Trabajo 3

Samuel Cardenete Rodríguez y Juan José Sierra González

11 de mayo de 2017

Introducción:

Para la realización de esta práctica obtendremos el ajuste de modelos lineales basados en dos problemas centrados en dos conjuntos de datos diferentes. En primer lugar trabajaremos con un problema de clasificación, basado en el conjunto de datos "South African Heart Disease", para el reconocimiento de enfermedades cardiovasculares en una población de Sudáfrica; y en segundo lugar con un problema de regresión, basado en el conjunto de datos "Los Angeles Ozone", para predecir los niveles de ozono en Los Angeles.

Comenzaremos primeramente abordando el problema de clasificación:

Clasificación: "South African Heart Disease"

En este caso nos encontramos frente a un problema de clasificación, tal y como hemos visto anteriormente. Se trata de un conjunto de datos que clasifica individuos de una población de Sudáfrica, indicando si padecen o no una enfermedad del corazón en función de los hábitos de vida (consumo de tabaco, obesidad, alcohol...). Como primer paso para abordar el problema, leeremos los datos y los dividiremos seleccionando nuestro conjunto de entrenamiento y de prueba.

Lectura de datos:

Procedemos a la lectura de datos tanto de la base de datos de clasificación 'South African Heart Disease', como para la de regresión 'Los Angeles Ozone'.

```
datos_sudafrica = read.csv("./datos/africa.data")
```

Si analizamos los datos obtenidos, podemos observar que existe un atributo, 'row.names', perteneciente a nuestro dataset de 'South African Heart Disease' que actúa como clave primaria, es decir, como identificador, así que dicho atributo nos es inútil para el aprendizaje de nuestro modelo lineal, y por tanto lo suprimimos:

```
datos_sudafrica = datos_sudafrica[,-which(names(datos_sudafrica) == "row.names")]
```

Si seguimos con en análisis de los atributos observando sus tipos, nos damos cuenta de que existe un atributo de tipo factor, es decir una variable cualitativa, 'famhist', que nos indica el historial familiar de enfermedades de corazón, clasificado cmo 'ausente' o 'presente':

```
class(datos_sudafrica$famhist)
```

```
## [1] "factor"
```

Por tanto procedemos a interpretarlo de forma numérica, de forma que sustituimos 'presente' por 1 y 'ausente' por 0:

```
#Cambiamos la columna 'famhist' que contiene caracteres por su equivalente en valores numéricos:
datos_sudafrica = data.frame(famhist = (ifelse(datos_sudafrica$famhist=="Absent",0,1)),datos_sudafrica[
```

