Evaluación MLII (temas 1, 2, 6 y 7): Ejercicio 1

Modelos clasificación binaria (Random Forest y Perceptrón muliticapas)

Inmaculada Perea Fernández

mayo 2017

Cargar el data frame *LetterRecognition* de la libería *mlbench*, que contiene datos apropiado para construir un sistema de reconocimiento de caracteres. La variable *lettr* es de tipo factor, presentando 26 niveles, cada uno es una letra mayúscula.

Establecer la semilla del generador de números pseudo-aleatorios de R mediante *set.seed(m)*, siendo *m* el número obtenido con las tres últimas cifras del DNI, y elegir aleatoriamente dos letras.

Utilizando los casos que correspondan a alguna de ambas letras, construir de forma razonada y comparar modelos de clasificación binaria basados en *Random Forests* y el *Perceptrón Multicapas* (nnet).

1 Carga, inspección y preparación de los datos

1.1. Carga e instalación de librerías necesarias

```
if (!require('mlbench')) install.packages('mlbench'); library('mlbench')
if (!require('randomForest')) install.packages('randomForest'); library('randomForest')
if (!require('nnet')) install.packages('nnet'); library('nnet')
if (!require('e1071')) install.packages('e1071'); library('e1071')
```

1.2. Carga e inspección de los datos

```
data(LetterRecognition)
head(LetterRecognition)
```

```
##   lettr x.box y.box width high onpix x.bar y.bar x2bar y2bar xybar x2ybr
## 1     T     2     8     3    5     1     8    13     0     6     6    10
## 2     I     5    12     3    7     2    10     5     5     4    13     3
## 3     D     4    11     6    8     6    10     6     2     6    10     3
## 4     N     7    11     6    6     3     5     9     4     6     4     4
## 5     G     2     1     3    1     1     8     6     6     6     6     5
## 6     S     4    11     5    8     3     8     8     6     9     5     6
##   xy2br x.ege xegvy y.ege yegvx
## 1     8     0     8     0     8
## 2     9     2     8     4    10
## 3     7     3     7     3     9
## 4    10     6    10     2     8
## 5     9     1     7     5    10
## 6     6     0     8     9     7
```

```
str(LetterRecognition)
```

```
## 'data.frame':    20000 obs. of  17 variables:
##  $ lettr: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...
```

```
##  $ x.box: num  2 5 4 7 2 4 4 1 2 11 ...
##  $ y.box: num  8 12 11 11 1 11 2 1 2 15 ...
##  $ width: num  3 3 6 6 3 5 5 3 4 13 ...
##  $ high : num  5 7 8 6 1 8 4 2 4 9 ...
##  $ onpix: num  1 2 6 3 1 3 4 1 2 7 ...
##  $ x.bar: num  8 10 10 5 8 8 8 8 10 13 ...
##  $ y.bar: num  13 5 6 9 6 8 7 2 6 2 ...
##  $ x2bar: num  0 5 2 4 6 6 6 2 2 6 ...
##  $ y2bar: num  6 4 6 6 6 9 6 2 6 2 ...
##  $ xybar: num  6 13 10 4 6 5 7 8 12 12 ...
##  $ x2ybr: num  10 3 3 4 5 6 6 2 4 1 ...
##  $ xy2br: num  8 9 7 10 9 6 6 8 8 9 ...
##  $ x.ege: num  0 2 3 6 1 0 2 1 1 8 ...
##  $ xegvy: num  8 8 7 10 7 8 8 6 6 1 ...
##  $ y.ege: num  0 4 3 2 5 9 7 2 1 1 ...
##  $ yegvx: num  8 10 9 8 10 7 10 7 7 8 ...
```

```
dim(LetterRecognition)
```

```
## [1] 20000    17
```

1.3. Elección aleatoria de dos letras

```
set.seed("271")
(selected_letters=sample(c(LETTERS), 2, replace=FALSE))
```

```
## [1] "G" "N"
```

