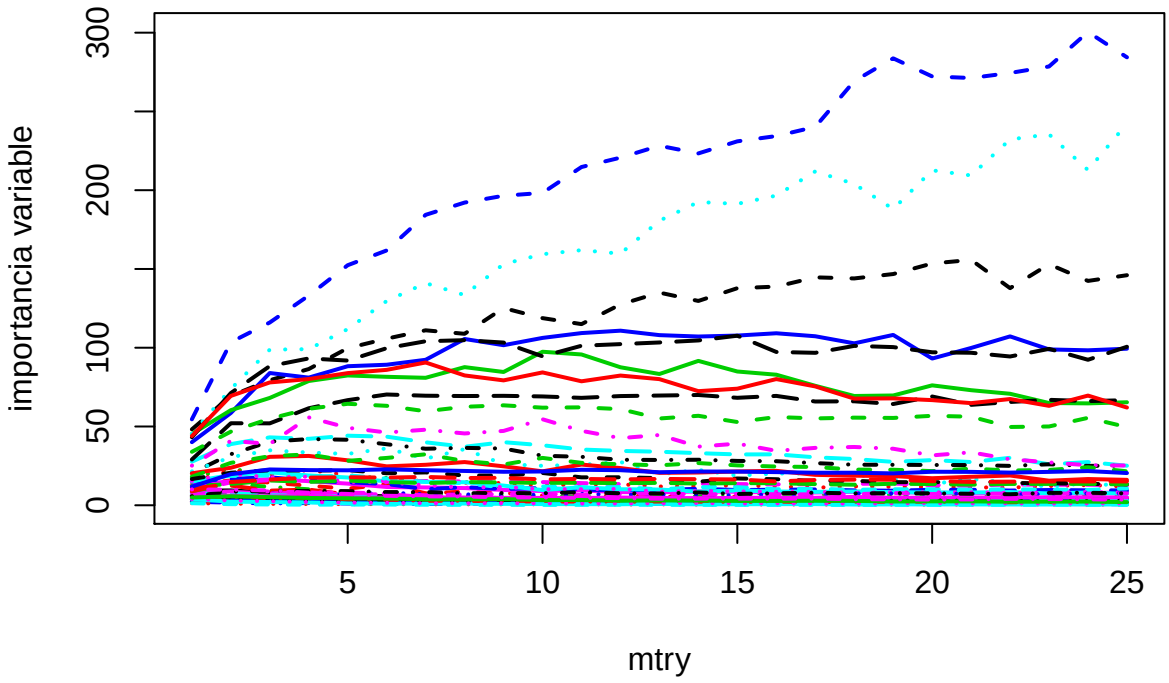
g) Analizamos la influencia del parámetro **mtry** en el error OOB y la importancia de las variables. El parámetro **mtry** determina la cantidad de variables candidatas que se tienen en cuenta en cada split. Se suele usar $sqrt(p)$ siendo p la cantidad de variables (para el caso de spam se usa **mtry** $= 7$ por defecto).

Para hacer este estudio vamos a hacer un bucle desde $k = 1$ a $k = 25$ generando un modelo Random Forest con **mtry** $= k$ para cada valor de k, guardando el error OOB y la importancia de las variables en cada iteración. Al final graficamos los resultados.