Conjuntos de training y test usados

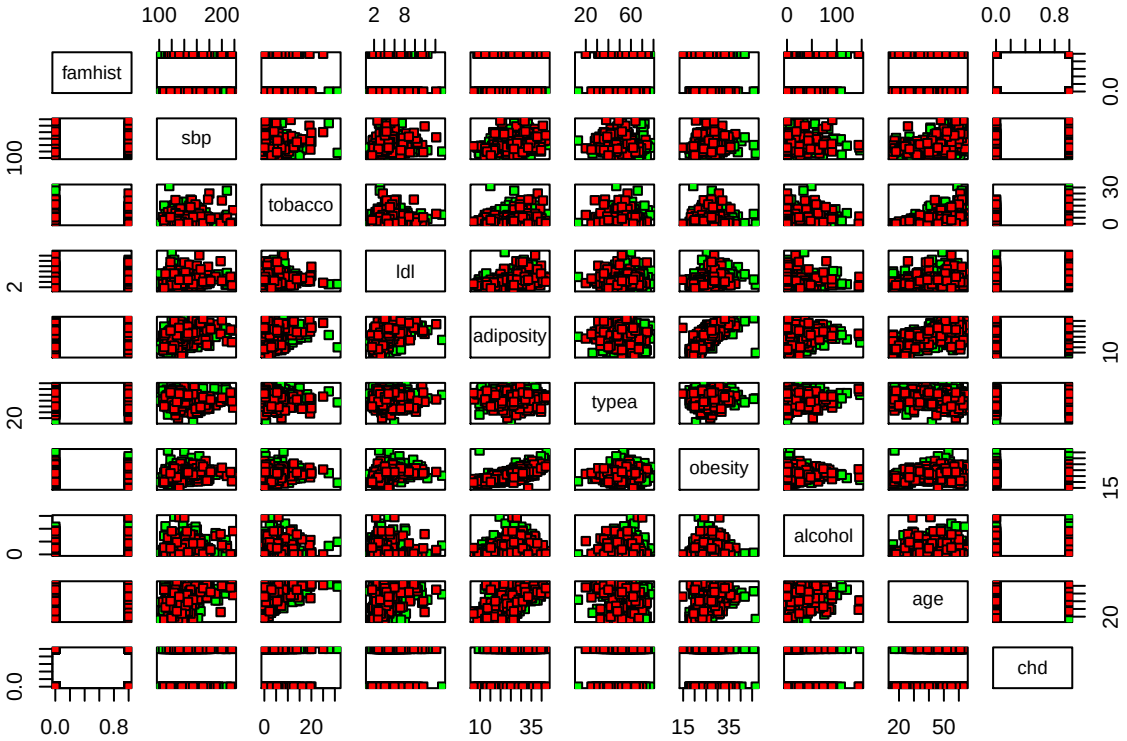
En primer lugar, realizaremos una división del conjunto de datos. Para el conjunto 'train' de entrenamiento emplearemos el 80% del conjunto total de los datos, de forma que el 20% restante será empleado para test. En caso de realizar validación más adelante explicaremos el cómo.
Procedemos al particionamiento de los datos, así como a su lectura:

Preprocesamiento de los datos

Procedemos ahora al preprocesado de los datos. Podemos emplear varias técnicas para la realización del preprocesamiento de datos. Pero antes de nada comencemos por realizar una vista preeliminar emplearemos el pairs, de tal forma que le indicaremos que compare cada atributo de los (incluido la clase) con todos los demás, de forma que obtengamos una matriz simétrica de gráficas, con los colores indicando la etiqueta según pueda padecer enfermedad (rojo) o no (verde):

```
#realizamos un attach de los datos para evitar repetir código:
color <- c(rep('green',sum(datos_sudafrica$chd ==0)),rep('red',sum(datos_sudafrica$chd==1)))


pairs(datos_sudafrica, bg = color, pch = 22)
```



Al representar la matriz de diagramas de dispersión de los datos podemos tener una vista preeliminar de la dispersión de los diferentes datos. Como podemos observar en la matriz de gráficas, la dispersión entre los datos es alta, de forma que apriori los únicos atributos que consideramos que se pueden estar relacionados son 'adiposity' con 'obesity'.

Veamos entonces si es podible realizar una reducción de la dimensionalidad de los datos mediante la aplicación de PCA (Computing the Principal Components) sobre el conjunto de datos para intentar comprobar si existen datos redundantes que puedan ser recombinados en nuevas características que sean resultado de combinaciones lineales de ellos. Prestaremos especial atención a los atributos obesidad y adiposidad.

PCA calcula la varianza de cada atributo respecto a los demás, de forma que aquellos atributos que posean menor varianza (cercana a cero) con respecto a los demás serán considerados como redundantes.