1.4. Extracción de los casos correspondientes a las letras seleccionadas

```
# Filtrado de las categorías seleccionadas (G y N)
data=LetterRecognition[which(LetterRecognition$lettr==selected_letters[1] |
                             LetterRecognition$lettr==selected_letters[2]), ]

# Forzamos a que la variable categórica "lettr"" tenga sólo 2 categorías posibles (G y N)
data$lettr=factor(data$lettr, levels = c("G","N"))
head(data)
```

```
##    lettr x.box y.box width high onpix x.bar y.bar x2bar y2bar xybar x2ybr
## 4      N     7    11     6    6     3     5     9     4     6     4     4
## 5      G     2     1     3    1     1     8     6     6     6     6     5
## 13     G     4     9     6    7     6     7     8     6     2     6     5
## 30     G     3     6     4    4     2     6     6     5     5     6     6
## 36     G     4     7     6    5     3     6     6     6     8     6     5
## 39     G     4     9     5    6     6     8     5     4     3     7     5
##    xy2br x.ege xegvy y.ege yegvx
## 4     10     6    10     2     8
## 5      9     1     7     5    10
## 13    11     4     8     7     8
## 30     9     2     8     4     8
## 36     9     3    10     4     8
## 39    11     6     8     5    11
```

```
dim(data)
```

```
## [1] 1556   17
```

```
str(data)
```

```
## 'data.frame':    1556 obs. of  17 variables:
##  $ lettr: Factor w/ 2 levels "G","N": 2 1 1 1 1 1 1 1 2 2 ...
##  $ x.box: num  7 2 4 3 4 4 4 1 3 3 ...
##  $ y.box: num  11 1 9 6 7 9 5 3 3 5 ...
##  $ width: num  6 3 6 4 6 5 5 2 3 4 ...
##  $ high : num  6 1 7 4 5 6 8 1 5 4 ...
##  $ onpix: num  3 1 6 2 3 6 3 1 2 2 ...
##  $ x.bar: num  5 8 7 6 6 8 7 7 7 7 ...
##  $ y.bar: num  9 6 8 6 6 5 6 7 7 8 ...
##  $ x2bar: num  4 6 6 5 6 4 8 5 13 5 ...
##  $ y2bar: num  6 6 2 5 8 3 8 6 2 4 ...
##  $ xybar: num  4 6 6 6 6 7 6 7 5 7 ...
##  $ x2ybr: num  4 5 5 6 5 5 6 5 6 7 ...
##  $ xy2br: num  10 9 11 9 9 11 10 10 8 7 ...
##  $ x.ege: num  6 1 4 2 3 6 1 1 5 6 ...
##  $ xegvy: num  10 7 8 8 10 8 8 9 8 9 ...
##  $ y.ege: num  2 5 7 4 4 5 6 3 0 2 ...
##  $ yegvx: num  8 10 8 8 8 11 11 9 8 5 ...
```

```
summary(data)
```

```
##  lettr       x.box            y.box            width       
##  G:773   Min.   : 1.000   Min.   : 0.000   Min.   : 1.000  
##  N:783   1st Qu.: 3.000   1st Qu.: 5.000   1st Qu.: 4.000  
##          Median : 4.000   Median : 7.000   Median : 5.000  
##          Mean   : 4.299   Mean   : 7.082   Mean   : 5.398  
##          3rd Qu.: 5.000   3rd Qu.:10.000   3rd Qu.: 7.000  
##          Max.   :11.000   Max.   :15.000   Max.   :14.000  
##       high            onpix            x.bar           y.bar      
##  Min.   :0.000   Min.   : 0.000   Min.   : 2.00   Min.   : 3.000  
##  1st Qu.:4.000   1st Qu.: 2.000   1st Qu.: 6.00   1st Qu.: 6.750  
##  Median :6.000   Median : 3.000   Median : 7.00   Median : 7.000  
##  Mean   :5.314   Mean   : 3.566   Mean   : 6.94   Mean   : 7.274  
##  3rd Qu.:7.000   3rd Qu.: 5.000   3rd Qu.: 7.00   3rd Qu.: 8.000  
##  Max.   :9.000   Max.   :12.000   Max.   :12.00   Max.   :13.000  
##      x2bar           y2bar            xybar           x2ybr      
##  Min.   : 2.0   Min.   : 0.000   Min.   : 3.000   Min.   : 0.000  
##  1st Qu.: 5.0   1st Qu.: 3.000   1st Qu.: 6.000   1st Qu.: 5.000  
##  Median : 6.0   Median : 5.000   Median : 7.000   Median : 6.000  
##  Mean   : 6.3   Mean   : 4.531   Mean   : 7.401   Mean   : 6.031  
##  3rd Qu.: 7.0   3rd Qu.: 6.000   3rd Qu.: 9.000   3rd Qu.: 7.000  
##  Max.   :15.0   Max.   :10.000   Max.   :14.000   Max.   :12.000  
##      xy2br           x.ege            xegvy           y.ege      
##  Min.   : 3.00   Min.   : 1.000   Min.   : 5.000   Min.   : 0.00  
##  1st Qu.: 7.00   1st Qu.: 2.000   1st Qu.: 8.000   1st Qu.: 1.00  
##  Median : 8.00   Median : 5.000   Median : 8.000   Median : 4.00  
##  Mean   : 8.45   Mean   : 4.149   Mean   : 8.409   Mean   : 3.29  
##  3rd Qu.:10.00   3rd Qu.: 6.000   3rd Qu.: 9.000   3rd Qu.: 5.00  
##  Max.   :14.00   Max.   :11.000   Max.   :13.000   Max.   :10.00
```

```
##      yegvx
##  Min.   : 0.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 8.123
##  3rd Qu.: 9.000
##  Max.   :14.000
```

1.5. División entrenamiento y test

Destinamos un 70% de los datos a entrenamiento y un 30% para test

```
n=nrow(data)
train.index=sort(sample(1:n, ceiling(0.7*n)))
train.data=data[train.index,]
test.data=data[-train.index,]
```

