Practica 3 - Ajuste de Modelos Lineales - Aprendizaje Automatico

Arthur M. Rodriguez Nesterenko - DNI:Y1680851W Carlos M. Sequi Sanchez - DNI:20486926K

23 de mayo de 2017

Este ejercicio se centra en el ajuste de un modelo lineal a conjuntos de datos dadas con el objetivo de obtener el mejor predictor posible. En todos los casos los pasos a desarrollar seran aquellos que nos conduzcan al ajuste y seleccion del mejor modelo y a la estimacion del error E_{out} del modelo final. Para ello hemos elegido un problema de clasificacion y otro de regresion: para el primero hemos escogido el dataset de **South African Heart Disease** y para el segundo **LAozone**.

Problema 1: clasificacion (South Africa Heart Disease)

1.1 Descripcion inicial de los atributos del primer conjunto de datos a tratar:

-famHist: Representa el historial familiar del individuo en cuanto a cardiopatias coronarias ocurridas en su familia, en un factor con etiquetas "ausente" y "presente".

-sbp: Representa la presion arterial sistolica (valores enteros comprendidos entre [101,218], con una media de 138).

-tobacco: Consumo acumulado de tabaco medido en Kg (valores reales en el rango [0,31.2]).

-ldl: Valores de colesterol malo (a mayor valor, peor para la salud). (valores reales en el rango [0.98,15.33]).

-adiposity: indice de masa corporal, valores por encima de 30Kg/m^2 consideran al individuo como adiposo. (valores reales en el rango [6.74,42.49]).

-obesity: Medida de la obesidad del individuo. (valores reales comprendidos en el rango [14.7,46.58]).

-alcohol: Medida del consumo de alcohol actual (valores reales en el rango[0,147.19]).

-age: Edad de los individuos a estudiar. (valores enteros en el rango [15,64]).

-typea: Patron de comportamiento asociado al desarrollo de la cardiopatia coronaria (valores enteros en el rango [13,78])

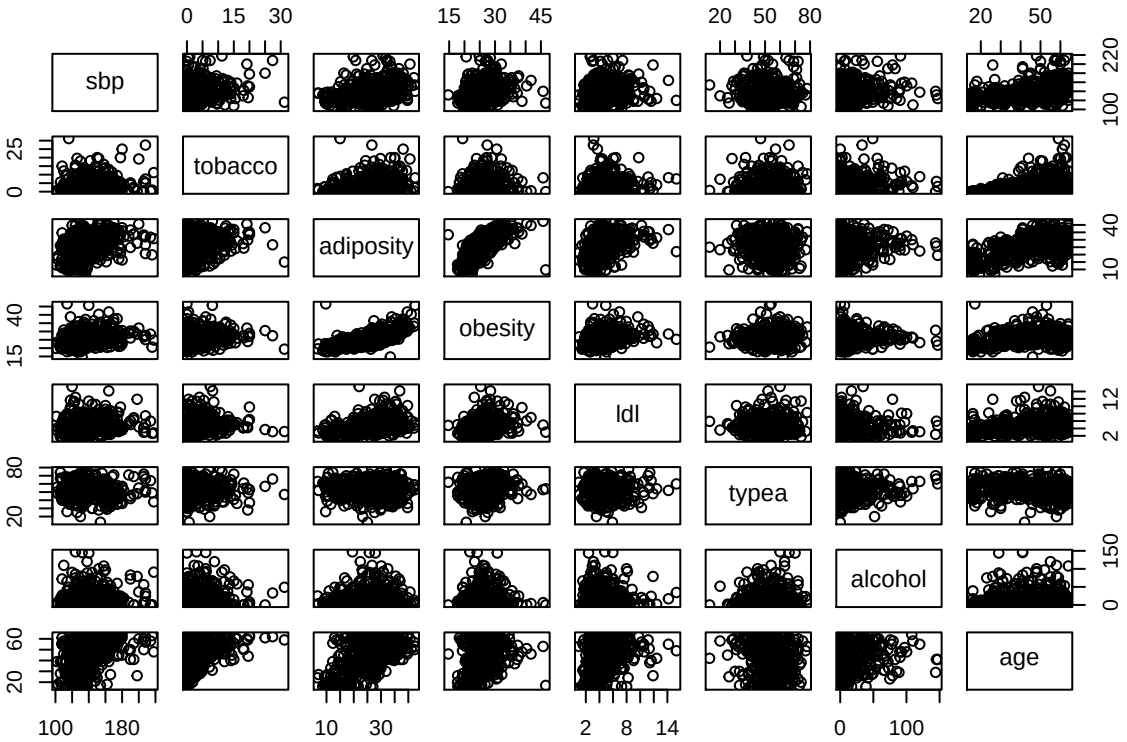
-chd: variable de respuesta (clase) que indica que el sujeto de estudio ha sufrido (1) o no ha sufrido (0) una cardiopatia coronaria en algun momento previo al estudio.

Se elimina la primera columna, ya que representa unicamente un identificador de tupla y se transforma el atributo cualitativo $famhist$ a un atributo cuantitativo ($famHist$) de tipo binario (0 o 1).

```r
set.seed(3)

# Leemos los datos
SAHeart = read.csv("./datos/SAHeart.csv")

# Eliminamos la primera columna (un identificador de la tupla)
# Además, dado que una de las caracteristicas (famhist) tiene
# un dominio binario cualitativo, vamos a cambiarlas por 0 y 1.
SAHeart = SAHeart[,!colnames(SAHeart)=="row.names"]
SAHeart = data.frame(famHist = (ifelse(SAHeart$famhist=="Absent",0,1)),SAHeart)
SAHeart = SAHeart[,!colnames(SAHeart)=="famhist"]
SAHeartLabels = SAHeart$chd
```

Utilizamos la funcion pairs para realizar una representacion "todos frente a todos", mostrando los diagramas de dispersion de cada atributo en funcion de los demas. En nuestro caso, el atributo $famHist$, que es de tipo binario, no nos aporta informacion a la hora de hacer lo diagramas de dispersion para identificar las correlaciones existentes entre atributos y por tanto sera descartado.

```r
# Ejecucion de pairs para obtener los diagramas de dispersion
# Se enfrenta cada atributo con todos los demas
pairs(~ sbp + tobacco +adiposity + obesity + ldl + typea + alcohol + age, data = SAHeart)
```



Como primera aproximacion podemos observar como la *obesity* y *adiposity* pueden ser un buen par de atributos representativos para realizar el proceso de aprendizaje a partir de ellos o bien puede ser que sean dos atributos demasiado correlados y por tanto sea necesario prescindir de alguno de ellos para evitar la redundancia de datos y el empobrecimiento de nuestro modelo de ajuste lineal. Con los posteriores experimentos se podra emitir una decision mucho mas concreta en cuanto a estos dos atributos.

1.2 Creacion de particiones de Train y Test

Las particiones de entrenamiento y test se distribuiran de la siguiente manera: un 80% de los datos, elegidos de forma aleatoria, representaran el conjunto de aprendizaje, mientras que el 20% restante sera asignado al conjunto de Test. Guardamos las etiquetas de cada uno de los subconjuntos para posteriomente utilizarlas en el proceso de aprendizaje de los modelos de ajuste y estimacion de error,

```
# Utilizamos el 80% de las muestras del conjunto de datos
# para crear el conjunto Train y las restantes para el Test
porcentajeMuestrasTrain = 0.8
train = sample(nrow(SAHeart), round(nrow(SAHeart)*porcentajeMuestrasTrain))
SAHeart.Train = SAHeart[train,]
SAHeart.Test = SAHeart[-train,]

# Guardamos las etiquetas del train y test en estructuras aparte
SAHeartLabels.Train = SAHeartLabels[train]
SAHeartLabels.Test = SAHeartLabels[-train]
```

1.3. Preprocesado de los datos

Una vez creadas las particiones realizaremos una primera prueba de preprocesado de los datos. Este paso comprende la utilizacion de distintas funciones que realizan distintas transformaciones sobre el data set para que sean lo mas simetricos, cercanos a la media y con varianza unitaria posible.

1.3.1 Preprocesado de los datos: transformacion de asimetria con BoxCoxTrans

La funcion *BoxCoxTrans* transforma los datos de tal manera que estos acaben siendo lo mas simetricos posible. Esto se consigue evaluando el *skewness* de un atributo y estimando un valor λ de forma que se aplique el mismo en una transformacion que reduzca la asimetria de los datos de un atributo. Realizamos dos pruebas sobre dos atributos, uno al que no se le aplica transformacion BoxCox y otro al que si se aplica.

