Proyecto final - Aprendizaje Automático

Francisco Javier Caracuel Beltrán

09 de Junio de 2017

A continuación, se detalla el resultado del proyecto final de Aprendizaje Automático. Este documento incluye todo el código utilizado en el archivo *proyecto.R*, así como los comentarios añadidos durante su creación.

Al entregarse el fichero *proyecto.Rmd*, no se incluyen puntos de parada en el código fuente que se encuentra en el fichero *proyecto.R*.

Se establece el directorio de trabajo

```
setwd("/mnt/A0DADA47DADA18FC/Fran/Universidad/3º/2 Cuatrimestre/AA/Mis prácticas/Proyecto/data")
```

Se establece la semilla para la generación de datos aleatorios

```
set.seed(1822)
```

Número de decimales que tendrán los porcentajes de error en la salida

```
perDec = 4
```

Ajuste de Modelos NO-Lineales

Para poder ejecutar el script sin problemas deben encontrarse instaladas las siguientes librerías con sus correspondientes dependencias:

- *ggplot2*
- *glmnet*
- *plyr*
- *e1071*
- *party*
- *randomForest*
- *neuralnet*

Problema de clasificación: 13. Human Activity Recognition Using Smartphones

Para la elaboración de este proyecto, se utiliza la estructura descrita en el *trabajo 3* de esta misma asignatura.

1. Comprender el problema a resolver.

La computación basada en el ser humano es un campo emergente de investigación cuyo objetivo es comprender el comportamiento humano e integrar a los usuarios y su contexto social con los sistemas informáticos. Una de las aplicaciones más recientes, desafiantes y atractivas en este campo consiste en detectar el movimiento de los humanos usando teléfonos móviles para recopilar información sobre las acciones de las personas.

Éste es el problema a resolver, partiendo de una base de datos que describe el *Reconocimiento de la Actividad* de cada persona, construida a partir de *30* voluntarios que realizan actividades cotidianas mientras llevan un teléfono móvil en su cintura, que cuenta con sensores de movimiento que ofrecen esta información se pretende aplicar una serie de modelos lineales y no lineales que consigan clasificar las distintas actividades que se han registrado.

Los experimentos se han llevado a cabo con un grupo de *30* voluntarios de entre *19* y *48* años. Cada persona realizó *seis* actividades: *caminar*, *subir escaleras*, *bajar escaleras*, *sentarse*, *levantarse* o *tumbarse*.

El teléfono usado es un *Samsung Galaxy SII*, del cual usan el acelerómetro y giroscopio que tienen incorporado.

Referencias:

- http://rstudio-pubs-static.s3.amazonaws.com/100601_62cc5079d5514969a72c34d3c8228a84.html

2. Los conjuntos de training, validación y test usados en su caso.

La base de datos cuenta con *561* características y *10.299* instancias obtenidas de los teléfonos móviles de los voluntarios.

Se puede descargar a través del siguiente enlace: https://archive.ics.uci.edu/ml/machine-learning-databases/00240/UCI%20HAR%20Dataset.zip

El archivo descargado ya cuenta con los conjuntos de training y test debidamente separados, por lo que solo es necesario su lectura y copiado en memoria en sus correspondientes variables.

El conjunto de training dispone de *7.352* instancias, mientras que el conjunto de test se encuentra con *2.947* instancias, lo que supone el *71,39%* y *28,61%* de las muestras respectivamente.

Se leen todas las muestras de la base de datos, guardando cada una en su lugar correspondiente. Para hacerlo se hace uso de un pequeño tutorial que se ha encontrado muy conveniente. Su referencia se adjunta al final de este apartado.

```
# Carpeta donde se encuentran las distintas muestras
folder = "UCI HAR Dataset"

# El fichero activity_labels.txt contiene el identificador y la etiqueta de
# cada muestra que se puede encontrar en la base de datos.
# Se lee el fichero y se guarda la descripción de ese fichero.
temp = read.table(paste(folder, "activity_labels.txt", sep="/"),
                  sep = "")

# El primer conjunto contiene los identificadores, el segundo la descripción,
# que es la que se quiere guardar
activityLabels = as.character(temp$V2)

# Se repite la operación con los atributos de cada muestra. Se lee del fichero
# features.txt y se guarda el segundo conjunto que contiene la descripción
# de cada atributo o característica.
temp = read.table(paste(folder, "features.txt", sep="/"), sep = "")

attributeNames = temp$V2

# En las operaciones realizadas posteriormente, se ha detectado que existen
# nombres de atributos repetidos, por lo que se deben renombrar para que
# sean únicos.
# Se muestra una lista con el número de los atributos repetidos
which(duplicated(attributeNames))
```

```
##  [1] 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333
## [18] 334 335 336 337 338 339 340 341 342 343 344 396 397 398 399 400 401
## [35] 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418
## [52] 419 420 421 422 423 475 476 477 478 479 480 481 482 483 484 485 486
## [69] 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502
```

```
# Estas dos sentencias renombran automáticamente los nombres repetidos de los
# atributos, concatenando al final un número (1,2,3,etc) para hacerlos únicos
attributeNames = make.names(attributeNames)
attributeNames = make.unique(attributeNames)

# Se muestra la lista donde ya no aparece ningún atributo repetido
which(duplicated(attributeNames))
```

