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'.

```
set.seed(5)
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:

```
set.seed(5)
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:

```r
set.seed(5)
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:

```r
set.seed(5)
#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:

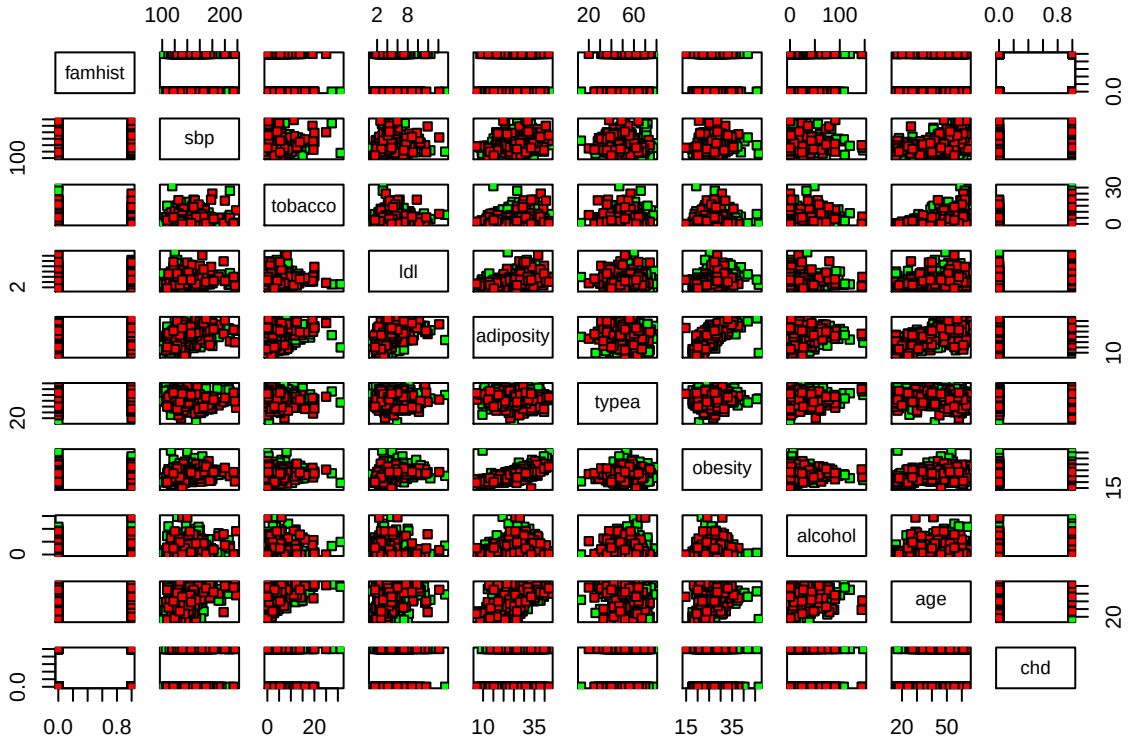
```r
set.seed(5)
#Si queremos obtener un conjunto de indices train para luego ejecutar un modelo lineal sobre el train:
train = sample (nrow(datos_sudafrica), round(nrow(datos_sudafrica)*0.8))
#definimos ambos conjuntos en dos data.frame diferentes:
sudafrica_train = datos_sudafrica[train,]
sudafrica_test = datos_sudafrica[-train,]
```

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):

```r
set.seed(5)
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):

```
set.seed(5)
sudafricaTrans = preProcess(datos_sudafrica, method = c("BoxCox", "center", "scale", "pca"),thresh = 0.9
sudafricaTrans$rotation
```

```
##                    PC1         PC2         PC3          PC4         PC5
## famhist   -0.2050139927 -0.30494678  0.34195449 -0.202594506 -0.74845338
## sbp       -0.3077083767  0.01703881 -0.28216981  0.118191931 -0.07472308
## tobacco   -0.2969827181 -0.40226677 -0.24989810 -0.009760375  0.44706077
## ldl       -0.3439441006  0.20764197  0.29544043 -0.149477339  0.11894894
```