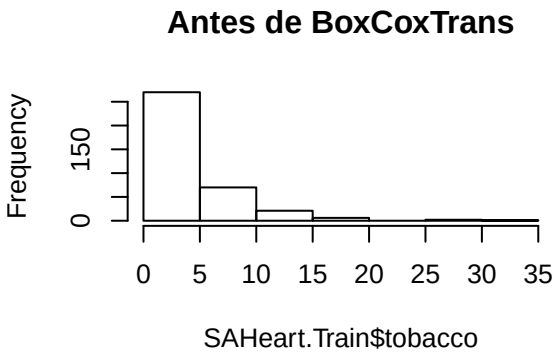
```
# Aplicacion de la funcion BoxCox a un atributo elegido tras realizar
# distintas pruebas y al que no se le aplica transformacion
par(mfrow=c(2,2))
hist(SAHeart.Train$tobacco, main = "Antes de BoxCoxTrans")
tobacco_trans = BoxCoxTrans(SAHeart.Train$tobacco)
tobacco_trans
```

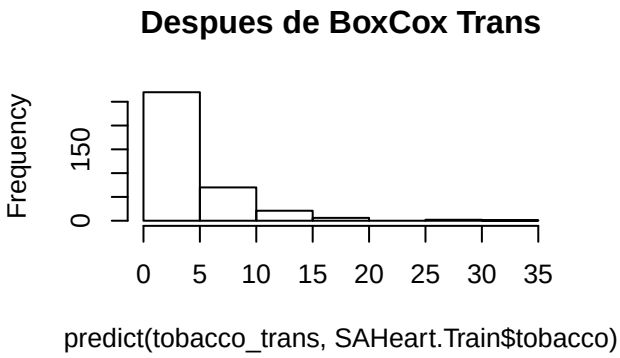
```
## Box-Cox Transformation
##
## 370 data points used to estimate Lambda
##
## Input data summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.050   1.735   3.531   5.375  31.200
##
## Lambda could not be estimated; no transformation is applied
```

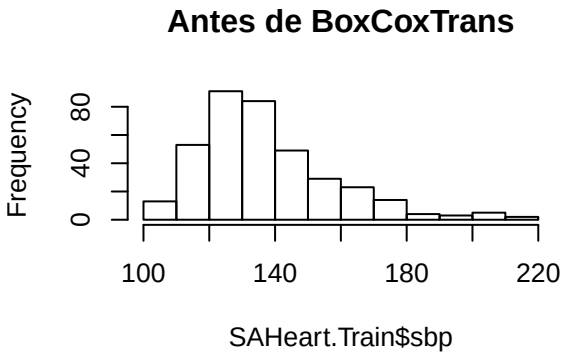
```
hist(predict(tobacco_trans,SAHeart.Train$tobacco),main = "Despues de BoxCox Trans")

hist(SAHeart.Train$sbp,main = "Antes de BoxCoxTrans")
sbp_trans = BoxCoxTrans(SAHeart.Train$sbp)
sbp_trans
```

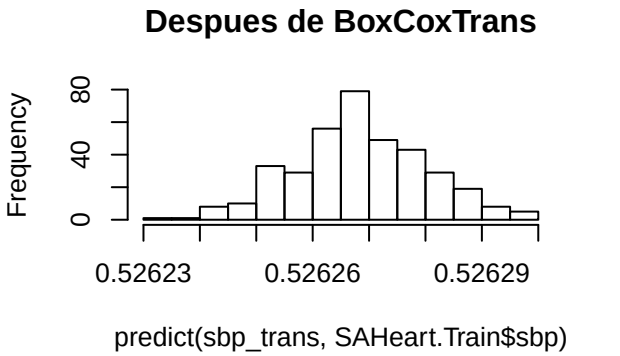
```
## Box-Cox Transformation
##
## 370 data points used to estimate Lambda
##
## Input data summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   101.0   124.0   134.0   138.3   148.0   218.0
##
## Largest/Smallest: 2.16
## Sample Skewness: 1.2
##
## Estimated Lambda: -1.9
```

```
hist(predict(sbp_trans,SAHeart.Train$sbp),main = "Despues de BoxCoxTrans" )
```









¿Por que la funcion BoxCoxTrans no aplica la transformacion de simetria sobre el atributo tobacco? Esto es debido a que hay valores menores o iguales a cero dentro de dicho atributo, por lo que la funcion no es capaz de estimar el valor λ para realizar la transformacion. En el caso del atributo sbp, los valores de los atributos si permiten a la funcion BoxCoxTrans calcular el valor de lambda para poder aplicar la transformacion.

1.3.2 Preprocesado de los datos: Reduccion de dimensionalidad, escalado y normalizacion

Para esta fase del preprocesado nos valdremos de la funcion *prcomp* para aplicar el algoritmo de analisis de componentes principales junto con el escalado y la normalizacion. A continuacion explicaremos brevemente cada uno.

Reduccion de dimensionalidad:

Aplicamos el algoritmo *prcomp* para extraer las componentes principales del algoritmo, componentes que se construyen como combinaciones de los atributos basicos, cada uno en una determinada proporcion. Aquel atributo en cuya fila exista una mayor cantidad de valores cercanos a cero, se considerara como posible descarte.

Escalado

Divide cada valor por la desviacion tipica para conseguir una varianza unitaria, evitando que la magnitud en la que los datos estan medidos influyan en el analisis. Se indica con el argumento *scale*.

Normalizacion

Se le resta a cada valor de la caracteristica la media del conjunto. Esto se indica con el argumento **center**.

La prueba se puede ver a continuacion, los resultados seran comentados posteriormente.

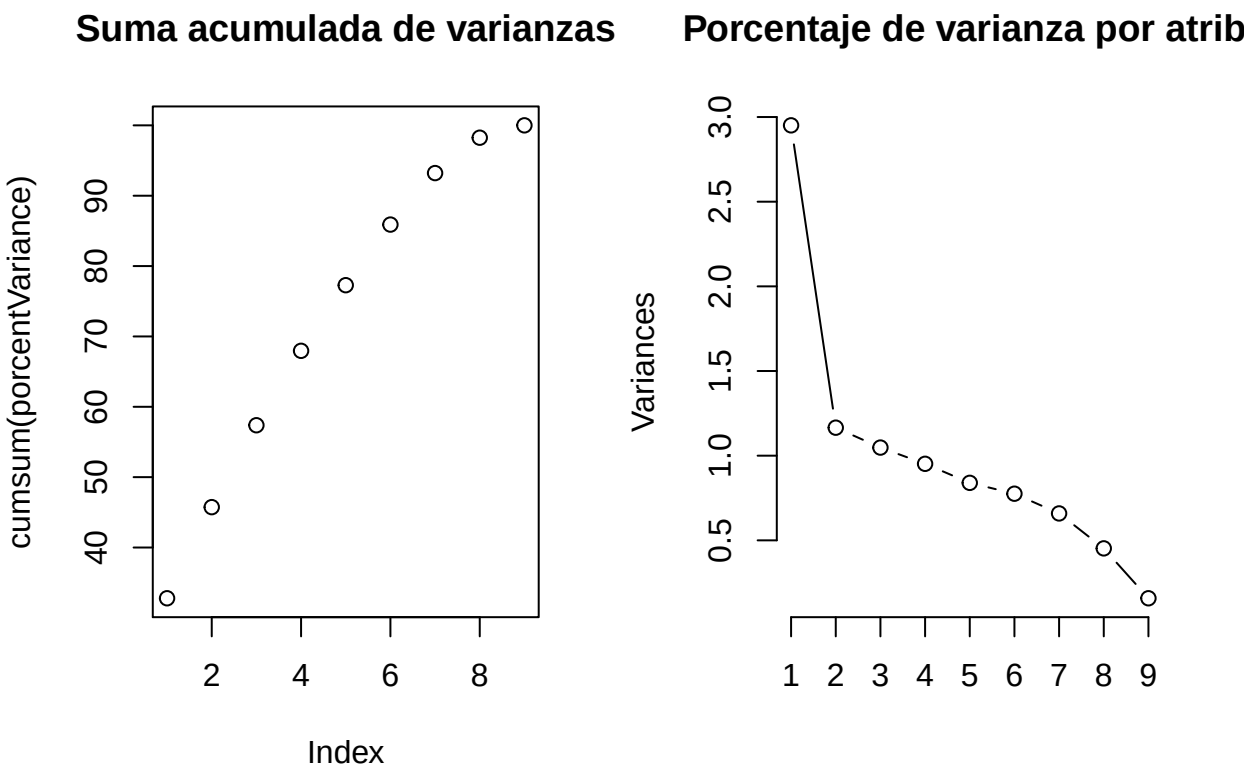
```
# Ejecucion del algoritmo PCA aplicando escalado y normalizacion.
pcaObject = prcomp(SAHeart.Train[,!colnames(SAHeart.Train)=="chd"], center = TRUE,
                   scale = TRUE)
attributes(pcaObject)
```

```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