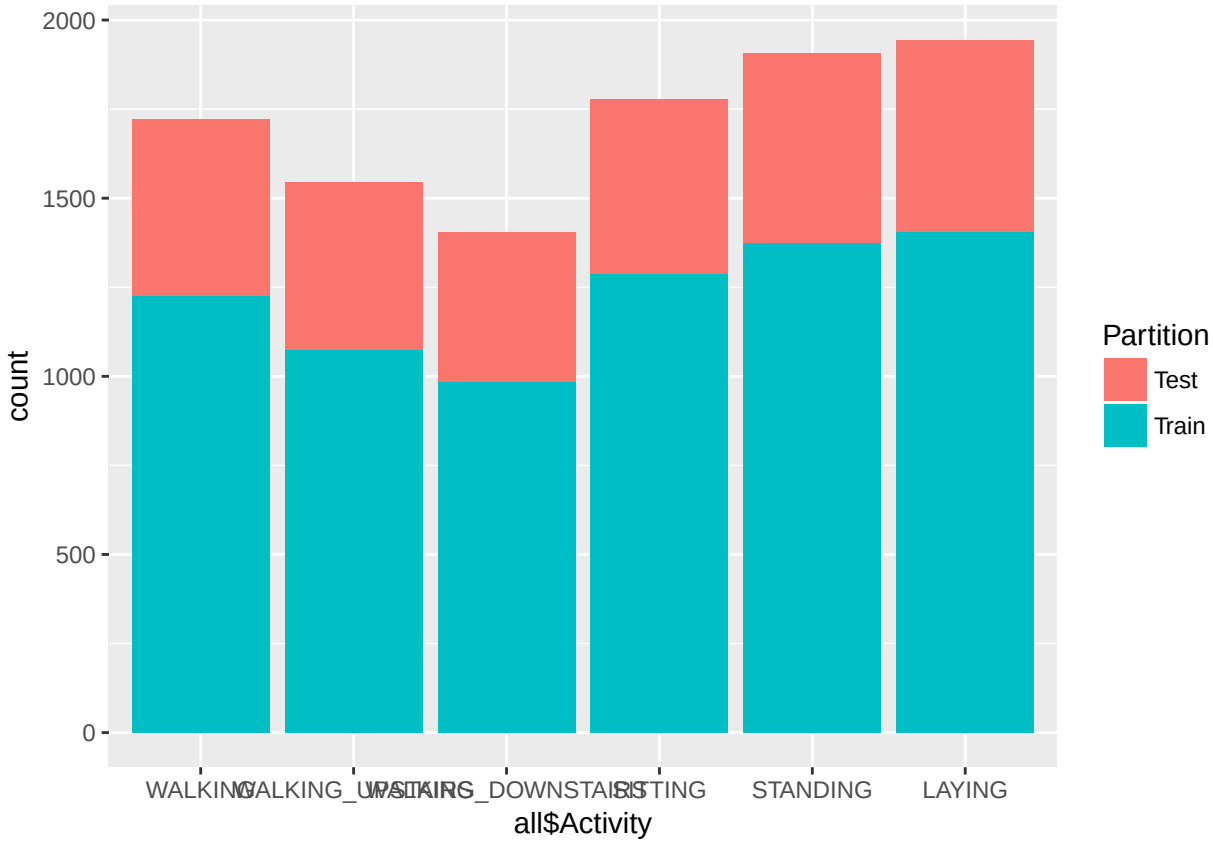
```
## integer(0)
```

```
# Se guardan las muestras de entrenamiento
train.x = read.table(paste(folder, "train/X_train.txt", sep="/"), sep = "")

# Al leer las muestras de un fichero en el que no se indica el nombre de cada
# atributo o característica, se guarda con un nombre incremental. Se le asigna
# al conjunto de entrenamiento el nombre de cada atributo (guardado
# anteriormente en "attributeNames")
colnames(train.x) = attributeNames

# Se guarda la clase que tiene cada muestra de entrenamiento
train.y = read.table(paste(folder, "train/y_train.txt", sep="/"), sep = "")

# Como tampoco tiene nombre la clase leída, se le asigna la que se desee
colnames(train.y) = "Activity"

# Se convierte en factor las etiquetas de las muestras para trabajar mejor con
# ellas
train.y$Activity = as.factor(train.y$Activity)

# Se relacionan las muestras con su respectiva clase
levels(train.y$Activity) = activityLabels

# En principio no es relevante, pero se guardan también los voluntarios que
# capturaron las muestras
#train.subjects = read.table(paste(folder, "train/subject_train.txt", sep="/"),
#                   sep = "")

# Se le asigna un nombre a la columna con los voluntarios/sujetos
#names(train.subjects) = "Subject"

# Se convierte en factor los sujetos que tomaron las muestras para trabajar
# mejor con ellos
#train.subjects$Subject = as.factor(train.subjects$Subject)

# Cuando ya están todos los subconjuntos de entrenamiento creados, se
# agrupan para los futuros cálculos
train = cbind(train.x, train.y)

# La operación que se ha realizado con las muestras de entrenamiento se repiten
# con las muestras de test
test.x = read.table(paste(folder, "test/X_test.txt", sep="/"), sep = "")
colnames(test.x) = attributeNames

test.y = read.table(paste(folder, "test/y_test.txt", sep="/"), sep = "")
```

```r
colnames(test.y) = "Activity"
test.y$Activity = as.factor(test.y$Activity)
levels(test.y$Activity) = activityLabels

#test.subjects = read.table(paste(folder, "test/subject_test.txt", sep="/"),
#                           sep = "")
#names(test.subjects) = "Subject"
#test.subjects$Subject = as.factor(test.subjects$Subject)

test = cbind(test.x, test.y)

# Se comprueban que los datos leídos sean correctos (al menos en apariencia)
train$Partition = "Train"
test$Partition = "Test"

# Se unen todas las muestras para mostrarlas en el gráfico
all = rbind(train, test)
```