```
## adiposity -0.4782065115  0.31182338 -0.02890177  0.084828439 -0.03212513
## typea     -0.0005199528 -0.14430241  0.66755256  0.525432878  0.30598887
## obesity   -0.3746620497  0.46852394  0.08884917  0.296289528 -0.09859014
## alcohol   -0.1099619188 -0.39412505 -0.29511662  0.665220259 -0.28388984
## age       -0.4390183606 -0.10972065 -0.18062508 -0.209959814  0.08705564
## chd       -0.2879347228 -0.44366490  0.28220217 -0.254721846  0.16633840
##                    PC6        PC7         PC8
## famhist   -0.02281391  0.2775154 -0.23876306
## sbp        0.84137981 -0.1466553 -0.18449046
## tobacco   -0.25761353  0.3604026 -0.35002407
## ldl       -0.21729271 -0.6298578 -0.51792653
## adiposity -0.11882828  0.1420658  0.20609767
## typea      0.24095641  0.2188496 -0.12113037
## obesity   -0.15703717  0.1924981  0.24717235
## alcohol   -0.26966337 -0.3599405  0.04646943
## age        0.04296918  0.2009201  0.02304498
## chd        0.08107516 -0.3159227  0.63028348
```

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:

```
set.seed(5)
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:

```
set.seed(5)
sudafricaTrans = preProcess(sudafrica_train[,-which(names(sudafrica_train) == "chd")], method = c("BoxC
sudafrica_train[,-which(names(sudafrica_train) == "chd")] =predict(sudafricaTrans,sudafrica_train[,-whi
sudafrica_test[,-which(names(sudafrica_test) == "chd")] =predict(sudafricaTrans,sudafrica_test[,-which(
```

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:

```
set.seed(5)
regsub_sudafrica =regsubsets(datos_sudafrica[,-which(names(datos_sudafrica) == "chd")], datos_sudafrica

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.

Regularización

Procedamos ahora a realizar un análisis para ver si es interesante aplicar 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 que mejores resultados nos pueda arrojar. En lugar de seleccionarlo de forma arbitraria, será mejor emplear validación cruzada:

```
set.seed(5)
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 obtenido el 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. El objetivo será comprobar si los parámetros obtenidos son significativamente diferentes como para que merezca la pena realizar regularización en nuestros modelos.

```
set.seed(5)
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
```

Aplicando regularización con el lambda obtenido tras la validación cruzada obtenemos una desviación del 0.237. Probemos ahora no aplizando regularización, empleando un hiperparámetro lambda de 0, y comprobemos su desviación:

```
set.seed(5)
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 (0.237 frente a 0.241), por tanto, comcluimos en que emplear regularización no merece la pena.

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'.

```
set.seed(5)
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:

```
set.seed(5)
calculoErrorMatrizConfusion  = function (modelo, test, etiquetas, imprimir_matriz=TRUE){
```

```
  # 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)])
  if (imprimir_matriz)
    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:

```
set.seed(5)
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':

```
set.seed(5)
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':

```
set.seed(5)
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':

```
set.seed(5)
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':

```
set.seed(5)
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':

```
set.seed(5)
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':

```
set.seed(5)
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
```

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 no 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':

```
set.seed(5)
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 una cúbica.

```
set.seed(5)
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
```

El error no es malo pero sigue siendo un peor modelo que el mejor que hemos generado ahora mismo. Hemos comprobado también que es contraproducente realizar transformaciones con logaritmos y raíces debido a la cantidad de datos negativos que tiene la base de datos una vez normalizada.

En resumen, tras realizar distintas combinaciones de atributos y tratar de predecir con ellos, utilizando alguna transformación no lineal, experimentalmente hemos reducido el error fuera de la muestra a un 0.239.

Este error se ha obtenido con un modelo con transformación cuadrática sobre el atributo 'age' (el que mejor representa la muestra como hemos visto en regSubsets) y utilizando los 4 siguientes mejores atributos.

Estimacion del error E_{out}

Una vez que hemos hecho una selección de los modelos y hemos dado con uno modelo lineal que proporciona un valor del error de 0.239, vamos a proceder a realizar una estimación del E_{out} de forma que obtengamos un valor más representativo del error.

Para ello vamos a repetir la generación de particiones, tanto train como test, con la misma proporción realizada anteriormente, empleando también un 0.8 para train y un 0.2 para test. Repetiremos el proceso 100 veces y obtendremos la media de los E_{out}.