Conjunto de entrenamiento

```
dim(train.data)
```

```
## [1] 1090   17
```

```
summary(train.data)
```

```
##  lettr       x.box            y.box            width
##  G:540   Min.   : 1.000   Min.   : 0.000   Min.   : 1.000
##  N:550   1st Qu.: 3.000   1st Qu.: 4.000   1st Qu.: 4.000
##          Median : 4.000   Median : 7.000   Median : 5.000
##          Mean   : 4.234   Mean   : 6.958   Mean   : 5.344
##          3rd Qu.: 5.000   3rd Qu.: 9.000   3rd Qu.: 7.000
##          Max.   :11.000   Max.   :15.000   Max.   :12.000
##       high            onpix           x.bar            y.bar
##  Min.   :0.000   Min.   : 0.00   Min.   : 2.000   Min.   : 3.000
##  1st Qu.:4.000   1st Qu.: 2.00   1st Qu.: 6.000   1st Qu.: 7.000
##  Median :6.000   Median : 3.00   Median : 7.000   Median : 7.000
##  Mean   :5.264   Mean   : 3.55   Mean   : 6.917   Mean   : 7.273
##  3rd Qu.:7.000   3rd Qu.: 5.00   3rd Qu.: 7.000   3rd Qu.: 8.000
##  Max.   :9.000   Max.   :12.00   Max.   :12.000   Max.   :11.000
##      x2bar            y2bar            xybar            x2ybr
##  Min.   : 2.000   Min.   : 0.000   Min.   : 3.00   Min.   : 0.000
##  1st Qu.: 5.000   1st Qu.: 3.000   1st Qu.: 6.00   1st Qu.: 5.000
##  Median : 6.000   Median : 5.000   Median : 7.00   Median : 6.000
##  Mean   : 6.304   Mean   : 4.532   Mean   : 7.38   Mean   : 6.036
##  3rd Qu.: 7.000   3rd Qu.: 6.000   3rd Qu.: 9.00   3rd Qu.: 7.000
##  Max.   :15.000   Max.   :10.000   Max.   :14.00   Max.   :12.000
##      xy2br            x.ege            xegvy            y.ege
##  Min.   : 3.000   Min.   : 1.000   Min.   : 5.000   Min.   : 0.000
##  1st Qu.: 7.000   1st Qu.: 2.000   1st Qu.: 8.000   1st Qu.: 1.000
##  Median : 8.000   Median : 4.000   Median : 8.000   Median : 4.000
##  Mean   : 8.418   Mean   : 4.131   Mean   : 8.424   Mean   : 3.291
##  3rd Qu.:10.000   3rd Qu.: 6.000   3rd Qu.: 9.000   3rd Qu.: 5.000
##  Max.   :14.000   Max.   :10.000   Max.   :13.000   Max.   :10.000
```

```
##      yegvx
##  Min.   : 0.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 8.094
##  3rd Qu.: 9.000
##  Max.   :14.000
```

Conjunto de test

```
dim(test.data)
```

```
## [1] 466  17
```

```
summary(test.data)
```

```
##  lettr       x.box            y.box            width
##  G:233   Min.   : 1.000   Min.   : 0.000   Min.   : 1.000
##  N:233   1st Qu.: 3.000   1st Qu.: 5.000   1st Qu.: 4.000
##          Median : 4.000   Median : 8.000   Median : 5.000
##          Mean   : 4.451   Mean   : 7.371   Mean   : 5.526
##          3rd Qu.: 5.000   3rd Qu.:10.000   3rd Qu.: 7.000
##          Max.   :11.000   Max.   :15.000   Max.   :14.000
##       high            onpix            x.bar            y.bar
##  Min.   :0.000   Min.   : 0.000   Min.   : 3.000   Min.   : 3.000
##  1st Qu.:4.000   1st Qu.: 2.000   1st Qu.: 6.000   1st Qu.: 6.000
##  Median :6.000   Median : 3.000   Median : 7.000   Median : 7.000
##  Mean   :5.431   Mean   : 3.603   Mean   : 6.996   Mean   : 7.275
##  3rd Qu.:7.000   3rd Qu.: 5.000   3rd Qu.: 7.000   3rd Qu.: 8.000
##  Max.   :9.000   Max.   :12.000   Max.   :12.000   Max.   :13.000
##      x2bar            y2bar            xybar            x2ybr
##  Min.   : 2.000   Min.   : 0.00   Min.   : 3.000   Min.   : 0.000
##  1st Qu.: 4.000   1st Qu.: 3.00   1st Qu.: 6.000   1st Qu.: 6.000
##  Median : 6.000   Median : 5.00   Median : 7.000   Median : 6.000
##  Mean   : 6.292   Mean   : 4.53   Mean   : 7.451   Mean   : 6.021
##  3rd Qu.: 7.000   3rd Qu.: 6.00   3rd Qu.: 9.000   3rd Qu.: 7.000
##  Max.   :15.000   Max.   :10.00   Max.   :14.000   Max.   :12.000
##      xy2br            x.ege            xegvy            y.ege
##  Min.   : 3.000   Min.   : 1.000   Min.   : 6.000   Min.   : 0.00
##  1st Qu.: 7.000   1st Qu.: 2.000   1st Qu.: 8.000   1st Qu.: 1.00
##  Median : 8.000   Median : 5.000   Median : 8.000   Median : 4.00
##  Mean   : 8.524   Mean   : 4.191   Mean   : 8.373   Mean   : 3.29
##  3rd Qu.:10.000   3rd Qu.: 6.000   3rd Qu.: 9.000   3rd Qu.: 5.00
##  Max.   :13.000   Max.   :11.000   Max.   :12.000   Max.   :10.00
##      yegvx
##  Min.   : 3.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 8.189
##  3rd Qu.: 9.000
##  Max.   :14.000
```

2. Ramdon Forest

A continuación construiremos un modelo basado en Ramdon Forest.

