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 la base de datos de clasificación 'South African Heart Disease'.

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

```
##   row.names sbp tobacco  ldl adiposity famhist typea obesity alcohol age
## 1         1 160   12.00 5.73     23.11 Present    49   25.30   97.20  52
## 2         2 144    0.01 4.41     28.61  Absent    55   28.87    2.06  63
## 3         3 118    0.08 3.48     32.28 Present    52   29.14    3.81  46
## 4         4 170    7.50 6.41     38.03 Present    51   31.99   24.26  58
## 5         5 134   13.60 3.50     27.78 Present    60   25.99   57.34  49
## 6         6 132    6.20 6.47     36.21 Present    62   30.77   14.14  45
##   chd
## 1   1
## 2   1
## 3   0
## 4   1
## 5   1
## 6   0
```

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

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

Si seguimos con el análisis de los atributos observando sus tipos, nos damos cuenta de que existe un atributo de tipo factor, es decir, se trata de una variable cualitativa. Este atributo es 'famhist', que nos indica el historial familiar de enfermedades de corazón, clasificado como 'ausente' si no se ha producido ningún caso en la familia, o 'presente' si es al contrario:

```
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

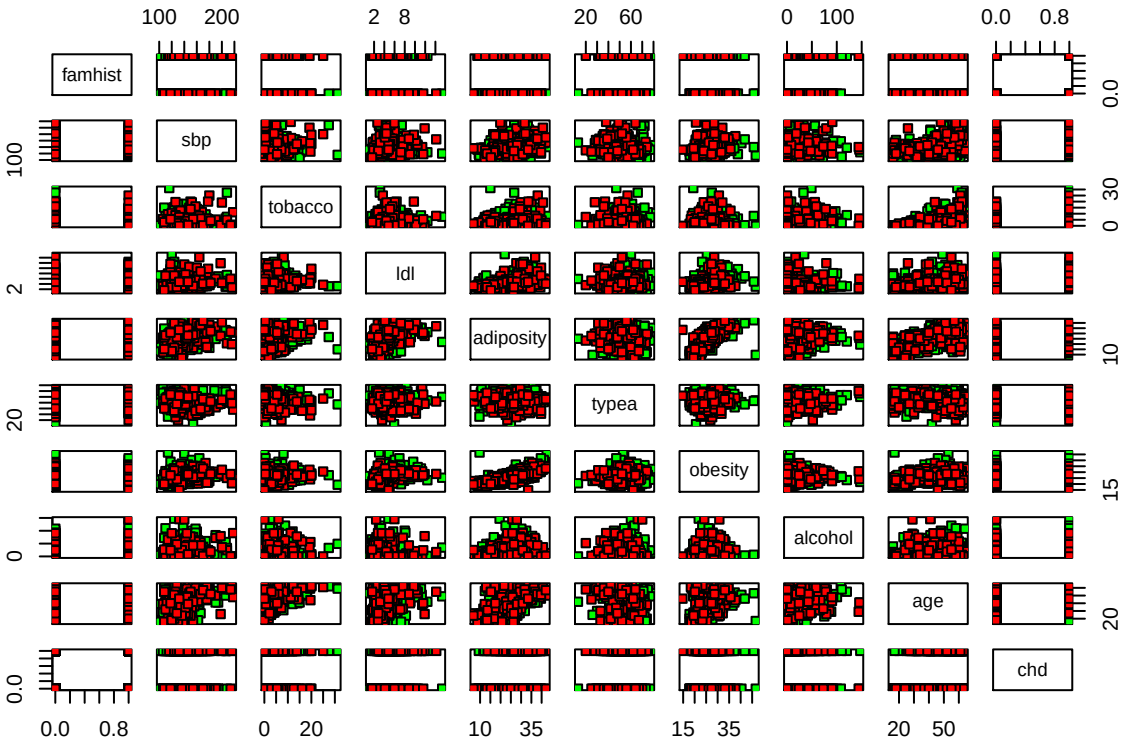
A continuación, realizaremos el particionamiento del conjunto de datos. Para el conjunto 'train' de entrenamiento emplearemos el 80% del conjunto total de los datos, de forma que el 20% restante será empleado para test. Hemos decidido utilizar este porcentaje dado que se nos ha explicado que un buen reparto entre train y test oscila entre dos terceras partes para train, o bien hasta un 80%, utilizando el resto para test. Procedemos al particionamiento de los datos, así como a su almacenamiento en respectivas variables:

Preprocesamiento de los datos

Antes de entrar en el preprocesamiento, consideramos interesante realizar una vista preliminar de las gráficas de cada atributo de la base de datos siendo reflejado con todos los demás. Así obtenemos una matriz simétrica de gráficas, con los colores indicando la etiqueta según padezca enfermedad (rojo) o no (verde):

```
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 podemos hacernos una primera idea de la dispersión de los datos. Como podemos observar esta dispersión es alta, y a priori los únicos atributos que consideramos que pueden estar sustancialmente relacionados entre sí son 'adiposity' y 'obesity'.