Definamos una función que realice tanto la generación y el particionado, así como las transformaciones sobre los conjuntos de datos y que devuelva el error generado por el mejor modelo obtenido (m7):

```
set.seed(5)
generarErrorParticionSudafrica = function(datos){
  #Si queremos obtener un conjunto de indices train para luego ejecutar un modelo lineal sobre el train
  el_train = sample (nrow(datos), round(nrow(datos)*0.7))
  #definimos ambos conjuntos en dos data.frame diferentes:
  sudafrica_train = datos[el_train,]
  sudafrica_test = datos[-el_train,]

  #TRANSFORMACIONES:
  sudafricaTrans = preProcess(sudafrica_train[,-which(names(datos_sudafrica) == "chd")], method = c("Bo
sudafrica_train[,-which(names(sudafrica_train) == "chd")] =predict(sudafricaTrans,sudafrica_train[,-whi
sudafrica_test[,-which(names(sudafrica_test) == "chd")] =predict(sudafricaTrans,sudafrica_test[,-which(

#EVALUACION DEL MODELO
m7_sudafrica = lm(chd ~ I(age^2) + famhist + tobacco + ldl + typea, data=sudafrica_train)

etest_m5sudafrica = calculoErrorMatrizConfusion(m7_sudafrica, sudafrica_test, "chd", FALSE)
etest_m5sudafrica
}
```

Una vez que hemos definido la funcion, realizamos una evaluación del error 1000 veces, de forma que cálculemos como resultado final la media de los errores generados para cada una de las 1000 particiones:

```
set.seed(5)
mean(replicate(100, generarErrorParticionSudafrica(datos_sudafrica)))
```

```
## [1] 0.2846043
```

Como podemos ver, obtenemos un E_{out} medio del 0.08869314, bástante bueno. Este error representa mejor el error real obtenido por nuestro modelo de regresión lineal.

Conclusión y modelo final seleccionado

Tras los diferentes modelos generados, habiendo utilizado distintas combinaciones de atributos, tratando de optimizar con transformaciones no lineales, y habiendo decidido no utilizar regularización, hemos conlcuido que el mejor modelo que hemos podido encontrar ha sido aquel que utiliza la agrupación de los 5 mejores atributos según regsubsets, y con una transformación cuadrática del atributo 'age'.

Con este modelo hemos reducido la tasa de error hasta un 23.9%. Si consideramos la distribución de los datos que observamos en la matriz de gráficas al comienzo del problema, y según hemos comprobado de

forma experimental, la base de datos no es lineal, y con un modelo lineal no podremos ajustar de mucha mejor forma los datos. A pesar de ello, decidimos empezar por ahí para encontrar qué características son más representativas a la hora de reducir el error. Hemos construido diferentes modelos combinando los atributos que mayor varianza presentan (más representativos) y hemos ido reduciendo el error hasta que se ha empezado a producir sobreajuste. A partir de aquí, añadir más atributos sólo nos continúa ajustando de más la función, provocando que se produzcan muchas curvas en la función que dificulten generalizar los datos del conjunto de test.

Llegados a este punto, como no nos conviene añadir más atributos y puesto que nos encontramos frente a un modelo no lineal, consideramos realizar algunas transformaciones no lineales sobre los atributos que ya tenemos, y encontramos el mejor error realizando una transformación cuadrática sobre el atributo 'age'. Optamos por esto dado que contamos con unos datos muy dispersos y difícilmente separables con una función lineal. Probando con nuevas transformaciones no logramos mejor resultado así que nos quedamos con este modelo. Y por último, dado que comprobando los parámetros derivados de la regularización hemos visto que no varían de forma significativa, hemos decidido no realizarla.

Como conclusión final, un modelo que utilice suficiente número de atributos para predecir (sin llegar al límite de sobreajuste) y que contenga algún tipo de transformación no lineal es comprensible que refleje buenos resultados en un problema real con datos dispersos y con ruido. Dentro de los distintos modelos que cumplan esas características, de forma experimental hemos podido comparar varios y nos hemos quedado con el anteriormente mencionado.

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.

A partir de dichos datos, buscamos predecir la cota máxima por hora de la concentración de ozono de la ciudad de Upland, California.