2.1. Cálculo de valor óptimo de m

En cada nodo, se eligen aleatoriamente m $<$ p variables predictoras, para a continuación elegir la mejor división entre esas m variables.

Por defecto la librería *randomForest* construye 500 árboles y toma m=pˆ1/2 para problemas de clasificación, donde p es el número de variables predictoras.

Reducir m reduce tanto la correlación como la fuerza, por lo que el error aumenta. Este es el único parámetro a ajustar respecto al cual RandomForests es sensible, puede ser ajustado con procedimientos de validación cruzada o con ayuda de la función *tuneRF* de la librería *randomForest*

Calcularemos a continuación el valor de m por defecto

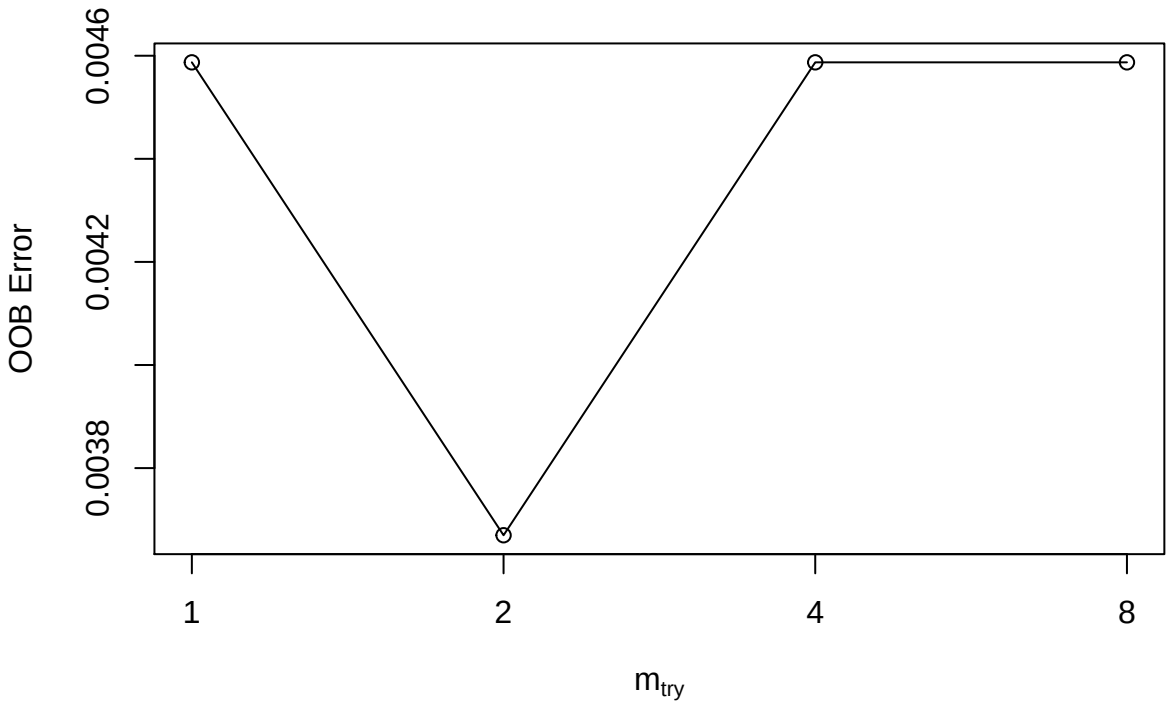
```
# Valor de m por defecto
mtry.default=floor(sqrt(dim(train.data)[2]))
mtry.default
```

```
## [1] 4
```

Ahora con *tuneRF* calcularemos el valor óptimo de m que minimiza el error *OOB*

```
letters.tuneRF=tuneRF(x=train.data[,-1], y=train.data[,1], stepFactor=2)
```

```
## mtry = 4  OOB error = 0.46%
## Searching left ...
## mtry = 2     OOB error = 0.37%
## 0.2 0.05
## mtry = 1     OOB error = 0.46%
## -0.25 0.05
## Searching right ...
## mtry = 8     OOB error = 0.46%
## -0.25 0.05
```




```
letters.tuneRF
```

```
##       mtry    OOBError
## 1.OOB    1 0.004587156
## 2.OOB    2 0.003669725
## 4.OOB    4 0.004587156
## 8.OOB    8 0.004587156
```

Se obtiene que el valor óptimo de m es 2, valor que no coincide con el que utiliza RandomForest por defecto (4), por este motivo habrá que especificar el valor obtenido en la construcción del modelo.