Se visualizan el número de muestras que existe de cada tipo, visualizando las que pertenecen a train y las que pertenecen a test.

```r
library(ggplot2)

qplot(data=all, x=all$Activity, fill=Partition)
```



Referencias:

- https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

- http://rstudio-pubs-static.s3.amazonaws.com/100601_62cc5079d5514969a72c34d3c8228a84.html
- https://stackoverflow.com/questions/18766700/r-rename-duplicate-col-and-rownames-subindexing

3. Preprocesado los datos

Las características seleccionadas para esta base de datos provienen de las señales 3-axiales del acelerómetro y giroscopio. Estas señales se capturaron a una velocidad constante de 50 Hz, después se filtraron usando un filtro mediano y, posteriormente, de nuevo con otro filtro Butterworth de paso bajo con una frecuencia de 20 Hz para eliminar ruido. De igual manera, la señal de aceleración fue separada en señales de aceleracion de cuerpo y gravedad a 0,3 Hz. Se aplicaron continuamente más procesos (ver referencia al final de este apartado para más información) para transformar las señales, dando lugar a las siguientes señales finales:

- *tBodyAcc-XYZ*
- *tGravityAcc-XYZ*
- *tBodyAccJerk-XYZ*
- *tBodyGyro-XYZ*
- *tBodyGyroJerk-XYZ*
- *tBodyAccMag*
- *tGravityAccMag*
- *tBodyAccJerkMag*
- *tBodyGyroMag*
- *tBodyGyroJerkMag*
- *fBodyAcc-XYZ*
- *fBodyAccJerk-XYZ*
- *fBodyGyro-XYZ*
- *fBodyAccMag*
- *fBodyAccJerkMag*
- *fBodyGyroMag*
- *fBodyGyroJerkMag*

Con todas estas señales, algunas de las variables que se estimaron fueron:

- *media*
- *desviación típica*
- *desviación media absoluta*
- *máximo*
- *mínimo*
- *rango intercuartílico*
- *entropía*
- *ángulo entre vectores, etc.*

Cuando ya se han realizado todas las operaciones correspondientes, se ofrece una base de datos con *561* atributos.

Los atributos han sido normalizados y redondeados en el intervalo *[-1, 1]* y todos han sido preprocesados, no siendo necesario ningún tratamiento adicional.

Se comprueba si existe algún valor perdido. Para eso se utiliza la función *which()* y la función *is.na()*, que nos indican, respectivamente, la posición en la que ocurre algún hecho y si alguna posición no es correcta.

```
which(is.na(all))
```

```
## integer(0)
```

El valor devuelto es *0*, por lo que no existen valores perdidos en ningún conjunto

Se ha querido reducir la dimensionalidad ya que *561* atributos son bastantes y probablemente no sean necesarios tantos para representar la muestra. La base de datos que se tiene es muy pesada y no se ha

conseguido obtener una fórmula que represente la mayor parte de la muestra al no terminar su cómputo en un día completo. Debido a la falta de tiempo se ha decidido utilizar todos los atributos en los cálculos del proyecto.

Referencias:

- UCI HAR Dataset/features_info.txt
- UCI HAR Dataset/README.txt

4. Selección de clases de funciones a usar.

En este proyecto se va a comprobar el error de test con las siguientes técnicas:

- *Regresión Lineal*: al igual que en el trabajo 3, se usará la librería glmnet para aplicar regresión lineal y se aplicará generalización con regresión Ridge y Lasso.
- *Máquinas de Soporte de Vectores* (*SVM*)
- *Boosting*
- *Random Forest*
- *Redes Neuronales*

El objetivo final será encontrar con qué modelo se obtiene el menor error posible.