```
# A continuacion obtenemos el porcentaje con el que cada uno de los
# atributos contribuye a la varianza total.
porcentVariance = pcaObject$sdev^2/sum(pcaObject$sdev^2)*100
porcentVariance
```

```
## [1] 32.787344 12.948216 11.642232 10.577180  9.327059  8.614896  7.321078
## [8]  5.027586  1.754410
```

```
# Representacion de los porcentajes de las varianzas
par(mfrow=c(1,2))
plot(cumsum(porcentVariance),main = "Suma acumulada de varianzas")
plot(pcaObject,type="l",main = "Porcentaje de varianza por atrib.")
```



Como podemos ver en los siguientes graficos:

Grafico 1(izquierda): a mayor numero de atributos, mayor es el porcentaje de explicacion de la varianza total de los datos, cuanto mayor sea esta explicacion, mayor poder de generalizacion tendremos. Por ejemplo, con 7 atributos obtenemos un porcentaje del 93% aproximadamente, y con 8 un porcentaje de 98%, el cual es un alto porcentaje sin tener en cuenta uno de los atributos, por lo que mas tarde asignaremos un threshold del 86% (experimentacion) a la hora de realizar la reduccion de dimensionalidad de forma automatica.

Grafico 2(derecha): se representa el porcentaje de contribucion a la varianza por parte de cada uno de los atributos. Por ejemplo, el atributo uno tiene un porcentaje de contribucion del 32%, mientras que el atributo 4 aporta un 10% a la varianza.

1.3.3 Preprocesado de los datos: automatico

Para el preprocesado automatico de los datos utilizamos directamente la funcion **preProcess** del paquete *caret*, la cual se encarga de realizar automaticamente las operaciones que se le indiquen como argumento, en nuestro caso las siguientes: transformaciones de simetria, escalado, normalizacion y reduccion de dimensionalidad.

El preprocesado de los datos se puede apreciar a continuacion, con los resultados debidamente comentados posteriormente.

```
# El valor thres indica el porcentaje de variabilidad de los datos que tendran que
# explicar las componentes elegidas por el algoritmo, en nuestro caso usaremos thres=0.86
thres = 0.86
objetoTrans = preProcess(SAHeart.Train[,!colnames(SAHeart.Train)=="chd"],
                         method = c("BoxCox", "center", "scale", "pca"), thres)
SAHeart.Train2 = predict(objetoTrans, SAHeart.Train)

# Eliminamos las variables con varianza 0 o muy proximas.
nearZeroVar(SAHeart.Train2)
```

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

Mostramos la matriz de rotacion (rotation) que representa cuanto participa cada uno de los atributos del conjunto de datos en la descripcion de las componentes principales ofrecidas por el algoritmo PCA.

```
# Para simplificar y prescindir del prefijo SAHeart
attach(SAHeart.Train2)
# Matriz de rotacion.
pcaObject$rotation
```

```
##                    PC1          PC2         PC3         PC4        PC5
## famHist   -0.195883850  0.003749386 -0.26555035  0.85267174 -0.3340945
## sbp       -0.313247341  0.241196602  0.05170311 -0.25633338 -0.1047889
## tobacco   -0.291879231  0.456031288 -0.10577419  0.07029626  0.6391042
## ldl       -0.330629631 -0.383456722  0.07118940  0.02116946  0.2047570
## adiposity -0.514782499 -0.166319976  0.10520751 -0.10889306 -0.1331988
## typea      0.000278667 -0.379126638 -0.79327722 -0.10326110  0.3323618
## obesity   -0.419991706 -0.359273775 -0.01594416 -0.24614045 -0.2747731
## alcohol   -0.145249516  0.504966062 -0.49770009 -0.28679507 -0.4286909
## age       -0.454427803  0.177027783  0.14911982  0.19104270  0.1988417
##                   PC6         PC7        PC8         PC9
## famHist    0.1024891 -0.03006458  0.2005149 -0.02856741
## sbp        0.8008237 -0.27185775  0.2210574 -0.01574809
## tobacco   -0.2116674  0.15984914  0.4584913 -0.04109659
## ldl       -0.2662608 -0.78732504  0.0546771  0.04937975
## adiposity -0.1034643  0.24696197 -0.1375480 -0.75898214
## typea      0.2551523  0.09549410 -0.1725520 -0.04302232
## obesity   -0.1324200  0.37431217  0.3619518  0.51916852
## alcohol   -0.3622364 -0.23484731 -0.1501839  0.03543291
## age        0.0890039  0.11993673 -0.7038932  0.38224646
```

A la vista de los resultados de la matriz de rotacion devuelta por el algoritmo PCA (realizado por separado en el apartado 1.3.2) podemos observar que todos los atributos participan lo suficiente en todas las componentens principales PC_i generados por PCA como para no poder prescindir de ninguno de los atributos iniciales. Esto es debido a que no hay ninguno de los atributos que tenga todos sus valores de participacion en cada PC_i muy proximos a 0.

Llegados a este punto, y considerando los resultados obtenidos, hemos tomado la decision de trabajar con el conjunto de datos original, esto es, sin eliminar ningun atributo, pero si considerando la aplicacion de la transformacion de asimetria mediante BoxCox, el escalado y la normalizacion.

1.3.4 Preprocesado de los datos: conjunto original

La decision de aplicar la transformacion de simetria, el escalado y normalizacion de los datos originales implica nuevamente la ejecucion de la funcion *preProcess* pero esta vez solo con los metodos indicados. Una vez realizado el preprocesamiento de los datos, podemos empezar a trabajar con nuestra particion de Train.

```
# Preprocesado automatico
objetoTrans = preProcess(SAHeart.Train[,!colnames(SAHeart.Train)=="chd"],
                         method = c("BoxCox", "center", "scale"))
SAHeart.TrainProcessed = predict(objetoTrans, SAHeart.Train)

# Comprobamos que no exista ninguna componente con varianza 0
nearZeroVar(SAHeart.TrainProcessed)
```

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

1.4. Seleccion de clases de funciones a usar.

Este problema de clasificacion sera abordado considerando modelos de regresion lineal y de regresion logistica, con el objetivo de establecer una comparativa entre los resultados de ambos y a partir de los mejores modelos "simples" de cada uno, optimizar los ajustes aplicando transformaciones no lineales, en concreto, de tipo cuadraticas, cubicas y productos de potencias.

1.5. Necesidad de regularizacion

El paquete glmnet contiene la funcion glmnet() mediante la cual podemos obtener un modelo de ajuste lineal al que se le puede aplicar regularizacion mediante Weight Decay (ridge regression) o Lasso. Dicha funcion recibe una matriz de datos matrizX, la variable respuesta vectorY, el valor de α el cual indica si utilizaremos el metodo de regularizacion Weight Decay (ya usado en la anterior practica) o el metodo de regularizacion Lasso. Ademas recibira un valor lambda el cual calcularemos previamente con la funcion cv.glmnet(). Esta ultima funcion nos devuelve el mencionado valor lambda para el que, usando 10 fold cross-validation, se obtiene el menor porcentaje de error medio sobre los datos.