Los atributos pertenecientes al conjunto de datos son los siguientes: * **vh**: Indica la presión atmosférica medida en la base aérea de los Angeles. * **wind**: Velocidad el viento en mph. * **humidity**: Representa la humedad en el aire. * **temp**: La temperatura que hay cuando se realiza la medición. * **ibh**: Altura apartir de la cuál la presión atmosférica cambia. * **dpg**: Nos indica el gradiante de la presión atmosférica actual. * **ibt**: Altura apartir de la cuál la temperatura atmosférica cambia. * **vis**: Indica el nivel de visibilidad en ese día. * **doy**: Nos indica el día en el que se realizó las mediciones anteriormente mencionadas. Básicamente, este campo actua como clave primaria para cada medición.

Procedamos a realizar la lectura de los datos obtenidos:

```
set.seed(5)
datos_ozone = read.csv("./datos/LAozone.data")
head(datos_ozone)
```

```
##   ozone   vh wind humidity temp  ibh dpg ibt vis doy
## 1     3 5710    4       28   40 2693 -25  87 250   3
## 2     5 5700    3       37   45  590 -24 128 100   4
## 3     5 5760    3       51   54 1450  25 139  60   5
## 4     6 5720    4       69   35 1568  15 121  60   6
## 5     4 5790    6       19   45 2631 -33 123 100   7
## 6     4 5790    3       25   55  554 -28 182 250   8
```

Si analizamos los datos obtenidos, podemos observar que existe un atributo, 'doy', que actúa como clave primaria, es decir, como identificador, indicándonos el dia en el que se realizó la medición así que dicho atributo nos es inútil para el aprendizaje de un modelo lineal, y por tanto lo suprimimos:

```r
set.seed(5)
datos_ozone = datos_ozone[,-which(names(datos_ozone) == "doy")]
```

Conjuntos de training y test usados

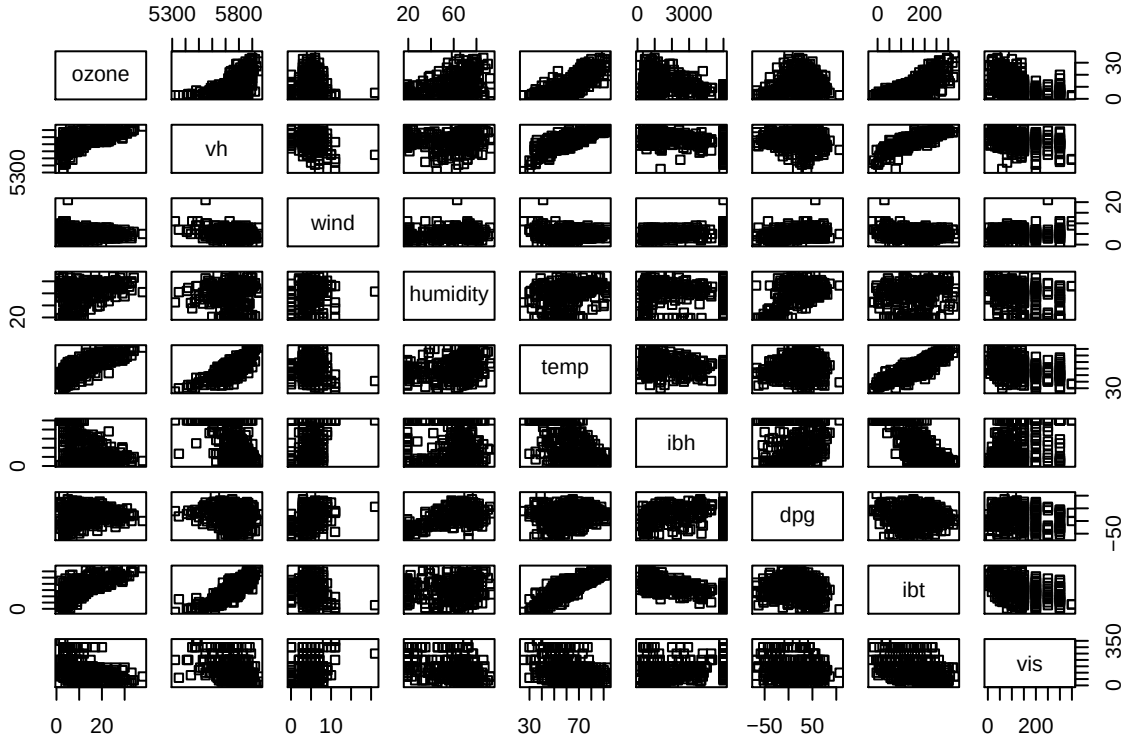
A continuación, realizaremos el particionamiento del conjunto de datos. Para el conjunto 'train' de entrenamiento emplearemos el 80% del conjunto total de los datos, de forma que el 30% restante será empleado para test. Hemos decidido utilizar este porcentaje dado que se nos ha explicado que un buen reparto entre train y test oscila entre dos terceras partes para train, o bien hasta un 70%, utilizando el resto para test. Procedemos al particionamiento de los datos, así como a su almacenamiento en respectivas variables:

```r
set.seed(5)
#Si queremos obtener un conjunto de indices train para luego ejecutar un modelo lineal sobre el train:
train = sample (nrow(datos_ozone), round(nrow(datos_ozone)*0.7))
#definimos ambos conjuntos en dos data.frame diferentes:
ozone_train = datos_ozone[train,]
ozone_test = datos_ozone[-train,]
```