5. Discutir la necesidad de regularización y en su caso la función usada.

La primera prueba se hará con regresión lineal. En la segunda prueba, que será donde se haga la regularización, se usará la función *cv.glmnet()*, que realiza una regresión logística aplicando generalización y haciendo cross-validation. La segunda prueba a su vez, se divide en dos, en las que en una se hará la regularización *Ridge* y en otra la regularización *Lasso*.

6. Definir los modelos a usar y estimar sus parámetros e hyperparámetros.

Regresión Lineal

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-10
```

```
# Se ajusta el modelo con regresión lineal. Los datos de los que aprende y
# sus etiquetas deben coincidir, por lo que se ajusta como una matriz los
# datos de entrenamiento.
# Como la variable dependiente es nominal con más de dos niveles (en
# concreto 6 clases), se debe utilizar la familia multinomial.
# Se quiere aplicar regresión lineal de la misma manera que si se utilizara
# lm(), por lo que el parámetro lambda debe ser establecido a 0.
har.lm = glmnet(as.matrix(train.x), train.y$Activity,
                family='multinomial',
                lambda=0)

# Se predice el resultado que tendrán las muestras de train con la regresión
# lineal generada con esas mismas muestras
har.lm.predict <- predict(har.lm, as.matrix(train.x))
```

```r
# Se han predicho los valores de cada muestra de test, pero se deben convertir
# a una etiqueta que se pueda comparar con la clase a la que corresponden
# realmente.
# Se utiliza apply para recorrer todos los valores predichos y convertirlos en
# su correspondiente clase
har.lm.predict.label = apply(har.lm.predict, 1,
                             function(value){which(value==max(value))})

# Solo queda convertir la clase en su etiqueta correspondiente, por lo que se
# hace uso de la función mapvalues de la librería plyr, que modifica unos datos
# de una estructura dada, indicando los originales y los que se quiere que
# aparezcan
library(plyr)

# El intervalo de valores originales debe comenzar desde el mínimo hasta el
# máximo
har.lm.predict.label = mapvalues(har.lm.predict.label,
                                 from = c(min(har.lm.predict.label):max(har.lm.predict.label)),
                                 to = activityLabels)

# Se calcula el error, comparando los generados con los reales y obteniendo
# la media de los que no coinciden
har.lm.mean.train = mean(har.lm.predict.label != as.character(train.y$Activity))

# Se muestra el resultado obtenido
print(paste("Etrain calculado con Regresión Lineal: ",
            round(har.lm.mean.train*100, digits = perDec), "%"))
```

```
## [1] "Etrain calculado con Regresión Lineal:  0 %"
```

```r
# Se predice el resultado que tendrán las muestras de test con la regresión
# lineal generada con las muestras de train
har.lm.predict <- predict(har.lm, as.matrix(test.x))

# Se han predicho los valores de cada muestra de test, pero se deben convertir
# a una etiqueta que se pueda comparar con la clase a la que corresponden
# realmente.
# Se utiliza apply para recorrer todos los valores predichos y convertirlos en
# su correspondiente clase
har.lm.predict.label = apply(har.lm.predict, 1,
                             function(value){which(value==max(value))})

# El intervalo de valores originales debe comenzar desde el mínimo hasta el
# máximo
har.lm.predict.label = mapvalues(har.lm.predict.label,
                                 from = c(min(har.lm.predict.label):max(har.lm.predict.label)),
                                 to = activityLabels)

# Se calcula el error, comparando los generados con los reales y obteniendo
# la media de los que no coinciden
har.lm.mean.test = mean(har.lm.predict.label != as.character(test.y$Activity))

# Se muestra el resultado obtenido
print(paste("Etest calculado con Regresión Lineal: ",
```

```
            round(har.lm.mean.test*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con Regresión Lineal:  5.565 %"
```

Regresión Logística con regularización Ridge

```
# Se repite el proceso haciendo regresión logística con regularización Ridge.
# Para indicar la regularización Ridge, se envía el parámetro alpha = 0.
har.glm.ridge = glmnet(as.matrix(train.x), train.y$Activity,
                       family='multinomial',
                       alpha=0)

# Se calcula eTrain
har.glm.ridge.predict <- predict(har.glm.ridge, as.matrix(train.x))

har.glm.ridge.predict.label = apply(har.glm.ridge.predict, 1,
                                    function(value){which(value==max(value))})

# Se obtiene un valor no deseado y se procede a su eliminación
unique(har.glm.ridge.predict.label)
```

```
## [1] 599 598 600 595 597 596 566
```

```
which(har.glm.ridge.predict.label==566)
```

```
## [1] 565
```

```
har.glm.ridge.predict.label[565] = 595

har.glm.ridge.predict.label = mapvalues(har.glm.ridge.predict.label,
                                        from = c(min(har.glm.ridge.predict.label):max(har.glm.ridge.pred
                                        to = activityLabels)

har.glm.ridge.mean.train = mean(har.glm.ridge.predict.label != as.character(train.y$Activity))

print(paste("Etrain calculado con Regresión Logística con regularización Ridge: ",
            round(har.glm.ridge.mean.train*100, digits = perDec), "%"))
```