```
# fijamos la semilla aleatoria
set.seed(14)

# Creacion de conjuntos
matrizX = model.matrix(chd ~ ., SAHeart.Train)
vectorY = SAHeart.Train$chd

# Obtenemos el mejor lambda mediante validacion cruzada
# para el cual el error es minimo sobre los datos.
# utilizamos un alpha = 0 para indicar que queremos usar Weight Decay.
cv.out = cv.glmnet(matrizX, vectorY, alpha = 0)
# calculamos el mejor lambda.
bestlam =cv.out$lambda.min
print(sprintf("Mejor valor lambda obtenido: %f", bestlam))
```

```
## [1] "Mejor valor lambda obtenido: 0.138952"
```

Como bien hemos calculado, el valor del lambda para el cual se cumple el menor error en la validacion cruzada es $\lambda = 0.138952$, por lo tanto este valor es el que vamos a pasarle a la funcion glmnet para comprobar, a traves del ratio de la desviacion de los datos, si aplicar regularizacion aportara mejores resultados en funcion de la tasa de clasificacion o realmente no influira en la misma. En caso de que la diferencia de las desviaciones sea significativa, entonces aplicar regularizacion a un modelo de regresion (ya sea lineal o logistica) permitira obtener mejores resultados en cuanto al error fuera de la muestra.

```
# El primer paso consiste en ejecutar glmnet con lambda = 0 para obtener la desviacion
# sin considerar regularizacion alguna
desv_GLMnet_NoReg = glmnet(matrizX, vectorY, alpha = 0, lambda = 0)
print(sprintf("Desviacion sin considerar regularizacion: %f ",desv_GLMnet_NoReg$dev.ratio))
```

```
## [1] "Desviacion sin considerar regularizacion: 0.226407 "
```

```
# Comprobamos como varia el valor de desviacion si se considera regularizacion
desv_GLMnet_REG= glmnet(matrizX, vectorY, alpha = 0, lambda = bestlam)
print(sprintf("Desviacion considerando regularizacion: %f ",desv_GLMnet_REG$dev.ratio))
```

```
## [1] "Desviacion considerando regularizacion: 0.220503 "
```

Como vemos, la diferencia de los resultados de las desviacions con y sin regularizacion no son significativos

por lo que, como hemos dicho previamente, el hecho de aplicar regularizacion no influira para nada en el resultado del error fuera de la muestra.

1.6. Definicion de modelos de ajuste y estimacion de parametros

El primer paso que vamos a considerar en el proceso de aprendizaje sera la seleccion de modelos a traves de la funcion *regsubsets*, que permite elegir aquellos atributos que mejor se ajustan a distintos modelos lineales: lo hace de forma exhaustiva (para cada tamaño de subconjunto, se seleccionan las combinaciones de atributos optimas para aprender el modelo), hacia adelante (forward: selecciona el mejor atributo para tamaño 1, hace lo mismo para los sucesivos tamaños pero arrastra los atributos seleccionados anteriormente) o hacia atras (backwards: lo contrario que forward) o de forma secuencial. Vamos a realizar una primera ejecucion con el metodo exhaustivo e interpretar los resultados para elegir aquellos atributos que sean los mejores candidatos para aprender modelos lineales.

```
# Nuestra variable de respuesta (clase) son las etiquetas que indican
# si una persona padeceria o no una cardiopatia coronaria
pruebaRegSubsets1 = regsubsets(chd ~ . , data=SAHeart.TrainProcessed,
                               method = "exhaustive")
summary(pruebaRegSubsets1)
```

```
## Subset selection object
## Call: regsubsets.formula(chd ~ ., data = SAHeart.TrainProcessed, method = "exhaustive")
## 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 ) "*"     " " "*"     "*" "*"       "*"   "*"     "*"     "*"
```

Cada fila muestra la dimension del subconjunto de atributos que se pueden utilizar para aprender y, en cada caso, cuales son los mejores candidatos. Vamos a utilizar subconjuntos de tamaño 1, 2, 3 y 5 para aprender modelos de Regresion Lineal y de Regresion Logistica, a partir de los cuales intentaremos predecir el E_{out} utilizando distintas metricas como la matriz de confusión y el porcentaje de etiquetas mal clasificadas. Procesamos los datos del Test para realizar la evaluacion de los modelos lineales que aprendamos en el siguiente punto:

```
# no consideramos la variable respuesta: chd
objetoTrans_Test = preProcess(SAHeart.Test[,!colnames(SAHeart.Test)=="chd"],
```

```
                          method = c("BoxCox", "center", "scale"))
SAHeart.TestProcessed = predict(objetoTrans_Test, SAHeart.Test)
```

1.6.1 Aprendizaje de Modelos de Regresion Lineal

Aprenderemos 4 modelos lineales teniendo en cuenta los atributos que forman parte de los subconjuntos de tamaño 1,2, 3 y 5.

```
# Nuestra variable de respuesta seran las etiquetas del Train
# Aprendemos el modelo Lineal con el subconjunto de tamaño 1 = age
modeloLineal1 = lm(SAHeartLabels.Train ~ age, data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 2 = age + famHist
modeloLineal2 = lm(SAHeartLabels.Train ~ age + famHist, data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 3 = age + famHist + ldl
modeloLineal3 = lm(SAHeartLabels.Train ~ age + famHist + ldl, data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 5
modeloLineal4 = lm(SAHeartLabels.Train ~ age + famHist + ldl + tobacco + typea,
                   data = SAHeart.TrainProcessed)
```

Creamos la funcion que calcula la evaluacion de nuestro modelo lineal. Consiste en calcular la tasa de prediccion de los mismos para compararlos, calculando previamente el conjunto de probabilidades sobre el conjunto de Test procesado y mostrando la matriz de confusion.

```
evaluacionModelo = function(modeloTrain, datosTest, etiquetasTest, indiceModelo)
{
  # Calculamos la probabilidad del modelo lineal
  probTest = predict(modeloTrain, data.frame(datosTest), type="response")
  # Calculamos la prediccion con el modelo lineal: inicializadas a 0
  predTest = rep(0, length(probTest))
  #Calculo de las predicciones reales
  predTest[probTest >=0.5] = 1 # >= 0.5 clase 1
  # Matriz de confusion
  print(sprintf("Matriz de confusion del Modelo %i", indiceModelo))
  print(table(predTest, etiquetasTest))
  # Calculo del Eout con el modelo lineal
  Eval = mean(predTest != etiquetasTest)
  print(sprintf("Evaluacion (Porcentaje de Error) del ML-%i -> %f",indiceModelo, Eval*100))
  cat("---------------------------------- \n")
}
```

Una vez tenemos todo lo necesario, evaluamos los modelos y comparamos resultados.

```
#Procedemos a evaluar los 4 modelos lineales aprendidos
evaluacionModelo(modeloLineal1, SAHeart.TestProcessed, SAHeartLabels.Test, 1)
```

```
## [1] "Matriz de confusion del Modelo 1"
##         etiquetasTest
## predTest  0  1
##        0 47 22
##        1 10 13
## [1] "Evaluacion (Porcentaje de Error) del ML-1 -> 34.782609"
```

```
## ------------------------------------
```

```
evaluacionModelo(modeloLineal2, SAHeart.TestProcessed, SAHeartLabels.Test,2)
```

```
## [1] "Matriz de confusion del Modelo 2"
##         etiquetasTest
## predTest  0  1
##        0 52 21
##        1  5 14
## [1] "Evaluacion (Porcentaje de Error) del ML-2 -> 28.260870"
## ------------------------------------
```

```
evaluacionModelo(modeloLineal3, SAHeart.TestProcessed, SAHeartLabels.Test,3)
```

```
## [1] "Matriz de confusion del Modelo 3"
##         etiquetasTest
## predTest  0  1
##        0 50 19
##        1  7 16
## [1] "Evaluacion (Porcentaje de Error) del ML-3 -> 28.260870"
## ------------------------------------
```

```
evaluacionModelo(modeloLineal4, SAHeart.TestProcessed, SAHeartLabels.Test,4)
```

```
## [1] "Matriz de confusion del Modelo 4"
##         etiquetasTest
## predTest  0  1
##        0 53 18
##        1  4 17
## [1] "Evaluacion (Porcentaje de Error) del ML-4 -> 23.913043"
## ------------------------------------
```

Aplicando la regresion lineal simple parece ser que el modelo no se ajusta lo suficientemente bien a los datos de entrenamiento y, por tanto, tiene un error fuera de la muestra lo suficientemente grande como para decir que el conjunto de datos (hasta el momento) no es linealmente separable. Continuamos aplicando la regresion logistica para intentar ajustar un mejor modelo.

1.6.2 Aprendizaje de modelos de Regresion Logistica

A continuacion aprenderemos otros cuatro modelos con regresion logistica y los mismos atributos considerados en la regresion lineal (los mejores obtenidos con regsubset). Usaremos la familia de funciones binomiales para explicar la distribucion de probabilidad del error.