```
# variables para ir guardando los resultados
importance=NULL
oob=NULL

for(k in 1:25){
  # generamos el random forest con mtry=k
  rf=randomForest(type~.,data=train,coob=TRUE,mtry=k)
  # almacenamos la importancia para este valor de k
  importance=cbind(importance,rf$importance)
  # almacenamos el error OOB para este valor de k
  oob=cbind(oob,rf$err.rate[500,1])
}

# graficamos el error OBB con respecto al mtry
plot(1:25,oob,type="l",main="Error OOB según mtry",
     lwd=2,col="red",xlab="mtry",ylab="error OOB")
```

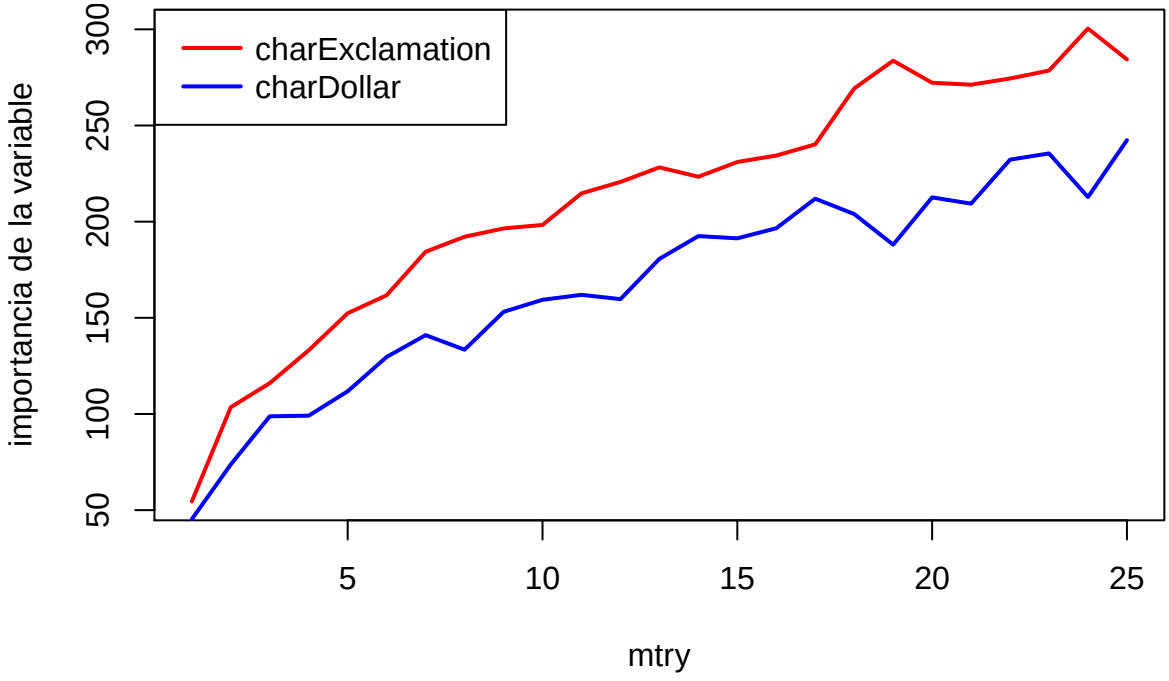



```
# graficamos la importancia de las variables con respecto al mtry
matplot(t(importance),type="l",lwd=2,xlab="mtry",ylab="importancia variable")
```

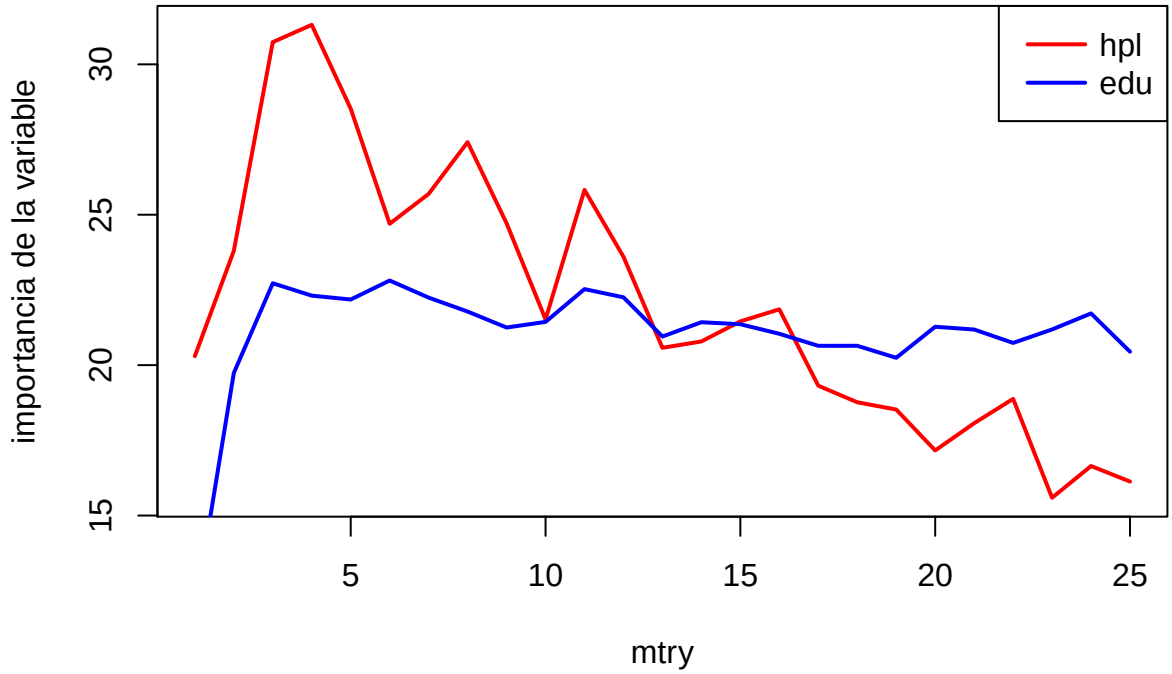


Como la gráfica anterior está un poco sobrecargada, vamos a graficar la importancia de algunas variables aparte para que se pueda apreciar mejor la influencia de mtry

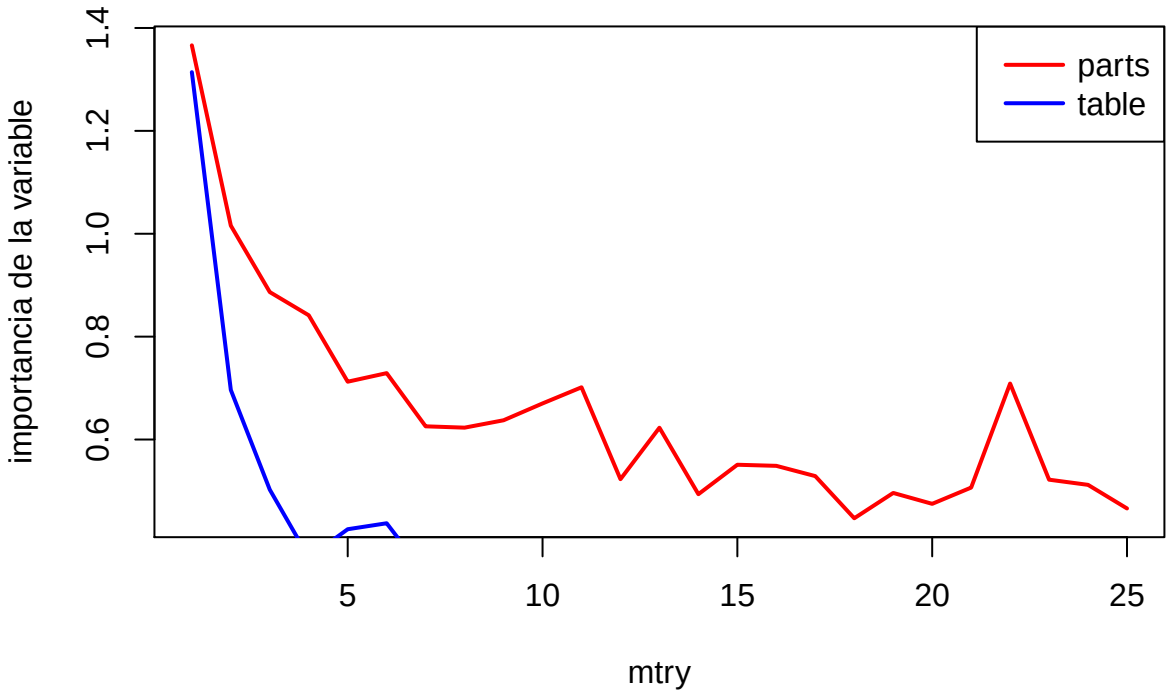