```
## [1] "Etrain calculado con Regresión Logística con regularización Ridge:  2.2851 %"
```

```
# Se calcula eTest
har.glm.ridge.predict <- predict(har.glm.ridge, as.matrix(test.x))

har.glm.ridge.predict.label = apply(har.glm.ridge.predict, 1,
                                    function(value){which(value==max(value))})

har.glm.ridge.predict.label = mapvalues(har.glm.ridge.predict.label,
                                        from = c(min(har.glm.ridge.predict.label):max(har.glm.ridge.pred
                                        to = activityLabels)

har.glm.ridge.mean.test = mean(har.glm.ridge.predict.label != as.character(test.y$Activity))

print(paste("Etest calculado con Regresión Logística con regularización Ridge: ",
            round(har.glm.ridge.mean.test*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con Regresión Logística con regularización Ridge:  5.7686 %"
```

Regresión Logística con Regularización Lasso

```
# Se repite el proceso haciendo regresión logística con regularización Lasso.
# Para indicar la regularización Lasso, se envía el parámetro alpha = 1.
har.glm.lasso = cv.glmnet(as.matrix(train.x), train.y$Activity,
                          family='multinomial',
                          alpha=1)

# Se calcula eTrain
har.glm.lasso.predict <- predict(har.glm.lasso, newx=as.matrix(train.x))

har.glm.lasso.predict.label = apply(har.glm.lasso.predict, 1,
                                    function(value){which(value==max(value))})

har.glm.lasso.predict.label = mapvalues(har.glm.lasso.predict.label,
                                        from = c(min(har.glm.lasso.predict.label):max(har.glm.lasso.pred
                                        to = activityLabels)

har.glm.lasso.mean.train = mean(har.glm.lasso.predict.label != as.character(train.y$Activity))

print(paste("Etest calculado con Regresión Logística con regularización Lasso: ",
            round(har.glm.lasso.mean.train*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con Regresión Logística con regularización Lasso:  0.6801 %"
```

```
# Se calcula eTest
har.glm.lasso.predict <- predict(har.glm.lasso, newx=as.matrix(test.x))

har.glm.lasso.predict.label = apply(har.glm.lasso.predict, 1,
                                    function(value){which(value==max(value))})

har.glm.lasso.predict.label = mapvalues(har.glm.lasso.predict.label,
                                        from = c(min(har.glm.lasso.predict.label):max(har.glm.lasso.pred
                                        to = activityLabels)

har.glm.lasso.mean.test = mean(har.glm.lasso.predict.label != as.character(test.y$Activity))

print(paste("Etest calculado con Regresión Logística con regularización Lasso: ",
            round(har.glm.lasso.mean.test*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con Regresión Logística con regularización Lasso:  5.056 %"
```

Referencias:

- http://ricardoscr.github.io/how-to-use-ridge-and-lasso-in-r.html

SVM

```
# Para la Máquina de Soporte de Vectores (SVM) se hace uso de la librería
# e1071.
library("e1071")

# Por defecto, según la documentación de esta librería, el núcleo que se
# utiliza es el que se pide, RBF-Gaussiano.
# Se necesita encontrar el mejor parámetro libre hasta una precisión de
# 2 cifras, y se pueden implementar de dos modos distintos. El primero
```

```
# consiste en realizar una búsqueda iterativa modificando los valores
# necesarios. El segundo consiste en utilizar la función tune, a la que se
# le puede indicar un listado de valores que se quieren comprobar y cuando
# termine el proceso devolverá los mejores parámetros (costo y gamma) que se
# ajustan al modelo.
# Se va a utilizar este segundo método, por lo que se le envían como parámetros
# el sistema que se quiere ajustar, los datos de entrenamiento como una matriz
# paa hacerla compatible a la función y las etiquetas de la clase de cada
# muestra.
# Los valores con los que se va a probar son 1, 10 y 100 para el coste y
# 0.01, 0.1, 1 y 10 para gamma (para cumplir con la condición de que como
# máximo se permite una precisión de 2 cifras).
# Debido a que los cálculos necesitan demasiado tiempo, se ha lanzado solo una
# vez, obteniendo el resultado, por lo que para generar la documentación se
# mantiene la instrucción comentada.

#svm_tune <- tune(svm, train.x=as.matrix(train.x), train.y=train.y$Activity,
#                 ranges=list(cost=list(1, 10, 100), gamma=10^(-2:1)))

# Cuando ya se ha obtenido el resultado, se pueden comprobar los parámetros que
# mejor se adaptan al modelo:

# svm_tune$best.parameters

# Parameter tuning of svm:
#
#  - best parameters:
#      cost gamma
#       10  0.01

# Queda establecido que el coste será 10 y gamma 0.01.
# El siguiente paso es adaptar la estructura de datos a la que necesita la
# función svm, por lo que se crea un dataframe estableciendo como parámetro
# x los datos de la muestra y como parámetro y las etiquetas de cada muestra
# convirtiéndolo a factor para que pueda utilizarlos svm al crear el modelo
df.train<- data.frame(x=train.x, y=as.factor(train.y$Activity))
df.test <- data.frame(x=test.x, y=as.factor(test.y$Activity))