```
# Variable de respuesta = etiquetas del Train
# Aprendemos el modelo de Regresion Logistica con el subconjunto de tamaño 1 = age
# y una familia de funciones binomiales
modeloRegLog1 = glm(SAHeartLabels.Train ~ age, family = binomial(logit) ,
                    data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 2 = age + famHist
modeloRegLog2 = glm(SAHeartLabels.Train ~ age + famHist, family = binomial(logit) ,
                    data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 3 = age + famHist + ldl
modeloRegLog3 = glm(SAHeartLabels.Train ~ age + famHist + ldl,family = binomial(logit) ,
                    data = SAHeart.TrainProcessed)
```

```r
# Subconjunto de tamaño 5
modeloRegLog4 = glm(SAHeartLabels.Train ~ age + famHist + ldl + tobacco + typea,
                    family = binomial(logit) ,data = SAHeart.TrainProcessed)
```

Los modelos de regresion Logistica aprendidos con los "mejores subconjuntos" seran evaluados en terminos del error fuera de la muestra, utilizando las matrices de confusion y el porcentaje de etiquetas mal clasificadas tal como hicimos para los modelos de regresion lineal.

```r
#Procedemos a evaluar los 4 modelos de regresion logistica aprendidos
evaluacionModelo(modeloRegLog1, SAHeart.TestProcessed, SAHeartLabels.Test, 1)
```

```
## [1] "Matriz de confusion del Modelo 1"
##         etiquetasTest
## predTest  0  1
##        0 47 22
##        1 10 13
## [1] "Evaluacion (Porcentaje de Error) del ML-1 -> 34.782609"
## ------------------------------------
```

```r
evaluacionModelo(modeloRegLog2, SAHeart.TestProcessed, SAHeartLabels.Test,2)
```

```
## [1] "Matriz de confusion del Modelo 2"
##         etiquetasTest
## predTest  0  1
##        0 51 20
##        1  6 15
## [1] "Evaluacion (Porcentaje de Error) del ML-2 -> 28.260870"
## ------------------------------------
```

```r
evaluacionModelo(modeloRegLog3, SAHeart.TestProcessed, SAHeartLabels.Test,3)
```

```
## [1] "Matriz de confusion del Modelo 3"
##         etiquetasTest
## predTest  0  1
##        0 50 19
##        1  7 16
## [1] "Evaluacion (Porcentaje de Error) del ML-3 -> 28.260870"
## ------------------------------------
```

```r
evaluacionModelo(modeloRegLog4, SAHeart.TestProcessed, SAHeartLabels.Test,4)
```

```
## [1] "Matriz de confusion del Modelo 4"
##         etiquetasTest
## predTest  0  1
##        0 52 18
##        1  5 17
## [1] "Evaluacion (Porcentaje de Error) del ML-4 -> 25.000000"
## ------------------------------------
```

Contemplando los resultados obtenidos entre los modelos de Regresion Lineal y Regresion Logistica, optamos por elegir los 3 mejores modelos de regresion lineal para intentar mejorarlos aplicando transformaciones no lineales.

1.7. Seleccion y ajuste del modelo final

El siguiente paso consiste en aprender modelos lineales aplicando transformaciones no lineales sobre los atributos que forman parte de los subconjuntos de tamaño 1, 3, 4 y 5 de algunos de los modelos ya probados con la regresión lineal anteriormente. Utilizaremos transformaciones no lineales como pueden ser cuadráticas, cúbicas o productos de potencias.

```
# Subconjunto de tamaño 1
modeloLinealTrans1 = lm(SAHeartLabels.Train ~ I(age^2) , data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 3
modeloLinealTrans2 = lm(SAHeartLabels.Train ~ age + famHist + ldl + I(age^3) + I(ldl^3)
                        + (I(age^3)*I(ldl^2)), data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 5
modeloLinealTrans3 = lm(SAHeartLabels.Train ~ age + famHist + ldl + tobacco + typea  +
                          I(age^3) + I(typea^3), data = SAHeart.TrainProcessed)

# Subconjunto de tamaño 4
modeloLinealTrans4 = lm(SAHeartLabels.Train ~ I(age^2) + I(famHist^2) + I(ldl^2) +
                          I(tobacco^2), data = SAHeart.TrainProcessed)
```

1.8. Estimacion del error Eout del modelo.

Una vez aprendidos los 4 modelos el siguiente paso consiste en calcular la tasa de prediccion de los mismos para compararlos, calculando previamente el conjunto de probabilidades sobre el conjunto de Test procesado.

```
#Procedemos a evaluar los 4 modelos de regresion logistica aprendidos
evaluacionModelo(modeloLinealTrans1, SAHeart.TestProcessed, SAHeartLabels.Test, 1)
```

```
## [1] "Matriz de confusion del Modelo 1"
##         etiquetasTest
## predTest  0  1
##        0 57 35
## [1] "Evaluacion (Porcentaje de Error) del ML-1 -> 38.043478"
## ------------------------------------
```

```
evaluacionModelo(modeloLinealTrans2, SAHeart.TestProcessed, SAHeartLabels.Test,2)
```

```
## [1] "Matriz de confusion del Modelo 2"
##         etiquetasTest
## predTest  0  1
##        0 50 19
##        1  7 16
## [1] "Evaluacion (Porcentaje de Error) del ML-2 -> 28.260870"
## ------------------------------------
```

```
evaluacionModelo(modeloLinealTrans3, SAHeart.TestProcessed, SAHeartLabels.Test,3)
```

```
## [1] "Matriz de confusion del Modelo 3"
##         etiquetasTest
## predTest  0  1
##        0 52 18
##        1  5 17
## [1] "Evaluacion (Porcentaje de Error) del ML-3 -> 25.000000"
```

```
## ------------------------------------
```

```
evaluacionModelo(modeloLinealTrans4, SAHeart.TestProcessed, SAHeartLabels.Test,4)
```

```
## [1] "Matriz de confusion del Modelo 4"
##         etiquetasTest
## predTest  0  1
##        0 41 16
##        1 16 19
## [1] "Evaluacion (Porcentaje de Error) del ML-4 -> 34.782609"
## ------------------------------------
```

Como podemos observar, ninguna de las pruebas realizadas reduce la mejor tasa de error encontrada con el modelo de regresión lineal (23% aprox.). Debido a esto, parece ser que estamos ante un caso de dataset el cual no puede ser separado linealmente con alta tasa de acierto, por lo que no podremos aprender bien una funcion que consiga un porcentaje de error fuera de la muestra lo suficientemente pequeño como para ser considerado un buen ajuste.

1.8.1. Estimacion del Eout lo mas ajustada posible.

El objetivo de este punto sera ajustar la estimacion del Eout utilizando el mejor de los modelos anteriores (considerando los atributos y las transformaciones aplicadas) para aprender con distintos conjuntos de datos train y test generados en varias ejecuciones independientes. Para ser mas especificos crearemos en cada ejecucion de la funcion un conjunto de train y test nuevos, para aprender a partir de los datos del Train y el mejor modelo escogido, posteriormente validandolo con los datos del Test y estimando un error de la misma forma que haciamos anteriormente. Para finalizar calcularemos la media de todas estas validaciones, lo que esperamos que ajuste de manera mas precisa el error fuera de la muestra.