Además, para no arriesgar mucho (puesto que PCA va a ciegas) representaremos sólo aquellos componentes que expliquen hasta el 90% de la variabilidad de los datos (es necesario que los datos estén escalados y centrados para aplicar PCA):

```
sudafricaTrans = preProcess(datos_sudafrica, method = c("BoxCox", "center", "scale", "pca"),thresh = 0.9
summary(sudafricaTrans$rotation)
```

```
##       PC1                PC2                PC3          
##  Min.   :-0.47821   Min.   :-0.4437   Min.   :-0.29512  
##  1st Qu.:-0.36698   1st Qu.:-0.3718   1st Qu.:-0.23258  
##  Median :-0.30235   Median :-0.1270   Median : 0.02997  
##  Mean   :-0.28440   Mean   :-0.0794   Mean   : 0.06393  
##  3rd Qu.:-0.22574   3rd Qu.: 0.1600   3rd Qu.: 0.29213  
##  Max.   :-0.00052   Max.   : 0.4685   Max.   : 0.66755  
##       PC4                PC5                PC6          
##  Min.   :-0.25472   Min.   :-0.74845   Min.   :-0.26966  
##  1st Qu.:-0.18932   1st Qu.:-0.09262   1st Qu.:-0.20223  
##  Median : 0.03753   Median : 0.02747   Median :-0.07082  
##  Mean   : 0.08634   Mean   :-0.01124   Mean   : 0.01631  
##  3rd Qu.: 0.25177   3rd Qu.: 0.15449   3rd Qu.: 0.07155  
##  Max.   : 0.66522   Max.   : 0.44706   Max.   : 0.84138  
##       PC7                PC8          
##  Min.   :-0.629858   Min.   :-0.51793  
##  1st Qu.:-0.273606   1st Qu.:-0.22519  
##  Median : 0.167282   Median :-0.04904  
##  Mean   :-0.006012   Mean   :-0.02593  
##  3rd Qu.: 0.214367   3rd Qu.: 0.16619  
##  Max.   : 0.360403   Max.   : 0.63028
```

Como podemos observar en la tabla, se han reducido los atributos a 8 atributos (combinaciones lineales de los 10 anteriores). Pero si observamos las varianzas de cada atributo en la tabla respecto al resto de atributos, vemos que no existe ningun atributo cuyas varianzas sean cercanas todas a 0, por lo tanto, aún así para asegurarnos lo comprobamos mediante la siguiente función:

```
nearZeroVar(sudafricaTrans$rotation)
```

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

Como comprobamos con la función nearZero observamos que no existe ningun atributo cuyas varianzas respecto a las demás sean todas cercanas a cero, tal y como decíamos, esto queire decir que todos los atributos son considerados representativos, pues poseen dispersión. Por conclusión, no realizaremos ninguna reducción de atributos.

Para conlcuir el preprocesamiento de los datos, realizaremos los siguientes preprocesamientos de los datos:

- **Escalado**: Se trata de dividir cada uno de los atributos por su desviación típica.
- **Centrado**: Calculamos la media de cada atributo y se la restamos para cada valor.
- **Box-Cox**: Se trata de intentar reducir el sesgo de cada atributo, para intentar hacer este mas próximo a una distribución Gaussiana.

Para aplicar las transformaciones, emplearemos la función preprocess sobre nuestro conjunto Train, de forma que realizando un predict sobre el preprocesamiento obtenido, obtengamos el conjunto de datos train preprocesado:

```
sudafricaTrans = preProcess(sudafrica_train[,-ncol(sudafrica_train)], method = c("BoxCox", "center", "s
sudafrica_train[,-ncol(sudafrica_train)] =predict(sudafricaTrans,sudafrica_train[,-ncol(sudafrica_train
```

Estimación de parámetros

Antes de realizar un modelo, vemos cuales son las características más representativas (varianza mayor), de forma que no empecemos a realizar modelos a ciegas, sino fijándonos en la calidad de sus atributos.