# Se debe crear el modelo intentando predecir la etiqueta de la clase de
# cada muestra, por lo que se llama a la función svm prediciendo y, que
# contiene la etiqueta
har.svm = svm(y ~ ., data=df.train, cost=10, gamma=0.01)

# Cuando ya se ha ajustado el modelo, se predice el resultado que tendrá
# con los dos conjuntos (train y test)
har.svm.predict.train <- predict(har.svm, df.train)
har.svm.predict.test <- predict(har.svm, df.test)

# Se calcula la media entre los valores predichos y los reales
har.svm.mean.train = mean(har.svm.predict.train != df.train[, 'y'])
har.svm.mean.test = mean(har.svm.predict.test != df.test[, 'y'])

print(paste("Etrain calculado con SVM: ",
```

```
            round(har.svm.mean.train*100, digits = perDec), "%"))
```

```
## [1] "Etrain calculado con SVM:  0 %"
```

```
print(paste("Etest calculado con SVM: ",
            round(har.svm.mean.test*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con SVM:  10.4852 %"
```

Referencias:

- http://www.louisaslett.com/Courses/Data_Mining/ST4003-Lab7-Introduction_to_Support_Vector_Machines.pdf
- http://rischanlab.github.io/SVM.html

Boosting

```
# Se quiere hacer boosting utilizando AdaBoost y para ellos se utiliza la
# librería adabag
require(adabag)
```

```
## Loading required package: adabag
```

```
## Loading required package: rpart
```

```
## Loading required package: mlbench
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
# Se quiere calcular el modelo prediciendo y (las etiquetas de las clases).
# La fórmula, como hasta ahora, será y (las etiquetas) utilizando todos los
# atributos de la muestra.
# El parámetro boos se establece a TRUE para indicar que cada muestra debe
# utilizar los pesos de cada iteración.
# mfinal se establece a 10, indicando el número de iteraciones que se va
# a hacer uso con el boosting.
har.boosting = boosting(y~., data=df.train, boos=TRUE, mfinal=10)

# Como en los métodos anteriores, se predice el resultado de ambos conjuntos
# según el modelo entrenado
har.boosting.predict.train = predict.boosting(har.boosting, newdata=df.train)
har.boosting.predict.test = predict.boosting(har.boosting, newdata=df.test)

# Una vez que se ha predicho con las muestras de train y test, se pueden
# comprobar los resultados u ver la matriz de confusion, el error, la clase
# que predice, la probabilidad de cada resultado, etc,
#har.boosting.train$confusion
#har.boosting.train$error
#har.boosting.train$class
#har.boosting.train$formula
#har.boosting.train$votes
#har.boosting.train$prob

# Se puede obtener el error utilizando har.boosting.train$error y aparece
# directamente, pero se va a utilizar el procedimiento que se ha seguido en
# los anteriores métodos
```

```
har.boosting.mean.train = mean(har.boosting.predict.train$class != df.train[,'y'])
har.boosting.mean.test = mean(har.boosting.predict.test$class != df.test[,'y'])

print(paste("Etrain calculado con AdaBoost: ",
            round(har.boosting.mean.train*100, digits = perDec), "%"))
```