```
ajusteEoutClasificacion = function(dataset, datasetLabels)
{
  # Utilizamos el 80% de las muestras del conjunto de datos
  # para crear el conjunto Train y las restantes para el Test
  porcentajeMuestrasTrain = 0.8
  train = sample(nrow(dataset), round(nrow(dataset)*porcentajeMuestrasTrain))
  dataset.Train = dataset[train,]
  dataset.Test = dataset[-train,]

  # Guardamos las etiquetas del train y test en estructuras aparte
  datasetLabels.Train = datasetLabels[train]
  datasetLabels.Test = datasetLabels[-train]

  objetoTrans = preProcess(dataset.Train[,!colnames(dataset.Train)=="chd"],
                           method = c("BoxCox", "center", "scale"))
  dataset.TrainProcessed = predict(objetoTrans, dataset.Train)
  dataset.TestProcessed = predict(objetoTrans, dataset.Test)

  # Aprendemos el mejor modelo de regresion lineal
  modeloLineal = lm(datasetLabels.Train ~ age + famHist + ldl + tobacco +
                      typea, data = dataset.TrainProcessed)

  # Calculamos la probabilidad del modelo lineal
  probTest = predict(modeloLineal, data.frame(dataset.TestProcessed), type="response")
  # Calculamos la prediccion con el modelo lineal: inicializadas a 0
  predTest = rep(0, length(probTest))
```

```
  #Calculo de las predicciones reales
  predTest[probTest >=0.5] = 1 # >= 0.5 clase 1
  # Calculo del Eout con el modelo lineal
  Eval = mean(predTest != datasetLabels.Test)

}

# Llamamos a la funcion para calcular el Eout medio con 100 iteraciones
errorMedio = mean(replicate(100,ajusteEoutClasificacion(SAHeart, SAHeartLabels)))*100
print(sprintf("El error medio obtenido con 100 iteraciones es: %f", errorMedio))
```

```
## [1] "El error medio obtenido con 100 iteraciones es: 26.358696"
```

Finalmente obtenemos una media del valor Eout superior al mejor obtenido realizando un unico experimento.

1.9 Conclusiones finales.

Con las pruebas realizadas hasta este punto, no hemos conseguido encontrar un modelo lineal que se ajuste de manera correcta al conjunto de datos tratado, sin embargo, el que mejores resultados ha mostrado es el modelo lineal sin transformaciones no lineales y sin regularizacion, utilizando un subconjunto de 5 de los mejores atributos obtenidos con la funcion regsubsets(). Con este modelo, el error fuera de la muestra (Eout) es de 23.9% lo que indica que los datos no son linealmente separables y por tanto con modelos de ajuste lineal como son regresion logistica y lineal no podremos acotar el error fuera de la muestra mucho mas alla de lo que ya hemos conseguido acotar.

1.9.1 Validacion de las conclusiones.

¿Como hemos llegado a obtener dichas conclusiones?

Primeramente optamos por comprobar la necesidad de realizar regularizacion mediante el metodo weight decay y nos encontramos con que aplicando regularizacion (a traves de un parametro λ estimado como aquel que tras 10fold-cross validation obteniamos mejores resultados en el error medio con los datos) no conseguiriamos una mejora significativa respecto de la que obtendriamos con los modelos lineales simples.

Tras ello decidimos aplicar modelos de regresion lineal obteniendo una cota de error demasiado alta como para considerar los modelos un buen ajuste, por lo tanto optamos por realizar otras experimentaciones. Una de esas experimentaciones consistia en el aprendizaje de modelos de regresion logistica como posible alternativa a los pobres resultados obtenidos con los modelos de regresion lineal. Tras aplicar estos modelos, observamos que no solo no mejoraban las cotas de error de los modelos lineales, sino que empeoraban, por lo que tuvimos que decantarnos por probar con transofmraciones no lineales sobre los modelos lineales que nos ofrecian mejores resultados.

A la hora de realizar transformaciones no lineales sobre los mejores atributos escogimos una familia de funciones cuadraticas, cubicas y producto de potencias, sin embargo tampoco obtuvimos resultados mejores que los conseguidos con los modelos lineales.

Para obtener una medida representativa del error fuera de la muestra optamos por repetir el experimento 100 veces calculando en cada experimento (iteracion) un nuevo par de conjuntos Train-Test, de manera que obtuviesemos una alta generalizacion al entrenar nuestro mejor modelo obtenido anteriormente con cada uno de los conjuntos de Train y validandolo posteriormente con los sucesivos conjuntos de Test. Para terminar calcularíamos la media de estos errores obteniendo un porcentaje aproximadamente un 4% superior al error fuera de la muestra calculado con tan solo un experimento.

En conclusion: sin mejores modelos de regresion que los lineales, sin posibles transformaciones no lineales que aplicar sobre los mismos y sin necesidad de aplicar regularizacion nos encontramos con que el mejor modelo utilizado para aprender a partir del conjunto de datos dado es el modelo lineal que contempla cinco de los mejores atributos y que aun asi solo consigue una cota de error del 23.9% fuera de la muestra, lo que indica que nuestro dataset no es linealmente separable y el modelo aprendido no representa un buen ajuste.

Problema 2: Regresion (Los Angeles ozone).

2.1. Descripcion inicial de los atributos del segundo dataset a tratar:

Ozone: valor maximo diario de la media de cada hora.

vh: altura (medida en metros) a la que se registra una medida de 500 milibares de presion, mediciones hechas en la AFB (Base Aérea de Vandenberg)

wind: velocidad del viento (mph).

himidity: porcentaje de humedad en el aire.

temperature: temperatura medida en grados Fahrenheit en la base aerea de Sandburg.

ibh: altura de inversion(cambio de aire denso a aire menos denso) termica medida en pies.

dpg: gradiente de presion (mm Hg) medido desde la ciudad de Los Angeles a Daggett.

ibt: tempertaura de inversion(cambio de aire denso a aire menos denso) termica medida en Fahrenheit

vis: visibilidad en millas desde el aeropuerto de Los Angeles.

doy: dia del año en el que han sido tomadas las muestras (sera eliminado del dataset al no aportar informacion util)

Lectura de los datos.

```
# Leemos los datos
LAozone = read.csv("./datos/LAozone.csv")

# transformamos el dataset en un dataframe
LAozone = data.frame(LAozone)

# Eliminamos el atributo day of year (doy) ya que no
# aporta ninguna informacion util
LAozone = LAozone[,!colnames(LAozone) == "doy"]
```

2.2. Creacion de particiones de Train y Test

Las particiones de entrenamiento y Test se distribuiran de la misma manera que antes: un 80% de los datos, elegidos de forma aleatoria, representaran el conjunto de aprendizaje, mientras que el 20% restante sera asignado al conjunto de Test.

```
set.seed(14)
# Utilizamos el 80% de las muestras del conjunto de datos
# para crear el conjunto Train y las restantes para el Test
porcentajeMuestrasTrain = 0.8
train = sample(nrow(LAozone), round(nrow(LAozone)*porcentajeMuestrasTrain))
LAozone.Train = LAozone[train,]
LAozone.Test = LAozone[-train,]
```

2.3. Preprocesado de los datos

Preprocesamos los datos con la funcion preProcess para aplicar normalizacion, transformaciones de simetria y escalado de los datos. Observamos en la matriz de rotacion si hay atributos cuya participacion en cada uno de las componentes principales sea muy cercana a 0.

```
# Preproceado de los datos con un thres del 0.85
objetoTransOzone = preProcess(LAozone.Train[,!colnames(LAozone.Train)=="ozone"],
                method = c("BoxCox", "center", "scale","pca"),thres=0.95)

# Matriz de rotacion
objetoTransOzone$rotation
```

```
##                  PC1         PC2         PC3         PC4         PC5
## vh         0.45579389 -0.12709529  0.29087718 -0.25609907  0.28338772
## wind      -0.12983504  0.42905157  0.61386767  0.62937268  0.14151892
## humidity   0.18911817  0.57123744 -0.31512753 -0.01805003 -0.10090200
## temp       0.46639809  0.13113034  0.31452941 -0.25177799  0.07196604
## ibh       -0.38949289  0.14758023  0.14055085 -0.44414610  0.68670871
## dpg        0.01719799  0.64512337 -0.05750099 -0.35611845 -0.22959948
## ibt        0.50547402 -0.08730829  0.18874642  0.04335248 -0.10635610
## vis       -0.33852248 -0.10328284  0.53069457 -0.38551238 -0.59051425
##                   PC6
## vh        -0.113800897
## wind       0.004731925
## humidity  -0.725880286
## temp       0.084223944
## ibh       -0.146972193
## dpg        0.578824877
## ibt       -0.041320030
## vis       -0.307708782
```

Como podemos ver en la matriz de rotacion no existen atributos con las caracteristicas descritas como para poder prescindir de ellos, es decir, no vamos a aplicar el algoritmo PCA para la reduccion de dimensionalidad puesto que todos los atributos contribuyen de manera significativa a la varianza de los datos. El preprocesado que se considerara para nuestro dataset comprendera unicamente las transformaciones de BoxCox, normalizacion y escalado. Tras estas transformaciones, tendremos preparado el conjunto de datos para aprender modelos con los que predecir el valor de la variable de respuesta cuando realicemos la validacion con unos datos fuera de la propia muestra.