2.2. Contrucción del bosque aleatorio

```
RF<- randomForest(lettr ~ ., data=train.data, importance=TRUE, do.trace=FALSE, mtry=2)
RF
```

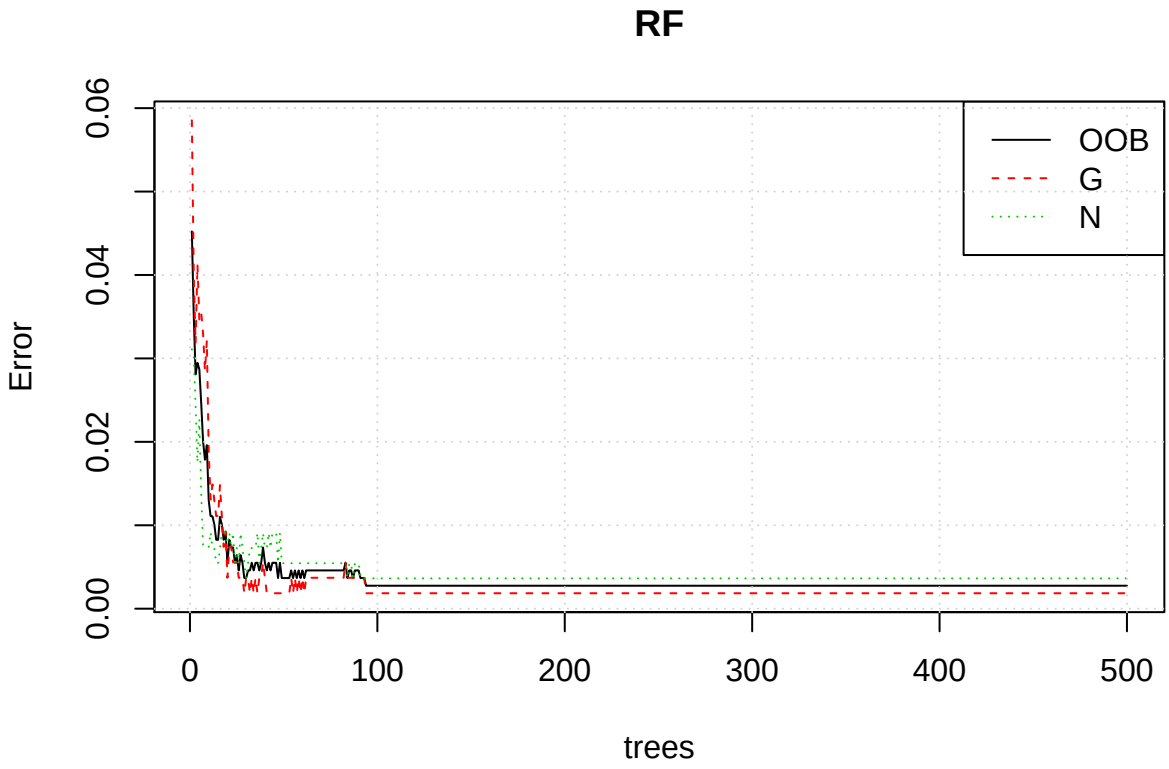
```
##
## Call:
##  randomForest(formula = lettr ~ ., data = train.data, importance = TRUE,      do.trace = FALSE, mtry
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.28%
## Confusion matrix:
##     G   N class.error
```

```
## G 539   1 0.001851852
## N   2 548 0.003636364
```

El OOB obtenido para el modelo en el conjunto de entrenamiento es igual a 0.28%, por tanto la tasa de acierto es de 99.72%.