Preprocesamiento de los datos

Antes de entrar en el preprocesamiento, consideramos interesante realizar una vista preliminar de las gráficas de dispersión para cada atributo de la base de datos siendo reflejado con todos los demás. Así obtenemos una matriz simétrica de gráficas.

```r
set.seed(5)
pairs(datos_ozone, pch = 22)
```



Al representar la matriz de gráficas podemos hacernos una primera idea de la dispersión de los datos. Como podemos observar esta dispersión es menor que en el ejercicio de clasificación, a priori los atributos que

consideramos que pueden estar sustancialmente relacionados entre sí son 'temp' y 'ibt', ''vh y 'temp', 'vh' y 'ibt'.

En este caso, estamos trabajando sobre un conjunto de datos que posee 8 atributos, por para realizar una reducción de datos debemos de estar seguros que se trata de un atributo redundante para el aprendizaje de nuestro modelo lineal.

Veamos entonces si es aconsejable 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 mencionados 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):

```
set.seed(5)
ozoneTrans = preProcess(datos_ozone, method = c("BoxCox", "center", "scale", "pca"),thresh = 0.9)
ozoneTrans$rotation
```

```
##                  PC1        PC2         PC3           PC4         PC5
## ozone      0.42272830  0.1235738 -0.09393314  0.0006545469  0.11795883
## vh         0.40352090 -0.2115870 -0.20026465 -0.3475784751 -0.23985957
## wind      -0.08066944  0.4450446 -0.62289360  0.4818318933 -0.40684509
## humidity   0.20818805  0.5317961  0.26900179 -0.0042913263  0.12458283
## temp       0.43864144  0.0480664 -0.26555914 -0.2953899630 -0.06798997
## ibh       -0.35184626  0.1638791 -0.07650560 -0.6567696290 -0.48242685
## dpg        0.05544026  0.6274628  0.05593271 -0.2798932110  0.32668701
## ibt        0.45634757 -0.1556804 -0.16580700  0.0735625937  0.02385079
## vis       -0.28476835 -0.1094347 -0.61944898 -0.2113176418  0.63471075
```

Como podemos observar en la tabla, se han reducido los atributos a 5 atributos (combinaciones lineales de los 8 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:

```
set.seed(5)
nearZeroVar(ozoneTrans$rotation)
```

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

Para aplicar las transformaciones, emplearemos la función preProcess sobre nuestro conjunto train, tal y como hemos hecho en el problema de clasificación, 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:

```
set.seed(5)
ozoneTrans = preProcess(ozone_train[,-which(names(datos_ozone) == "ozone")], method = c("BoxCox", "cent
ozone_train[,-which(names(ozone_train) == "ozone")] =predict(ozoneTrans,ozone_train[,-which(names(ozone
ozone_test[,-which(names(ozone_test) == "ozone")] =predict(ozoneTrans,ozone_test[,-which(names(ozone_te
```