```r
# calculamos las transformaciones necesarias en objetoTransOzone
objetoTransOzone = preProcess(LAozone.Train[,!colnames(LAozone.Train)=="ozone"],
                              method = c("BoxCox", "center", "scale"))

# realizamos las tranformaciones calculadas con predict
Laozone.TrainProcessed = predict(objetoTransOzone, LAozone.Train)

# Mostramos las transformaciones que se han aplicado con el preProcess
# y sobre que atributos se han realizado
objetoTransOzone$method
```

```
## $BoxCox
## [1] "vh"       "humidity" "temp"     "ibh"      "vis"
##
## $center
## [1] "vh"       "wind"     "humidity" "temp"     "ibh"      "dpg"
## [7] "ibt"      "vis"
##
## $scale
## [1] "vh"       "wind"     "humidity" "temp"     "ibh"      "dpg"
## [7] "ibt"      "vis"
##
## $ignore
## character(0)
```

2.4. Seleccion de clases de funciones a usar

Este problema de regresion sera abordado unica y exclusivamente considerando modelos de regresion lineal, ya que no tiene sentido aplicar regresion logistica en problemas de regresion. La razon principal radica en que la salida de un modelo de regresion logistica es la probabilidad de obtener una determinada salida binaria (etiqueta 0 o 1) y, como nuestra variable respuesta en este problema es un conjunto de valores enteros y no una clase binaria, no podemos hacer uso de ese tipo de modelo lineal, ya que no proporcionaria una salida adecuada para nuestro problema. Para optimizar los ajustes nos valdremos de transformaciones no lineales, en concreto, de tipo cuadraticas y productos de potencias.

2.5. Regularizacion , ¿es necesario aplicarla?

Para discutir la necesidad de aplicar regularizacion o no vamos a utilizar el mismo conjunto de funciones que para el problema anterior. Primero estimaremos el valor λ a traves de una 10 fold-cross validation (cv.glmnet) del conjunto de datos con el objetivo de aplicar este valor a un modelo lineal con regularizacion "Weight Decay". La necesidad de aplicar regularizacion vendra dado por la diferencia de las desviaciones de los datos utilizando un modelo sin regularizacion ($\lambda = 0$) y el modelo con regularizacion con el mejor lambda. En caso de que la diferencia sea significativa, aplicar regularizacion ofrecera mejores resultados. A continuacion se muestran los experimentos y la interpretacion de los resultados.

```r
set.seed(14)

# Creacion de conjuntos
matrizX = model.matrix(ozone ~ ., LAozone.Train)
vectorY = LAozone.Train$ozone
```

```r
# Obtenemos el mejor lambda mediante validacion cruzada
cv.out = cv.glmnet(matrizX, vectorY, alpha = 0)
# calculamos el mejor lambda.
bestlam =cv.out$lambda.min
print(sprintf("Mejor valor lambda obtenido: %f", bestlam))
```

```
## [1] "Mejor valor lambda obtenido: 0.656166"
```

El valor del mejor $\lambda = 0.656166$, por lo tanto este valor es el que vamos a pasarle a la funcion glmnet para comprobar, a traves del ratio de la desviacion de los datos, si compensa realmente aplicar regularizacion o no.

```r
# El primer paso consiste en ejecutar glmnet con lambda = 0 para obtener la desviacion
# sin considerar regularizacion alguna
desv_GLMnet_NoReg = glmnet(matrizX, vectorY, alpha = 0, lambda = 0)
print(sprintf("Desviacion sin considerar regularizacion: %f ",
              desv_GLMnet_NoReg$dev.ratio))
```

```
## [1] "Desviacion sin considerar regularizacion: 0.677589 "
```

```r
# Comprobamos como varia el valor de desviacion si se considera regularizacion
desv_GLMnet_REG= glmnet(matrizX, vectorY, alpha = 0, lambda = bestlam)
print(sprintf("Desviacion considerando regularizacion: %f ",
              desv_GLMnet_REG$dev.ratio))
```

```
## [1] "Desviacion considerando regularizacion: 0.674986 "
```

A la vista de los valores de la desviacion considerando regularizacion y sin considerarla, concluimos que no conseguiremos mejores ajustes aplicandola, por lo tanto no sera un procedimiento considerado en la resolucion de nuestro problema y en la seleccion de los mejores modelos.

2.6. Definicion de modelos de ajuste y estimacion de parametros

Vamos a hacer uso, como en el problema de clasificacion, de la funcion regsubsets para la seleccion de los mejores atributos a la hora de aprender distintos modelos con los que ajustar nuestra funcion objetivo desconocida. Hay que destacar que regsubsets unicamente nos guia en el proceso de seleccion de los mejores atributos, sin embargo no garantiza que los resultados sean necesariamente mejores cuantos mas atributos seleccionemos. Por lo tanto, sera muy importante y decisivo para nuestro proceso de aprendizaje y seleccion del mejor modelo que el procedimiento a traves del cual calcularemos el error sea lo suficientemente representativo e interpretable como para decantarnos por un modelo u otro.

```r
# Nuestra variable de respuesta son lo valores que
# indican los niveles de ozono en la atmosfera.
regSubsetsLA = regsubsets(ozone ~ . , data=Laozone.TrainProcessed, method = "exhaustive")
summary(regSubsetsLA)
```

```
## Subset selection object
## Call: regsubsets.formula(ozone ~ ., data = Laozone.TrainProcessed,
##     method = "exhaustive")
## 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 ) "*" "*"  "*"      "*"  "*" "*" "*" "*"
```

Una vez conocemos los mejores atributos para cada uno de los subconjuntos con los que podamos aprender un modelo, el siguiente paso consiste en ajustar nuestros modelos de regresion Lineal utilizando unos determinados subconjuntos de atributos que nos lleven hacia un ajuste de cierta calidad.

2.6.1 Modelos de Regresion Lineal.

En primera instancia realizamos el preprocesado de los datos del test: normalizacion, transformaciones de simetria y escalado. Cabe destacar que para el conjunto de Test debemos realizar las mismas transformaciones que para el conjuno de Train, dado que hay que preservar y mantener las mismas condiciones a lo largo de todo el experimento.

```
# No realizamos el preprocesado sobre la variable respuesta
objetoTrans_TestLA = preProcess(LAozone.Test[,!colnames(LAozone.Test)=="ozone"],
                        method = c("BoxCox", "center", "scale"))
#objetoTrans_TestLA = preProcess(LAozone.Test,
#                         method = c("BoxCox", "center", "scale"))
LAozone.TestProcessed = predict(objetoTrans_TestLA, LAozone.Test)
```

Ahora nos disponemos a aprender los modelos lineales con los que posteriormente validaremos un conjunto de datos que no pertenecen a la muestra de entrenamiento. Aprenderemos 5 modelos lineales teniendo en cuenta los atributos que forman parte de los subconjuntos de tamaño 1 hasta 5.

```
# Nuestra variable de respuesta sera el atributo ozone.
modeloLineal1 = lm(ozone ~ temp, data = Laozone.TrainProcessed)
modeloLineal2 = lm(ozone ~ humidity + ibt, data = Laozone.TrainProcessed)
modeloLineal3 = lm(ozone ~ humidity + temp + ibh, data = Laozone.TrainProcessed)
modeloLineal4 = lm(ozone ~ humidity + temp + vh + ibt, data = Laozone.TrainProcessed)
modeloLineal5 = lm(ozone ~ humidity + temp + vh + ibt + vis, data = Laozone.TrainProcessed)
```

Creamos la funcion que evaluará los modelos aprendidos en funcion del error cuadratico medio, esto es,

$$E_{out} = \frac{1}{N}\sum_{i=0}^{N}(\hat{y}_i - y)^2$$

y evaluamos nuestro primer modelo lineal.