Para ello emplearemos la función regsubsets. Esta función realiza una búsqueda exhaustiva (empleando Branch&Bound) de los mejores conjuntos de atributos en nuestro conjunto de datos (en nuestro caso train) para predecir en una regresión lineal:

```
regsub_sudafrica =regsubsets(datos_sudafrica[,-ncol(datos_sudafrica)], datos_sudafrica[,ncol(datos_suda

summary(regsub_sudafrica)
```

```
## Subset selection object
## 9 Variables  (and intercept)
##           Forced in Forced out
## famhist       FALSE      FALSE
## sbp           FALSE      FALSE
## tobacco       FALSE      FALSE
## ldl           FALSE      FALSE
## adiposity     FALSE      FALSE
## typea         FALSE      FALSE
## obesity       FALSE      FALSE
## alcohol       FALSE      FALSE
## age           FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          famhist sbp tobacco ldl adiposity typea obesity alcohol age
## 1  ( 1 ) " "     " " " "     " " " "       " "   " "     " "     "*"
## 2  ( 1 ) "*"     " " " "     " " " "       " "   " "     " "     "*"
## 3  ( 1 ) "*"     " " "*"     " " " "       " "   " "     " "     "*"
## 4  ( 1 ) "*"     " " "*"     "*" " "       " "   " "     " "     "*"
## 5  ( 1 ) "*"     " " "*"     "*" " "       "*"   " "     " "     "*"
## 6  ( 1 ) "*"     " " "*"     "*" " "       "*"   "*"     " "     "*"
## 7  ( 1 ) "*"     "*" "*"     "*" " "       "*"   "*"     " "     "*"
## 8  ( 1 ) "*"     "*" "*"     "*" "*"       "*"   "*"     " "     "*"
```

Como podemos observar, los mejores atributos son 'age' y 'famhist', seguidos por 'tobbaco' y 'ldl'.

Ahora que sabemos cuáles son las características más recomendables para realizar modelos, vamos a construir una serie de ellos con algunas de estas características y validaremos con el conjunto de test para comprobar los errores que reflejan.

Definición de modelos

```
#############################################################
# ESTO ES UNA CHAPUZA Y NO SABEMOS SI HABRÁ QUE HACERLO ASÍ Y/O AQUÍ
#############################################################

sudafrica_test[,-ncol(sudafrica_test)] =predict(sudafricaTrans,sudafrica_test[,-ncol(sudafrica_test)])
```

Para empezar calculamos un modelo lineal de forma que predecimos chd (etiquetas) a partir del atributo más representativo, en nuestro caso como hemos comprobado 'age'.

Una vez calculado el modelo, empleamos la función predict para obtener la probabilidad de cada etiqueta. Como en nuestro caso.

Definimos una función para el cálculo del error, etiquetas mal clasificadas entre etiquetas totales:

```
calculoErrorMatrizConfusion  = function (modelo, test, etiquetas){

  prob_test = predict(modelo, test[,-which(names(test) == etiquetas)], type="response")

  pred_test = rep(0, length(prob_test))
   # predicciones por defecto 0
  pred_test[prob_test >=0.5] = 1
   # >= 0.5 clase 1
  matriz_conf = table(pred_test, test[,which(names(test) == etiquetas)])
  print(matriz_conf)

  eout = mean(pred_test != test[,which(names(test) == etiquetas)])
}
```

```
m1_sudafrica = lm(chd ~ age, data=sudafrica_train)
eout_m1sud = calculoErrorMatrizConfusion(m1_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 46 21
##         1 11 14
```

```
eout_m1sud
```

```
## [1] 0.3478261
```

Obtenemos un error de 0.35, para nada aceptable, por tanto busquemos un modelo diferente empleando otra caraterística, la siguiente más representativa para el cálculo del modelo que en nuestro caso es famhist:

```
sudafrica_frame_1 = data.frame(sudafrica_train[,which(names(sudafrica_train) == "chd" | names(sudafrica

m1_sudafrica = lm(chd ~ . , data=sudafrica_frame_1)


eout_m1sud = calculoErrorMatrizConfusion(m1_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 48 20
##         1  9 15
```

```
eout_m1sud
```

```
## [1] 0.3152174
```

```
sudafrica_frame_2 = data.frame(sudafrica_train[,which(names(sudafrica_train) == "chd" | names(sudafrica

m2_sudafrica = lm(chd ~ . , data=sudafrica_frame_2)

eout_m2sud = calculoErrorMatrizConfusion(m2_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
```

```
##         0 50 19
##         1  7 16
```

```
eout_m2sud
```

```
## [1] 0.2826087
```

```
sudafrica_frame_3 = data.frame(sudafrica_train[,which(names(sudafrica_train) == "chd" | names(sudafrica

m3_sudafrica = lm(chd ~ . , data=sudafrica_frame_3)
eout_m3sud = calculoErrorMatrizConfusion(m3_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 51 17
##         1  6 18
```

```
eout_m3sud
```