2.3 Representación gráfica del error total y el de cada categoría

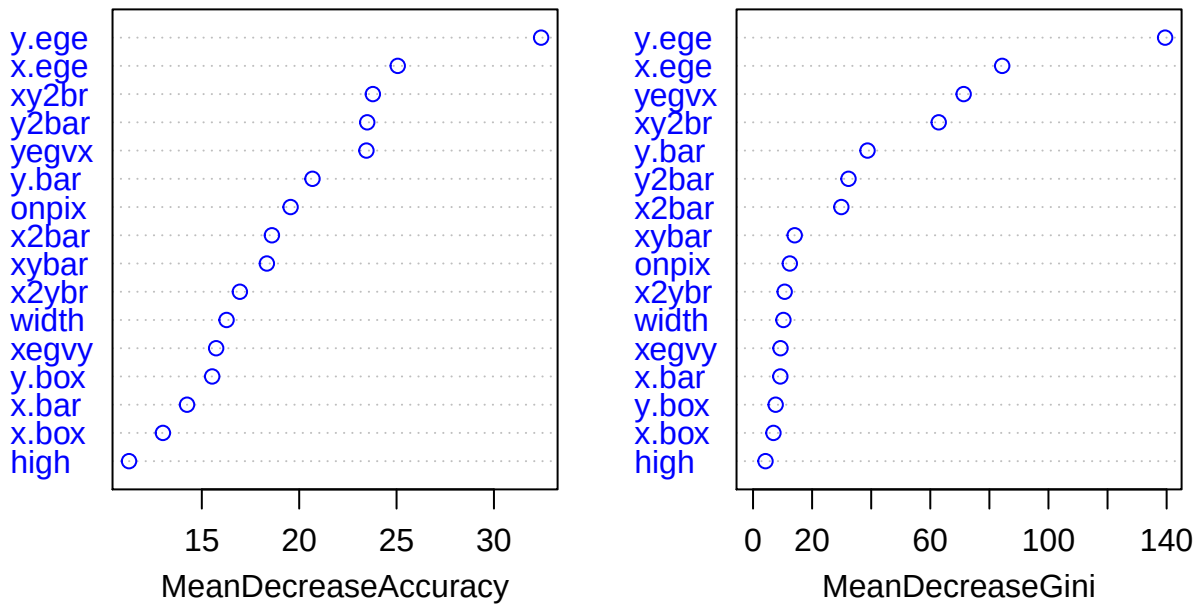
```
plot(RF)
legend("topright", col=1:3, lty=1:3, legend=c("OOB",levels(train.data$lettr)))
grid()
```



El error obtenido para cada categoría es muy similar.

2.4. Representación gráfica de la importancia de las variables

```
varImpPlot(RF, col="blue")
```



Se obtienen resultados similares con ambos criterios, en ambos casos las 2 variables que presentan mayor importancia son *y.ege* y *x.ege* y las que menos presentan menor importancia son *high* y *x.box*

2.5. Evaluación del rendimiento

A continuación calcularemos el error sobre el conjunto test, y así poder comparar con el modelo que construiremos en el siguiente apartado basado en el perceptrón multicapas

```
# Cálculo de las predicciones sobre el conjunto test
predictest<- predict(RF, newdata=test.data, type="response")
```

```
# Tabla de confusión
confusion.table.RF<-table(test.data$lettr, predictest)
confusion.table.RF
```

```
##    predictest
##       G   N
##   G 232   1
##   N   0 233
```

```
RF.group.G.accuracy=round((100*diag(prop.table(confusion.table.RF, 2)))[1], 3)
RF.group.N.accuracy=round((100*diag(prop.table(confusion.table.RF, 2)))[2], 3)
RF.total.accuracy=round(100*sum(diag(prop.table(confusion.table.RF))), 3)


cat(" Acierto grupo G  =\t",
    RF.group.G.accuracy,"\n",
    "Acierto grupo N  =\t",
```

```
    RF.group.N.accuracy,"\n",
    "Acierto total    =\t",
    RF.total.accuracy,"\n")
```

```
##  Acierto grupo G  =   100
##  Acierto grupo N  =   99.573
##  Acierto total    =   99.785
```

Con Random Forest hemos obtenido un modelo muy satisfactorio con una tasa de acierto bastante elevada que funciona muy bien con el conjunto de datos LetterRecognition y con las letras seleccionadas aleatoriamente.

3. Perceptrón multicapas

A continuación construiremos un modelo basado en el Perceptrón multicapa.

3.1. Tipificación de las variables predictoras

No conviene que las variables predictoras tengan valores dispares, por tanto es recomendable tipificar.

En primer lugar se tipificará el conjunto de entrenamiento usando la función *scale*, y después las observaciones test se transformarán con las medias y desviaciones típicas de los datos de entrenamiento, de este modo evitamos que el conjunto test intervenga en el entrenamiento del modelo

3.1.1 Normalización del conjunto de entrenamiento

```
zent<- scale(train.data[,-1], center=TRUE, scale=TRUE)
medias<- attr(zent, "scaled:center")
dt<- attr(zent, "scaled:scale")
```

3.1.2 Aplica mismo escalado sobre el conjunto test

```
ztest<- scale(test.data[,-1], medias, dt)
```

3.2 Construcción y ajuste del modelo

A continuación construiremos el modelo basado en el Perceptrón multicapa. Usaremos la función *tune* de la libreria *e1071* para encontrar los valores óptimos de *size* (tamaño de la capa oculta) y el parámetro *decay* (regularización L2 para evitar sobreajuste)

La función *tune* obtiene mediante validación cruzada los errores de clasificación de todas las combinaciones de valores de *size* y *decay* que se le pasan como entrada en la variable *ranges*.