```
## [1] "Etrain calculado con AdaBoost:  0.8977 %"
```

```
print(paste("Etest calculado con AdaBoost: ",
            round(har.boosting.mean.test*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con AdaBoost:  8.6529 %"
```

Referencias:

- https://cran.rstudio.com/web/packages/adabag/adabag.pdf

Random Forest

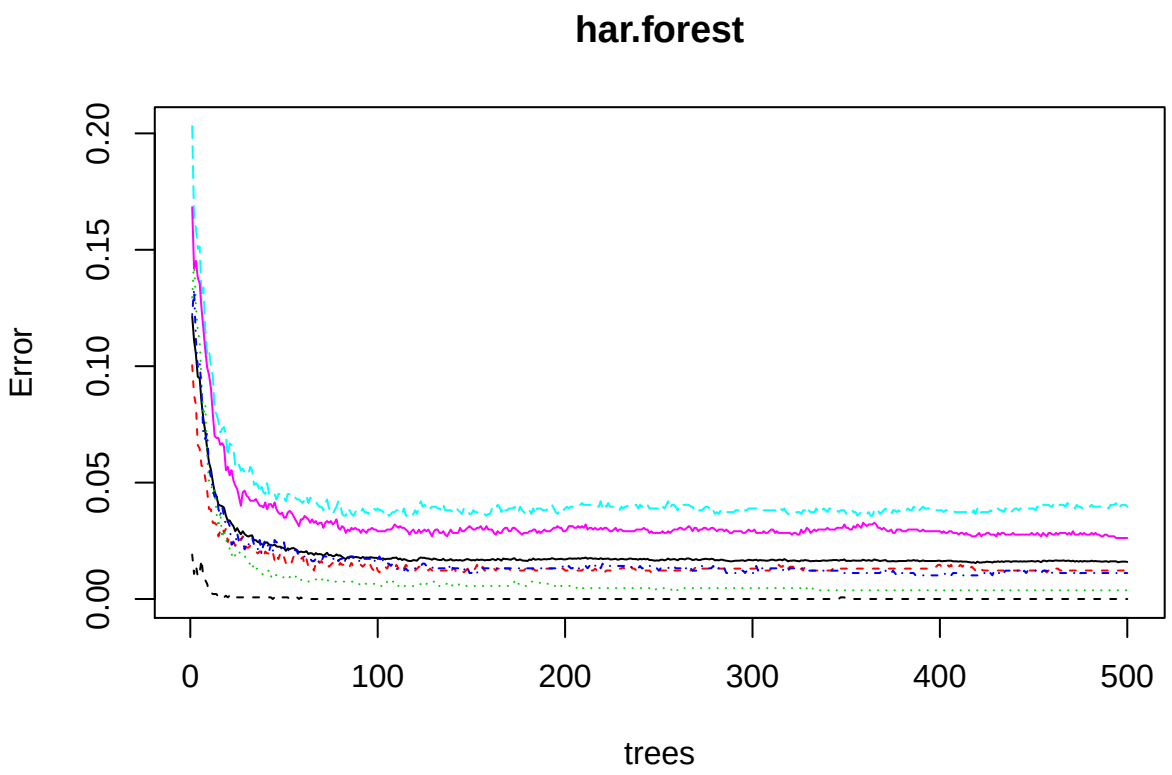
```
# Para utilizar Random Forest se hace uso de las siguientes librerías:
library(party)
```

```
## Loading required package: grid
## Loading required package: mvtnorm
## Loading required package: modeltools
## Loading required package: stats4
##
## Attaching package: 'modeltools'
## The following object is masked from 'package:plyr':
##
##     empty
## Loading required package: strucchange
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
## Loading required package: sandwich
```

```
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
# Se predice la etiqueta de las muestras utilizando todos los atributos y
# se indica que el número de árboles que se van a utilizar sean 500. Cuando
# se haya creado el modelo se comprobará su progresión.
# El parámetro importance indica que se debe evaluar la importancia de
# cada predictor
har.forest <- randomForest(y ~., data = df.train, ntree=500, importance=T)

# Para comprobar la progresión que ha tenido en el entrenamiento, se muestra
# su gráfica
plot(har.forest)
```




```r
# Se puede ver como realmente son necesarios menos de 100 árboles para aprender
# el modelo actual al estabilizarse el error de la muestra de entrenamiento

# Se predicen los resultados que deben tener tanto el conjunto de train como
# de test
har.forest.predict.train = predict(har.forest, df.train)
har.forest.predict.test = predict(har.forest, df.test)

# Se calcula la media de error de los dos conjuntos
har.forest.mean.train = mean(har.forest.predict.train != df.train[,'y'])
har.forest.mean.test = mean(har.forest.predict.test != df.test[,'y'])

print(paste("Etrain calculado con Random Forest: ",
            round(har.boosting.mean.train*100, digits = perDec), "%"))
```

```
## [1] "Etrain calculado con Random Forest:  0.8977 %"
```

```r
print(paste("Etest calculado con Random Forest: ",
            round(har.boosting.mean.test*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con Random Forest:  8.6529 %"
```

Referencias:

- https://cran.r-project.org/web/packages/randomForest/randomForest.pdf
- https://www.tutorialspoint.com/r/r_random_forest.htm
- http://dni-institute.in/blogs/random-forest-using-r-step-by-step-tutorial/

Redes neuronales

```r
# Para entrenar los datos de la muestra con redes neuronales se hace uso
# de la librería neuralnet
library(neuralnet)

# Los datos que se utilizan para crear la red neuronal no pueden contener
# valores cualitativos, por lo que los que se tienen hasta ahora no son válidos
# al contener la etiqueta de las clases de cada muestra. Se leen las clases y
# se mantienen con valores numéricos
train.y.nn = read.table(paste(folder, "train/y_train.txt", sep="/"), sep = "")
test.y.nn = read.table(paste(folder, "test/y_test.txt", sep="/"), sep = "")

# Se crea un dataframe para hacerlo compatible con la red neuronal
nn.train <- data.frame(x=train.x, y=train.y.nn)
nn.test <- data.frame(x=test.x, y=test.y.nn)

# La fórmula que se envía como parámetro a neuralnet tampoco permite que se
# escriba "y ~ .", por lo que se va a crear una fórmula que contenga todos los
# atributos "manualmente".
# Al crear la nueva estructura, ahora y es V1.
f <- as.formula(paste("V1 ~",
                      paste(colnames(nn.train)[!colnames(nn.train) %in% "V1"],
                            collapse = " + ")))