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

```
sudafrica_frame_4 = data.frame(sudafrica_train[,which(names(sudafrica_train) == "chd" | names(sudafrica

m4_sudafrica = lm(chd ~ . , data=sudafrica_frame_4)
eout_m4sud = calculoErrorMatrizConfusion(m4_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 50 19
##         1  7 16
```

```
eout_m4sud
```

```
## [1] 0.2826087
```

Como podemos comprobar en los modelos lineales analizados y los errores correspondientes obtenidos, nos encontramos frente a una base de datos no lineal. Para mejorar el error hemos realizado diferentes transformaciones lineales sobre los atributos seleccionados como más representativos.

```
sudafrica_frame_5 = data.frame(sudafrica_train[,which(names(sudafrica_train) == "chd" | names(sudafrica

m5_sudafrica = lm(chd ~ I(age^2) + famhist + tobacco + ldl + typea, data=sudafrica_frame_5)
eout_m5sud = calculoErrorMatrizConfusion(m5_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 53 25
##         1  4 10
```

```
eout_m5sud
```

```
## [1] 0.3152174
```

Tras realizar transformaciones logarítmicas y cuadráticas sobre los datos, experimentalmente hemos reducido el error fuera de la muestra a un 0.23 empleando una transformación cuadrática sobre el atributo 'age' (el que mejor representa la muestra como hemos visto en regSubsets).

Regularización

Procedamos ahora a realiar una regularización empleando Weight-decay mediante la función glmnet. Dicha función recibe los siguientes hiperparámetros:

Alpha = *Para aplicar el weight-decay utilizaremos dicho argumento con valor 0.* **lambda** = Parámetro de regularización. (Multiplica la matriz de identidad)

Hace falta tener en cuenta antes la correcta elección del lambda, de forma que escojamos el mejor lambda. En lugar de escoger un lambda de forma arbitraria, sería mejor emplear validación cruzada

```
etiquetas = sudafrica_train[,which(names(sudafrica_train) == "chd")]
tr = sudafrica_train[,-which(names(sudafrica_train) == "chd")]
tr = as.matrix(tr)
crossvalidation =cv.glmnet(tr,etiquetas,alpha=0)
print(crossvalidation$lambda.min)
```

```
## [1] 0.07095296
```

Una vez obtenida la lambda que proporciona un menor E_{out}, procedemos a generar un modelo de regularización, en primer lugar empleando el valor de lambda generado por validación cruzada, y en segundo lugar empleando un lambda igual a cero, de forma que no apliquemos regularización.

```
modelo_reg = glmnet(tr,etiquetas,alpha=0,lambda=crossvalidation$lambda.min)
print(modelo_reg)
```

```
##
## Call:  glmnet(x = tr, y = etiquetas, alpha = 0, lambda = crossvalidation$lambda.min)
##
##      Df   %Dev  Lambda
## [1,]  9 0.2496 0.07095
```

Como podemos ver, aplicando regularización con el lambda obtenido tras la validación cruzada obtenemos una varianza del 0.23. Probemos ahora no aplizando regularización, empleando un hiperparámetro lambda de 0 y comprobemos su varianza:

```
modelo_reg = glmnet(tr,etiquetas,alpha=0,lambda=0)
print(modelo_reg)
```

```
##
## Call:  glmnet(x = tr, y = etiquetas, alpha = 0, lambda = 0)
##
##      Df   %Dev Lambda
## [1,]  9 0.2534      0
```

Como podemos comprobar, las desviaciones obtenidas empleando o no regularización mediante weight-decay son similares, por tanto, comcluimos en que emplear regularización no merece la pena.