```
# gráficas para las dos variables más importantes en el caso por defecto
plot(1:25,importance[52,],type="l",
     lwd=2,col="red",xlab="mtry",ylab="importancia de la variable")
lines(1:25,importance[53,],type="l",
     lwd=2,col="blue",xlab="mtry",ylab="importancia de la variable")
legend("topleft",x.intersp=0.5,
       col=c("red","blue"),lwd=2,
       legend=c(rownames(importance)[52],rownames(importance)[53]))
```



```r
# grvficas para dos variables con importancia cercana al promedio en el caso por defecto
plot(1:25,importance[26,],type="l",
     lwd=2,col="red",xlab="mtry",ylab="importancia de la variable")
lines(1:25,importance[46,],type="l",
     lwd=2,col="blue",xlab="mtry",ylab="importancia de la variable")
legend("topright",x.intersp=0.5,
       col=c("red","blue"),lwd=2,
       legend=c(rownames(importance)[26],rownames(importance)[46]))
```

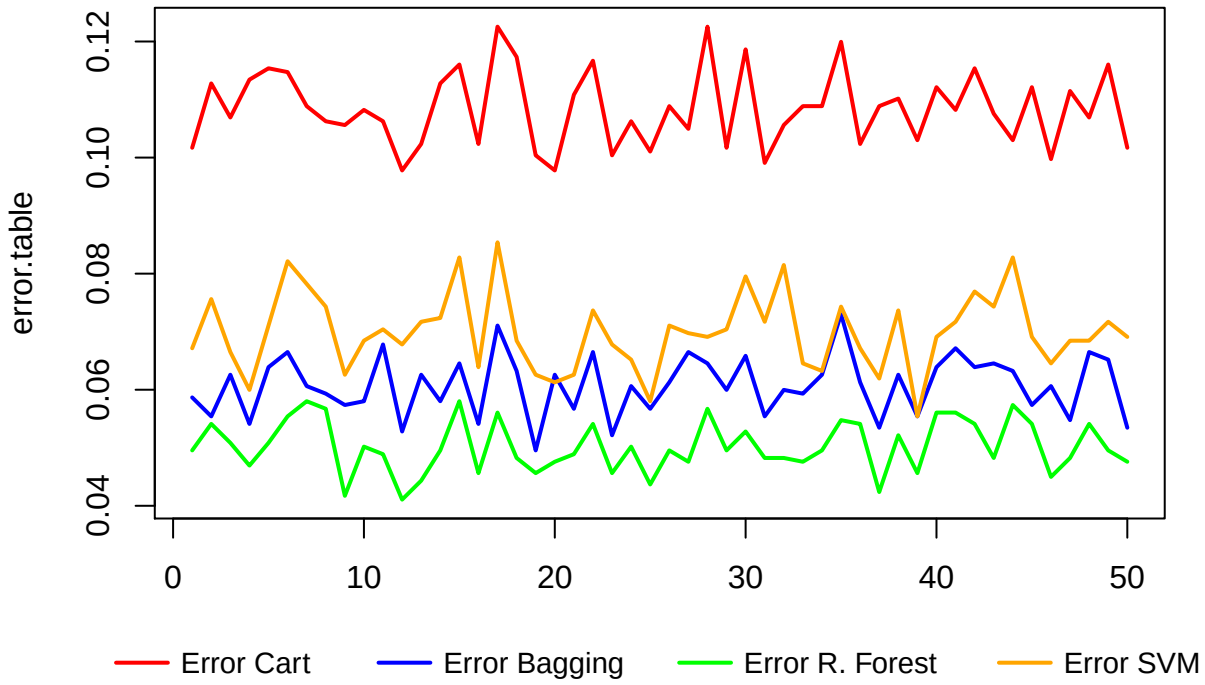



```r
# graficas para las dos variables menos importantes en el caso por defecto
plot(1:25,importance[38,],type="l",
     lwd=2,col="red",xlab="mtry",ylab="importancia de la variable")
lines(1:25,importance[47,],type="l",
     lwd=2,col="blue",xlab="mtry",ylab="importancia de la variable")
legend("topright",x.intersp=0.5,
       col=c("red","blue"),lwd=2,
       legend=c(rownames(importance)[38],rownames(importance)[47]))
```


importancia de la variable
0.6
0.8
1.0
1.2
1.4
5
10
15
20
25
mtry
parts
table

h) Entrenamos modelos CART, BAGGING, RANDOM FOREST y SVM en un bucle de 50 iteraciones, separando el dataset en train/test en cada iteración(con seeds distintas). Almacenamos los errores de test de cada modelo en cada iteración. Al terminar el bucle graficamos los errores de cada modelo para compararlos y nos quedamos con el mejor modelo. El mejor modelo va a ser el que tenga la menor media de error. Finalmente entrenamos el mejor modelo sobre el dataset spam entero.