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:

```
set.seed(5)
regsub_ozone =regsubsets(datos_ozone[,-which(names(datos_ozone) == "ozone")], datos_ozone[,which(names(

summary(regsub_ozone)
```

```
## Subset selection object
## 8 Variables  (and intercept)
##          Forced in Forced out
## vh           FALSE      FALSE
## wind         FALSE      FALSE
## humidity     FALSE      FALSE
## temp         FALSE      FALSE
## ibh          FALSE      FALSE
## dpg          FALSE      FALSE
## ibt          FALSE      FALSE
## vis          FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          vh  wind humidity temp ibh dpg ibt vis
## 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 'temp' y 'ibt', ''vh y 'temp', 'vh' y 'ibt'.

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.

Regularización

Procedamos ahora a realizar un análisis para ver si es interesante aplicar 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 que mejores resultados nos pueda arrojar. En lugar de seleccionarlo de forma arbitraria, será mejor emplear validación cruzada:

```
set.seed(5)
variable_respuesta = ozone_train[,which(names(ozone_train) == "ozone")]
tr = ozone_train[,-which(names(ozone_train) == "ozone")]
tr = as.matrix(tr)
crossvalidation =cv.glmnet(tr,variable_respuesta,alpha=0)
print(crossvalidation$lambda.min)
```

```
## [1] 0.6519704
```

Una vez obtenido el 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. El objetivo será comprobar si los parámetros obtenidos son significativamente diferentes como para que merezca la pena realizar regularización en nuestros modelos.

```
set.seed(5)
modelo_reg = glmnet(tr,variable_respuesta,alpha=0,lambda=crossvalidation$lambda.min)
print(modelo_reg)
```

```
##
## Call:  glmnet(x = tr, y = variable_respuesta, alpha = 0, lambda = crossvalidation$lambda.min)
##
##      Df   %Dev Lambda
## [1,]  8 0.6626  0.652
```

Aplicando regularización con el lambda obtenido tras la validación cruzada obtenemos una desviación del 0.6934. Probemos ahora no aplizando regularización, empleando un hiperparámetro lambda de 0, y comprobemos su desviación:

```
set.seed(5)
modelo_reg = glmnet(tr,variable_respuesta,alpha=0,lambda=0)
print(modelo_reg)
```

```
##
## Call:  glmnet(x = tr, y = variable_respuesta, alpha = 0, lambda = 0)
##
##      Df   %Dev Lambda
## [1,]  8 0.6659      0
```

Como podemos comprobar, las desviaciones obtenidas empleando o no regularización mediante weight-decay son similares (0.6934 frente a 0.6982), por tanto, comcluimos en que emplear regularización no merece la pena. ###Definición de modelos Para la realización de los modelos hemos pensado en emplear regresión logística mediante la función 'glm'; pero si empleamos regresión logística obtendremos un conjunto de probabilidades que predigan los valores. Para ello, como precondición sería necesario normalizar los valores de la variable respuesta de forma que se encontraran en el intervalo [0,1].

Entonces procederemos a generar modelos mediante regresión lineal, así obtendremos un valor dependiendo de la distancia de los atributos a la recta generada.

Procedamos comenzando por un modelo de un único atributo, temp. Antes definimos una función para el cálculo del error cuadrático:

Pero antes de todo, definamos una función para el cálculo del error de nuestros modelos de regresión lineal que generemos. Para ello realizamos un cálculo de la media de errores normalizados en un intervalo.

El cálculo se realiza entre el cociente de la resta de las variables respuesta menos las predichas, y la diferencia entre el valor máximo y mínimo de las variables respuesta:

```
set.seed(5)

calculoErrorMedioIntervalo  = function (modelo, test, variable_respuesta){
  prob_test = predict(modelo, test[,-which(names(test) == variable_respuesta)])

  etest = mean(abs(prob_test - test[,which(names(test) == variable_respuesta)])/(max(ozone_test$ozone)-