# Se deben crear estructuras con 1, 2 y 3 capas en un rango de 0 a 50. Se elige
# 50 al hacer varias comprobaciones y obtener el mejor resultado entre todas
# ellas.
# A la función neuralnet se le debe enviar la fórmula creada, los datos con los
# que entrenar la red, el número de capas de unidades ocultas con el número de
# unidades por capa. Para enviar esto se hace uso del parámetro hidden, que
# se compone de un vector con tantos elementos como número de capas de unidades
# ocultas se quiera.
# threshold es el porcentaje de error con el que se debe parar la comprobación
# en caso de no mejorarlo. Se establece 1%.
# stepmax es el máximo número de pasos que puede utilizar la red neuronal para
# entrenar
har.nn <- neuralnet(f, data=nn.train, hidden=c(50, 50, 50),
                    threshold=0.01, stepmax=1e6)

# La función compute es un método de clasificar objetos creados por neuralnet.
# Calcula la salida de todas las neuronas calculadas anteriormente, recibiendo
# todas las muestras que se quieren predecir
har.nn.comp.train <- compute(har.nn, train.x)
har.nn.comp.test <- compute(har.nn, test.x)

# La función compute devuelve un valor real pero para comparar se necesita
```

```r
# un valor natural, por lo que se redondea a su valor más cercano
har.nn.comp.train.round = round(har.nn.comp.train$net.result)
har.nn.comp.test.round = round(har.nn.comp.test$net.result)

# Se calcula la media de error de ambos conjuntos
har.nn.mean.train = mean(har.nn.comp.train.round != train.y.nn)
har.nn.mean.test = mean(har.nn.comp.test.round != test.y.nn)

print(paste("Etrain calculado con Redes Neuronales: ",
            round(har.nn.mean.train*100, digits = perDec), "%"))
```

```
## [1] "Etrain calculado con Redes Neuronales:  0.0408 %"
```

```r
print(paste("Etest calculado con Redes Neuronales: ",
            round(har.nn.mean.test*100, digits = perDec), "%"))
```

```
## [1] "Etest calculado con Redes Neuronales:  7.1259 %"
```

Referencias:

- https://www.r-bloggers.com/fitting-a-neural-network-in-r-neuralnet-package/
- https://stackoverflow.com/questions/17457028/working-with-neuralnet-in-r-for-the-first-time-get-requires-numeric-comp
- https://stackoverflow.com/questions/19360835/neuralnet-overcoming-the-non-convergence-of-algorithm
- https://www.quora.com/What-does-the-threshold-value-in-the-neuralnet-function-represent

7. Selección y ajuste modelo final.

Se muestra una tabla con los resultados obtenidos utilizando los modelos anteriores:

```r
c.train = c("RL"=har.lm.mean.train,
            "RL Ridge"=har.glm.ridge.mean.train,
            "RL Lasso"=har.glm.lasso.mean.train,
            "SVM"=har.svm.mean.train,
            "Boosting"=har.boosting.mean.train,
            "Random Forest"=har.forest.mean.train,
            "Red Neuronal"=har.nn.mean.train)

c.test = c("RL"=har.lm.mean.test,
           "RL Ridge"=har.glm.ridge.mean.test,
           "RL Lasso"=har.glm.lasso.mean.test,
           "SVM"=har.svm.mean.test,
           "Boosting"=har.boosting.mean.test,
           "Random Forest"=har.forest.mean.test,
           "Red Neuronal"=har.nn.mean.test)

general.train = data.frame("eTrain"=c.train, "eTest"=c.test)

general.train
```

```
##                      eTrain         eTest
## RL            0.0000000000000 0.05564981337
## RL Ridge      0.0228509249184 0.05768578215
## RL Lasso      0.0068008705114 0.05055989141
## SVM           0.0000000000000 0.10485239226
## Boosting      0.0089771490751 0.08652867323
```

```
## Random Forest 0.0000000000000 0.07125890736
## Red Neuronal  0.0004080522307 0.07125890736
```

Se puede determinar que el modelo que mejor ajusta la muestra es la *Regresión Logística con regularización Lasso.*

8. Estimacion del error E out del modelo lo más ajustada posible.

El menor error de test obtenido de todos los modelos ha sido *5.506%* con *Regresión Logística con regularización Lasso.*

9. Discutir y justificar la calidad del modelo encontrado y las razones por las que considera que dicho modelo es un buen ajuste que representa adecuadamente los datos muestrales.

El modelo que mejor ajusta las muestras es la Regresión Logística con regularización Lasso. Junto a este modelo, los mejores son Regresión Lineal y Regresión Logística con regularización Ridge. Que estos modelos lineales sean mejores que modelos más avanzados no lineales significa que la muestra realmente se puede separar linealmente (teniendo en cuenta que es multiclase y se realiza regresión múltiple). Tanto SVM, Random Forest y la Red Neuronal sobreajusta la muestra, ya que consigue predecir correctamente todas las muestras con el conjunto que ha entrenado el modelo, pero el error obtenido con el conjunto de test se eleva un poco más. Posiblemente con SVM se podría haber conseguido un mejor ajuste si se hubiera podido utilizar un parámetro libre con mayor precisión.