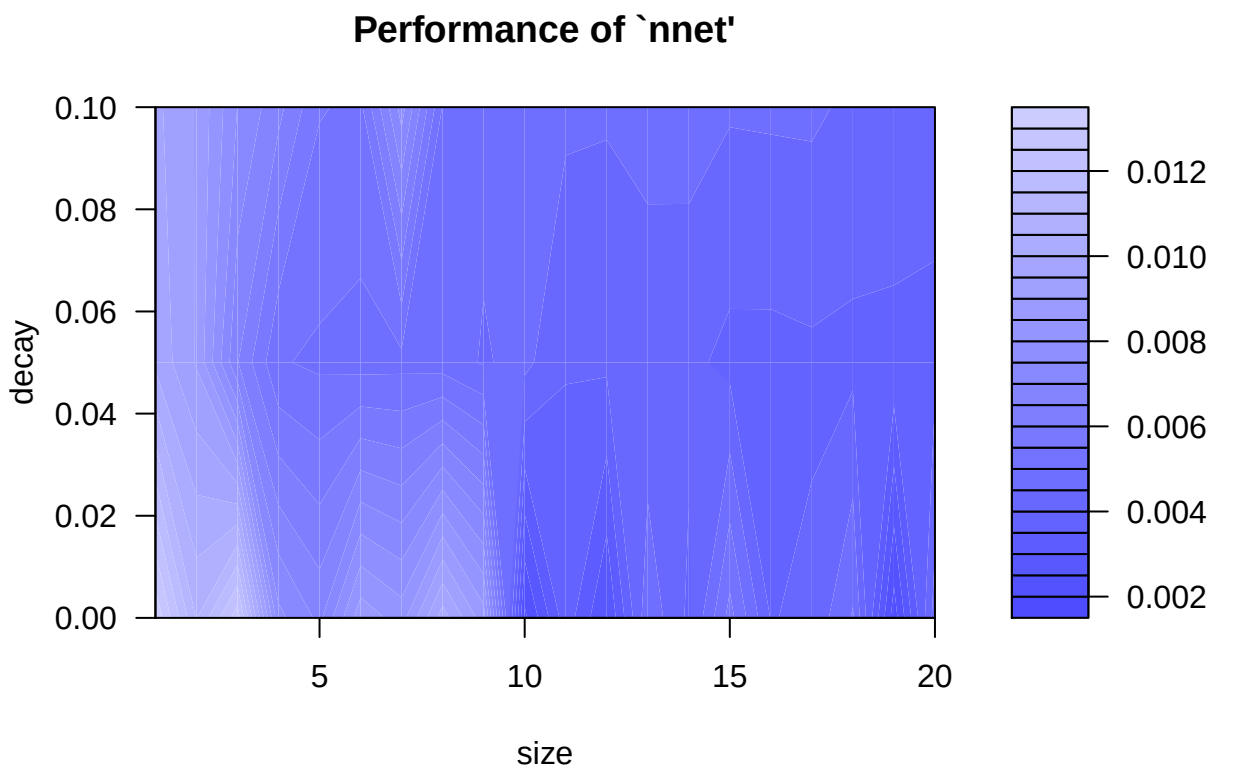
```
letters.tunePM<- tune(nnet, # modelo Percentron multicapa
                      zent, # datos de entrenamiento tipificados
                      as.numeric((train.data[,1]=="G")), # se codifica G como TRUE y N como FALSE
                      entropy=TRUE,  # recomendable en problemas de clasificación
                      ranges=list(size=1:20, decay=c(0, 0.05, 0.1)),
                      maxit=100,   # número máximo de iteraciones
                      trace=FALSE) # para que no imprima la traza de todo el proceso
```

```r
summary(letters.tunePM)
```

```
##
## Parameter tuning of 'nnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  size decay
##    19     0
##
## - best performance: 0.001738233
##
## - Detailed performance results:
##    size decay       error  dispersion
## 1     1  0.00 0.013404713 0.009321869
## 2     2  0.00 0.010975758 0.009209414
## 3     3  0.00 0.012795479 0.010218355
## 4     4  0.00 0.007647920 0.008585296
## 5     5  0.00 0.006875787 0.006990381
## 6     6  0.00 0.008821363 0.008753789
## 7     7  0.00 0.008275596 0.009085792
## 8     8  0.00 0.010252476 0.009701634
## 9     9  0.00 0.008731871 0.012479689
## 10   10  0.00 0.001883110 0.005726809
## 11   11  0.00 0.003789465 0.004199021
## 12   12  0.00 0.002471492 0.003168375
## 13   13  0.00 0.004861814 0.006736100
## 14   14  0.00 0.003896482 0.004649697
## 15   15  0.00 0.005687442 0.006770960
## 16   16  0.00 0.003973744 0.005502583
## 17   17  0.00 0.004109695 0.005823489
## 18   18  0.00 0.005052780 0.006442912
## 19   19  0.00 0.001738233 0.003776046
## 20   20  0.00 0.004731082 0.006555007
## 21    1  0.05 0.009891415 0.009892594
## 22    2  0.05 0.008951077 0.009714115
## 23    3  0.05 0.006528559 0.007656067
## 24    4  0.05 0.005048483 0.005696521
## 25    5  0.05 0.004904552 0.005328221
## 26    6  0.05 0.004809515 0.005716684
## 27    7  0.05 0.004847980 0.005328410
## 28    8  0.05 0.004760331 0.005327909
## 29    9  0.05 0.004455351 0.005023378
## 30   10  0.05 0.004641086 0.005090079
## 31   11  0.05 0.004019763 0.004333413
## 32   12  0.05 0.004092525 0.004439525
## 33   13  0.05 0.004066272 0.004333341
## 34   14  0.05 0.004134024 0.004360703
## 35   15  0.05 0.003852428 0.004077791
## 36   16  0.05 0.003847803 0.004067652
## 37   17  0.05 0.003905344 0.004107315
## 38   18  0.05 0.003868828 0.004130242
```

```
## 39   19  0.05 0.003864742 0.004016540
## 40   20  0.05 0.003870214 0.004040883
## 41    1  0.10 0.009621467 0.009423775
## 42    2  0.10 0.008959412 0.009234726
## 43    3  0.10 0.007489965 0.008010570
## 44    4  0.10 0.006656764 0.007191663
## 45    5  0.10 0.005538408 0.006390956
## 46    6  0.10 0.005387165 0.006082662
## 47    7  0.10 0.007723940 0.007362156
## 48    8  0.10 0.004972559 0.005350398
## 49    9  0.10 0.004635522 0.005075113
## 50   10  0.10 0.004707434 0.005004763
## 51   11  0.10 0.004612126 0.004926053
## 52   12  0.10 0.004559713 0.004884697
## 53   13  0.10 0.004765999 0.005157240
## 54   14  0.10 0.004723267 0.005145274
## 55   15  0.10 0.004555131 0.004863915
## 56   16  0.10 0.004578107 0.004940953
## 57   17  0.10 0.004592657 0.004867658
## 58   18  0.10 0.004395830 0.004698466
## 59   19  0.10 0.004312275 0.004670130
## 60   20  0.10 0.004195644 0.004419779
```

```
plot(letters.tunePM)
```