```
library(e1071)

error.table = NULL

for(k in 1:50){
  set.seed(2019+k)
  # separamos datos en train/test
  smp_size = floor(2/3 * nrow(spam))
  train_ind = sample(seq_len(nrow(spam)), size = smp_size)
  train =  spam[train_ind,]
  test  = spam[-train_ind,]

  # entrenamos modelos
  model.cart =        rpart(type~.,data=train)
  model.bag  =      bagging(type~.,data=train)
  model.rf   = randomForest(type~.,data=train)
  model.svm  =          svm(type~.,data=train)

  # obtenemos predicciones
  pred.cart = predict(model.cart, type="class", test)
  pred.bag  = predict(model.bag,  type="class", test)
  pred.rf   = predict(model.rf,   type="class", test)
  pred.svm  = predict(model.svm,  type="class", test)

  # computamos los errores
  error.cart = sum(pred.cart != test[,58]) / dim(test)[1]
  error.bag  = sum(pred.bag  != test[,58]) / dim(test)[1]
  error.rf   = sum(pred.rf   != test[,58]) / dim(test)[1]
  error.svm  = sum(pred.svm  != test[,58]) / dim(test)[1]

  # los agregamos a la tabla
  error.table = rbind(error.table,c(error.cart,error.bag,error.rf,error.svm))

}

# graficamos los errores
colnames(error.table)=c("Error Cart","Error Bagging",
                        "Error R. Forest","Error SVM")
matplot(error.table,type="l",lwd=2,lty=1,col=c("red","blue","green","orange"))
legend("bottom",inset=c(0,-.35),xpd=TRUE,horiz=TRUE,x.intersp=0.5,
       cex=.9,text.width=c(10,11,0,12),box.lty=0,
       legend=c(colnames(error.table)[1],colnames(error.table)[2],
                colnames(error.table)[3],colnames(error.table)[4]),
       col=c("red","blue","green","orange"),bg="white",lwd=c(2,2,2,2))
```




```
# calculamos la media de los errores
error.means=colMeans(error.table); error.means
```

```
##      Error Cart   Error Bagging Error R. Forest       Error SVM
##      0.10844850      0.06074316      0.05022164      0.06998696
```

```
# nos quedamos con el mejor y entrenamos sobre todo el data set
index = which.min(error.means)

invisible(ifelse(index==1,
                 model.best<-rpart(type~.,data=spam),
                 ifelse(index==2,
                        model.best<-bagging(type~.,data=spam),
                        ifelse(index==3,
                               model.best<-randomForest(type~.,data=spam),
                               model.best<-svm(type~.,data=spam)))))

model.best
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = spam)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 4.43%
## Confusion matrix:
##         nonspam spam class.error
## nonspam    2712   76  0.02725968
## spam        128 1685  0.07060121
```