}
```

Comenzamos planteando los modelos con uno sencillo, tratando de predecir la variable respuesta en función únicamente de aquella que recomienda regsubsets, 'temp':

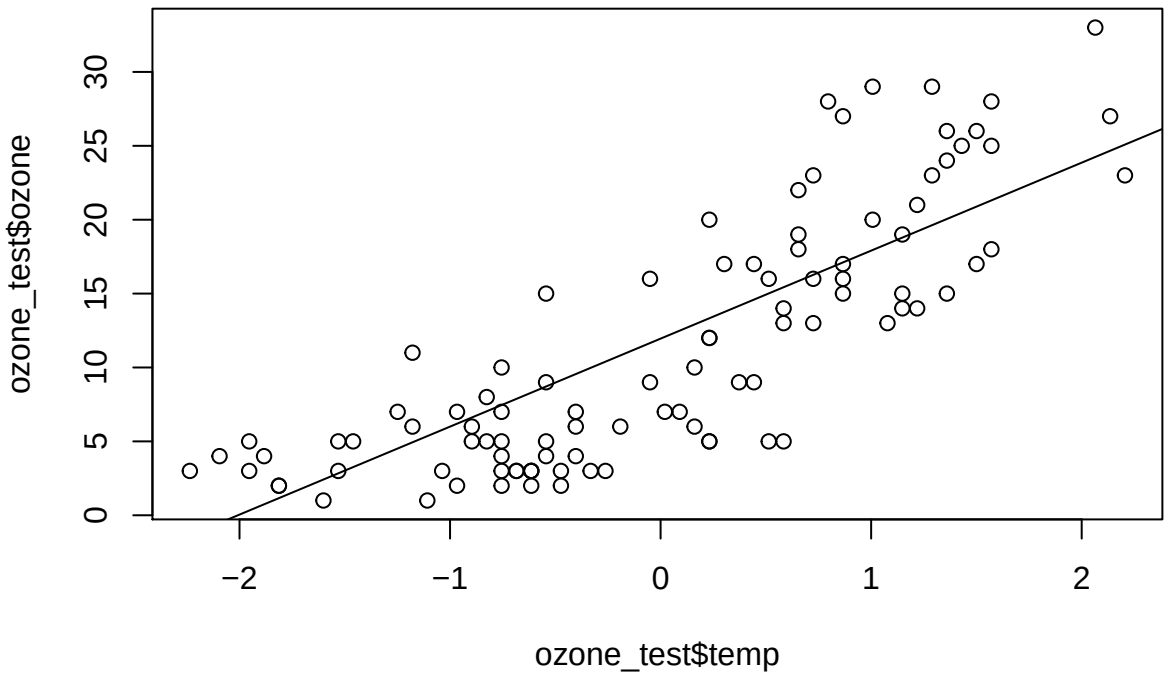
```
set.seed(5)
m1_ozone = lm(ozone ~ temp, data = ozone_train)

etest_m1ozone = calculoErrorMedioIntervalo(m1_ozone, ozone_test, "ozone")
etest_m1ozone
```

```
## [1] 0.128535
```

Representemos el modelo generado en función del atributo 'temp':

```
set.seed(5)
plot(ozone_test$temp, ozone_test$ozone)
abline(m1_ozone$coefficients)
```



Como podemos observar, se ajusta un modelo de regresión lineal en función de las variables respuestas, pero veamos ahora el error obtenido.

Obtenemos un error de 0.115, para una primera aproximación no está nada mal. No obstante, vamos a seguir haciendo modelos utilizando las agrupaciones de atributos más representativas. Añadimos al siguiente modelo el atributo 'ibh':

```
set.seed(5)
m2_ozone = lm(ozone ~ temp + ibh, data = ozone_train)

etest_m2ozone = calculoErrorMedioIntervalo(m2_ozone, ozone_test, "ozone")
etest_m2ozone
```

```
## [1] 0.123185
```

El error desciende hasta 0.109 cuando utilizamos dos características para predecir. Continuemos el proceso con las tres características más representativas, añadiendo 'humidity':

```
set.seed(5)
m3_ozone = lm(ozone ~ temp + ibh + humidity, data = ozone_train)

etest_m3ozone = calculoErrorMedioIntervalo(m3_ozone, ozone_test, "ozone")
etest_m3ozone
```

```
## [1] 0.1116171
```

Mejoramos el error, llegando hasta 0.1. Añadimos un nuevo atributo, llegando a una combinación lineal de 4, con 'vis' y evaluamos de nuevo el modelo que obtengamos:

```
set.seed(5)
m4_ozone = lm(ozone ~ temp + ibh + humidity + vis , data = ozone_train)

etest_m4ozone = calculoErrorMedioIntervalo(m4_ozone, ozone_test, "ozone")
etest_m4ozone
```