Como estamos tratando un problema de clasificación binaria, generar las gráficas del atributo clase no nos da ninguna ventaja, pero lo hemos dejado en la matriz final para que se pueda apreciar que no aporta ninguna información. Esto es debido a que para todos los atributos, existen casos de ambas clases. De forma contraria,

la clase se podría llegar a definir en función de un único parámetro, algo prácticamente inimaginable en un problema real.

Veamos entonces si es posible 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 atributos redundantes. Estos atributos redundantes podrían ser recombinados en nuevas características producto de combinaciones lineales de ellos. Prestaremos especial atención a los atributos 'obesity' y 'adiposity' por el motivo indicado anteriormente.

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 trabaja "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. Aún así, para asegurarnos lo comprobamos mediante la siguiente función:

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

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

Como comprobamos con la función nearZeroVar no existe ningun atributo cuyas varianzas respecto a las demás sean todas cercanas a 0, y por tanto todos los atributos son considerados representativos, pues poseen dispersión. En conclusión, no realizaremos ninguna reducción de atributos.

Para conlcuir el preprocesamiento de los datos, realizaremos las siguientes operaciones sobre ellos:

- **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 objeto transformación obtenido, obtengamos el conjunto de datos train preprocesado. A continuación, realizaremos las mismas transformaciones sobre el conjunto de test, utilizando el mismo objeto transformación:

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

Estimación de parámetros

Antes de realizar un modelo, veamos cuáles son las características más representativas (las que ofrecen varianza mayor con respecto al resto de datos), 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 las mejores agrupaciones de atributos en nuestro conjunto de entrenamiento para predecir en una regresión lineal:

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

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 ) "*"     "*" "*"     "*" "*"       "*"   "*"     " "     "*"
```

En la gráfica aportada por la función, interpretamos como atributos más representativos aquellos cuyas columnas tienen más estrellas. Nuestro criterio a seguir a la hora de generar modelos lineales será, por tanto, tratar de predecir utilizando agrupaciones de atributos considerados representativos, comenzando por el

mejor y de forma descendente. Para mostrar un ejemplo que explique la gráfica, los mejores atributos serían 'age' y 'famhist', y los peores 'adiposity' y 'alcohol'.

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

Para empezar, calculamos un sencillo modelo lineal con el que intentamos predecir 'chd' (nuestras etiquetas) a partir del atributo más representativo, en nuestro caso y como hemos comprobado, 'age'.

```
m_muestra_sudafrica = lm(chd ~ age, data=sudafrica_train)
```

Para el cálculo del error, y a fin de intentar aportar un punto de generalización al problema a abordar, definimos una función que calcula el error (etiquetas mal clasificadas en función de etiquetas totales) y muestra la matriz de confusión en el conjunto de test a partir de un modelo:

```
calculoErrorMatrizConfusion  = function (modelo, test, etiquetas){
  # Una vez calculado el modelo, empleamos la función predict
  # para obtener la probabilidad de cada etiqueta.
  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)

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

Utilizamos nuestra función para calcular el error del modelo de muestra generado anteriormente:

```
etest_mmuestrasud = calculoErrorMatrizConfusion(m_muestra_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 49 21
##         1 10 12
```

```
etest_mmuestrasud
```

```
## [1] 0.3369565
```

Obtenemos un error de 0.337, pero se trata de un modelo excesivamente simple como para reflejar buenos resultados en un problema real. Por tanto busquemos un modelo diferente empleando otra carasterística, la siguiente más representativa en el conjunto de datos, que en nuestro caso es 'famhist':

```
m1_sudafrica = lm(chd ~ age + famhist, data=sudafrica_train)

etest_m1sud = calculoErrorMatrizConfusion(m1_sudafrica, sudafrica_test, "chd")
```