Obtenemos los valores de *size* y *decay* que minimizan el error de clasificación asi como el mejor modelo que está construido con estos parámetros óptimos.

```
# Valores óptimos de los parámetros
letters.tunePM$best.parameters
```

```
##    size decay
## 19   19     0
```

```
# Red con la mejor configuración
(PM=letters.tunePM$best.model)
```

```
## a 16-19-1 network with 343 weights
## options were - entropy fitting
```

El mejor modelo se obtiene con una red neuronal con 19 nodos en la capa oculta.

3.3 Evaluación del rendimiento

En primer lugar será necesario obtener las predicciones del conjunto test aplicando el modelo obtenido. Para obtener decisiones G/N, se deben comparar las probabilidades estimadas con un punto de corte (u), ya que la salida binaria está codificada con *0* (clase N) y *1* (categoría G):

Construiremos una función que traduzca si la clase seleccionada es G o N en función de la probabilidad estimada p y comparandola con un umbral (u) (si $p >= u$, decisión= G)

```
predclase<- function (p, u)
{
  ifelse(p>=u,"G","N")
}
PM.predict=predclase(predict(PM, ztest), 0.5)
```

Construimos la tabla de confusión

```
confusion.table.PM<-table(test.data$lettr, PM.predict)
confusion.table.PM
```

```
##    PM.predict
##       G   N
##   G 232   1
##   N   0 233
```

Calculamos el acierto por grupos y el acierto total

```
PM.group.G.accuracy=round((100*diag(prop.table(confusion.table.PM, 2)))[1], 3)
PM.group.N.accuracy=round((100*diag(prop.table(confusion.table.PM, 2)))[2], 3)
PM.total.accuracy=round(100*sum(diag(prop.table(confusion.table.PM))), 3)


cat(" Acierto grupo G  =\t",
    PM.group.G.accuracy,"\n",
    "Acierto grupo N  =\t",
    PM.group.N.accuracy,"\n",
    "Acierto total    =\t",
    PM.total.accuracy,"\n")
```

```
##  Acierto grupo G  =   100
##  Acierto grupo N  =   99.573
##  Acierto total    =   99.785
```

El modelo basado en el Perceptrón multicapa se ajusta muy bien a los datos y presenta una tasa de acierto muy alta.

4. Conclusiones

A continuación construiremos la tabla resumen con la tasa de acierto para ambos modelos

```
table_RF=c(RF.group.G.accuracy, RF.group.N.accuracy, RF.total.accuracy)
table_PM=c(PM.group.G.accuracy, PM.group.N.accuracy, PM.total.accuracy)

tabla_resumen = data.frame (round(rbind(table_RF, table_PM), 3),
                            row.names=c("Random Forest", "Perceptrón multicapa"))

print(knitr::kable(tabla_resumen, format = "pandoc",
                   col.names = c("Acierto G", "Acierto N", "Acierto total"),
                   align='c'))
```

```
##
##
##                         Acierto G    Acierto N    Acierto total
## ---------------------  -----------  -----------  ---------------
## Random Forest              100         99.573          99.785
## Perceptrón multicapa       100         99.573          99.785
```

A la vista de los resultados podemos concluir que ambos modelos se ajustan muy bien a los datos y que presentan un tasa de acierto alta. No existe diferencia en cuanto a tasa de acierto entre ambos modelos pero quizá desde el punto de vista computacional el modelo Random Forest tiene mejor rendimiento, el sistema ha tardado menos en construirlo y ajustarlo.