```
## [1] 0.1118312
```

En este punto llegamos al sobreajuste, ya que encontramos un error mayor que el del modelo anterior, 0.103. Llegados a este punto, en lugar de añadir más atributos a nuestro modelo de regresión de forma lineal, vamos a realizar transformaciones no lineales sobre el conjunto de atributos del mejor modelo obtenido hasta el momento:

```
set.seed(5)
m5_ozone = lm(ozone ~ temp * ibh * humidity, data = ozone_train)

etest_m5ozone = calculoErrorMedioIntervalo(m5_ozone, ozone_test, "ozone")
etest_m5ozone
```

```
## [1] 0.09824793
```

Con la transformación no lineal que hemos realizado (multiplicar las características entre sí) el error desciende hasta 0.828. La mejora es significativa, pero vamos a probar una serie de combinaciones no lineales por si alguna ofrece un mejor resultado. Comenzamos por añadir una nueva característica multiplicada, 'ibt':

```
set.seed(5)
m6_ozone = lm(ozone ~ temp * ibh * humidity * ibt , data = ozone_train)

etest_m6ozone = calculoErrorMedioIntervalo(m6_ozone, ozone_test, "ozone")
etest_m6ozone
```

```
## [1] 0.09276957
```

El error es muy similar pero ligeramente superior. Por último, comprobaremos cómo se ajusta un modelo con una característica más, 'vis':

```
set.seed(5)
m7_ozone = lm(ozone ~ temp * ibh * humidity * ibt * vis, data = ozone_train)
```

```
etest_m7ozone = calculoErrorMedioIntervalo(m7_ozone, ozone_test, "ozone")
etest_m7ozone
```

```
## [1] 0.1015987
```

Tras evaluar todos los modelos que hemos considerado, el que mejor resultados ha proporcionado para nuestro problema ha sido el que realiza el producto de las características 'temp', 'ibh' y 'humidity' (combinaciones no lineales entre atributos).

Estimacion del error E_{out}

Una vez que hemos hecho una selección de los modelos y hemos dado con uno modelo lineal que proporciona un valor del error inferior al 0.1 (concretamente 0.828), vamos a proceder a realizar una estimación del E_{out} de forma que obtengamos un valor más representativo del error.

Para ello vamos a repetir la generación de particiones, tanto train como test, con la misma proporción realizada anteriormente, empleando también un 0.7 para train y un 0.3 para test. Repetiremos el proceso 100 veces y obtendremos la media de los E_{out}.

Definamos una función que realice tanto la generación y el particionado, así como las transformaciones sobre los conjuntos de datos y que devuelva el error generado por el mejor modelo obtenido (m5):

```
set.seed(5)
generarErrorParticionOzone = function(datos){
  #Si queremos obtener un conjunto de indices train para luego ejecutar un modelo lineal sobre el train
  el_train = sample (nrow(datos), round(nrow(datos)*0.7))
  #definimos ambos conjuntos en dos data.frame diferentes:
  ozone_train = datos_ozone[el_train,]
  ozone_test = datos_ozone[-el_train,]

  #TRANSFORMACIONES:
  ozoneTrans = preProcess(ozone_train[,-which(names(datos_ozone) == "ozone")], method = c("BoxCox", "ce
ozone_train[,-which(names(ozone_train) == "ozone")] =predict(ozoneTrans,ozone_train[,-which(names(ozone_
ozone_test[,-which(names(ozone_test) == "ozone")] =predict(ozoneTrans,ozone_test[,-which(names(ozone_tes

#EVALUACION DEL MODELO
m5_ozone = lm(ozone ~ temp * ibh * humidity, data = ozone_train)

etest_m5ozone = calculoErrorMedioIntervalo(m5_ozone, ozone_test, "ozone")
etest_m5ozone
}
```

Una vez que hemos definido la funcion, realizamos una evaluación del error 1000 veces, de forma que cálculemos como resultado final la media de los errores generados para cada una de las 1000 particiones:

```
set.seed(5)
mean(replicate(100, generarErrorParticionOzone(datos_ozone)))
```

```
## [1] 0.1022679
```

Como podemos ver, obtenemos un E_{out} medio del 0.08869314, bástante bueno. Este error representa mejor el error real obtenido por nuestro modelo de regresión lineal.