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

```
##         0 47 18
##         1 12 15
```

```
etest_m1sud
```

```
## [1] 0.326087
```

Utilizando dos características el error en el conjunto de test desciende hasta un 0.326. Seguimos probando nuevos modelos, así que añadimos la siguiente característica recomendada por regsubsets, 'tobacco':

```
m2_sudafrica = lm(chd ~ age + famhist + tobacco, data=sudafrica_train)

etest_m2sud = calculoErrorMatrizConfusion(m2_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 47 15
##         1 12 18
```

```
etest_m2sud
```

```
## [1] 0.2934783
```

Un error reflejado de 0.293 ya se acerca más a lo que buscamos, poco a poco vamos avanzando hacia un error menor en el conjunto de validación. Como aún nos queda una buena cantidad de características por probar, añadimos una más al siguiente modelo, 'ldl':

```
m3_sudafrica = lm(chd ~ age + famhist + tobacco + ldl, data=sudafrica_train)

etest_m3sud = calculoErrorMatrizConfusion(m3_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 50 15
##         1  9 18
```

```
etest_m3sud
```

```
## [1] 0.2608696
```

En esta ocasión tenemos un error de 0.261, que mejora sustancialmente al que teníamos antes. Probemos con la siguiente característica, 'typea':

```
m4_sudafrica = lm(chd ~ age + famhist + tobacco + ldl + typea, data=sudafrica_train)

etest_m4sud = calculoErrorMatrizConfusion(m4_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 51 15
##         1  8 18
```

```
etest_m4sud
```

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

El error se reduce a 0.25, lo podemos empezar a considerar un error aceptable. No obstante, sigamos probando a añadir la siguiente característica según regsubsets, 'obesity':

```
m5_sudafrica = lm(chd ~ age + famhist + tobacco + ldl + typea + obesity, data=sudafrica_train)

etest_m5sud = calculoErrorMatrizConfusion(m5_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 50 17
##         1  9 16
```

```
etest_m5sud
```

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

Aquí encontramos el primer bache, y es que aumentando el número de características empeoramos el error en el test. Seguramente sea debido a sobreajuste, pero para asegurarnos vamos a sustituir 'obesity' por el siguiente atributo más válido según regsubsets, 'sbp':

```
m6_sudafrica = lm(chd ~ age + famhist + tobacco + ldl + typea + sbp, data=sudafrica_train)

etest_m6sud = calculoErrorMatrizConfusion(m6_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 49 15
##         1 10 18
```

```
etest_m6sud
```

```
## [1] 0.2717391
```

¿Transformaciones lineales o NO lineales? ##
##
##
Efectivamente, seguimos encontrando errores peores, por lo que abandonamos esta línea de exploración al estar enfrentándonos a una base de datos no lineal. Para mejorar el error hemos realizado diferentes transformaciones lineales sobre los atributos seleccionados como más representativos.

Volvemos al modelo en el que menor error obtuvimos, el formado por los 5 mejores atributos según regsubsets, y probamos a utilizar una variable cuadrática, en este caso elevar al cuadrado la característica 'age':

```
m7_sudafrica = lm(chd ~ I(age^2) + famhist + tobacco + ldl + typea, data=sudafrica_train)

eout_m7sud = calculoErrorMatrizConfusion(m7_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 53 16
##         1  6 17
```

```
eout_m7sud
```

```
## [1] 0.2391304
```

En este modelo hemos conseguido otra mejora, esta vez el error desciende hasta 0.239. Seguiremos intentando encontrar mejores errores realizando nuevas transformaciones, ahora con una nueva característica.

```
m8_sudafrica = lm(chd ~ I(age^3) + famhist + tobacco + ldl + typea, data=sudafrica_train)

eout_m8sud = calculoErrorMatrizConfusion(m8_sudafrica, sudafrica_test, "chd")
```

```
##
## pred_test  0  1
##         0 52 16
##         1  7 17
```

```
eout_m8sud
```

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

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] 0.07023182
```

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 0.2365 0.07023
```

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 0.2405      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.

Regresión: “LA Ozone”

En este caso, el problema de regresión a abortar se basa en los registros de concentración de ozono en la atmósfera de Los Angeles.

Apartir de dichos datos, buscamos predecir These data record the level of atmospheric ozone concentration from eight daily meteorological measurements made in the Los Angeles basin in 1976.