```
evaluacionModelo = function(modeloTrain, datosTest, varRespuesta, indiceModelo)
{
  # Calculamos la probabilidad del modelo lineal
  probTest = predict(modeloTrain, data.frame(datosTest), type="response")
  # Calculo del Eout con el modelo lineal
  Eval = mean((probTest - varRespuesta)^2)
```

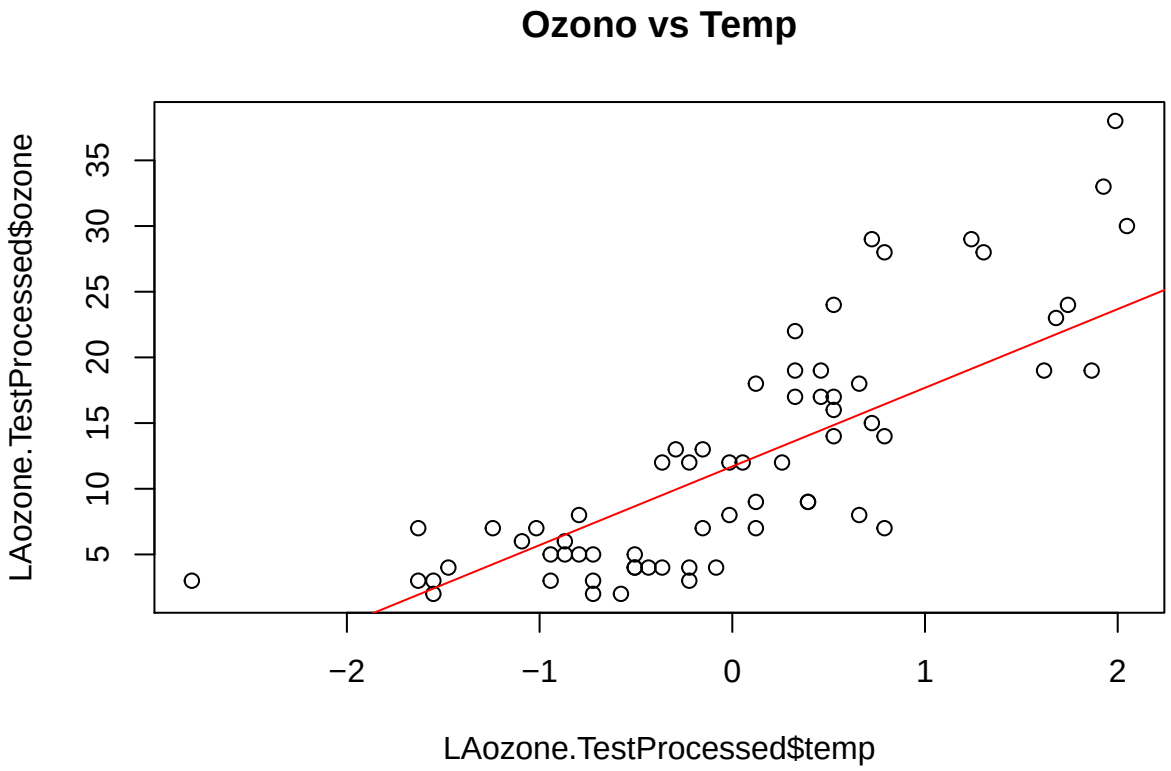
```
  print(sprintf("Evaluacion (diferencia de minimos cuadrados) del ML-%i -> %f",indiceModelo, Eval))
  cat("----------------------------------- \n")
}
```

La evaluacion del primer modelo lineal sirve para asegurarnos de que obtengamos un valor de error interpretable y que sirva para comparar los distintos modelos que hemos obtenido anteriormente.

```
# evaluacion del modelo 1
evaluacionModelo(modeloLineal1, LAozone.TestProcessed, LAozone.Test$ozone, 1)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-1 -> 28.381051"
## -----------------------------------
```

```
# Representamos el ajuste del modelo 1 frente a los datos del test
plot(LAozone.TestProcessed$temp,LAozone.TestProcessed$ozone, main = "Ozono vs Temp")
abline(modeloLineal1$coefficients, col=2 )
```



Este es el ajuste que nuestro primer modelo de regresion lineal consigue realizar sobre los datos del Test. Finalmente evaluamos los demas modelos para seleccionar los mejores, dado que seran sobre los que aplicaremos transformaciones no lineales posteriormente para intentar reducir el MSE.

```
#Procedemos a evaluar los otros 4 modelos lineales aprendidos
evaluacionModelo(modeloLineal2, LAozone.TestProcessed, LAozone.Test$ozone,2)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-2 -> 27.043615"
## -----------------------------------
```

```
evaluacionModelo(modeloLineal3, LAozone.TestProcessed, LAozone.Test$ozone,3)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-3 -> 22.608044"
## -----------------------------------
```

```
evaluacionModelo(modeloLineal4, LAozone.TestProcessed, LAozone.Test$ozone,4)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-4 -> 22.828014"
## -----------------------------------
```

```
evaluacionModelo(modeloLineal5, LAozone.TestProcessed, LAozone.Test$ozone,5)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-5 -> 22.717369"
## -----------------------------------
```

2.7 Seleccion y ajuste del modelo final

2.7.1 Transformaciones no lineales sobre los mejores modelos de Regresion Lineal

Llegados a este punto apicaremos transformaciones no lineales a algunos de los mejores modelos aprendidos con el fin de reducir los errores de minimos cuadrados y por ende ajustarnos mejor a los datos. Este es el paso previo a la seleccion del mejor modelo en funcion del ajuste que es capaz de proporcionar, dado que intentaremos realizar una estimacion del error fuera de la muestra de forma que el valor del mismo sea una medida mucho mas representativa y que sea capaz de aportar una mayor capacidad de generalizacion.

```
# modelo con transformacion no lineal 1
modeloLinealMejora1 = lm(ozone ~ humidity * ibh * temp, data = Laozone.TrainProcessed)
evaluacionModelo(modeloLinealMejora1, LAozone.TestProcessed, LAozone.Test$ozone,1)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-1 -> 18.351891"
## -----------------------------------
```

```
# modelo con transformacion no lineal 2
modeloLinealMejora2 =  lm(ozone ~  humidity * temp * ibh * ibt,
                          data = Laozone.TrainProcessed)
evaluacionModelo(modeloLinealMejora2, LAozone.TestProcessed, LAozone.Test$ozone,2)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-2 -> 17.727805"
## -----------------------------------
```

```
# modelo con transformacion no lineal 3
modeloLinealMejora3 = lm(ozone ~ humidity * temp * ibh * ibt * vis,
                         data = Laozone.TrainProcessed)
evaluacionModelo(modeloLinealMejora3, LAozone.TestProcessed, LAozone.Test$ozone,3)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-3 -> 24.464444"
## -----------------------------------
```

```
# modelo con transformacion no lineal 4
modeloLinealMejora4 = lm(ozone ~ humidity * temp * ibh * I(ibt^2) + I(vis^2),
                         data = Laozone.TrainProcessed)
evaluacionModelo(modeloLinealMejora4, LAozone.TestProcessed, LAozone.Test$ozone,4)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-4 -> 17.919499"
## -----------------------------------
```

```
# modelo con transformacion no lineal 5
modeloLinealMejora5 = lm(ozone ~ humidity * temp * ibh + I(ibt^2) + I(vis^2),
                         data = Laozone.TrainProcessed)
evaluacionModelo(modeloLinealMejora5, LAozone.TestProcessed, LAozone.Test$ozone,5)
```

```
## [1] "Evaluacion (diferencia de minimos cuadrados) del ML-5 -> 17.869944"
```

```
## ----------------------------------
```

2.8. Estimacion del error Eout del modelo con mejor ajuste.

¿Como interpretar los resultados obtenidos anteriormente? El error de minimos cuadrados es una metrica que no somos capaces de interpretar con certeza, razon por la cual procederemos a modificar la forma de calcular el error de nuestro modelo: dicho error sera calculado como la media de los errores normalizados en un intervalo. El resultado es un porcentaje de error que nos ofrece una vision mas significativa del ajuste que somos capaces de alcanzar con nuestro modelo, por lo que utilizando esta metrica seremos capaces de elegir de forma mucho mas comoda y certera el mejor de todos los modelos que explica nuestros datos, con una calidad determinada. A continuacion se muestra la funcion que calcula la media de los errores normalizados en un intervalo definido como la diferencia entre el maximo y minimo valor de nuestra variable respuesta. El calculo del error sigue la siguiente formula:

$$E_{out} = \frac{1}{N} \mathring{u} \sum_{i=1}^{N} \frac{|\hat{y}_i - y_i|}{max(y) - min(y)}$$

```
errorNormalizadoIntervalo = function(modeloTrain, datosTest,
  varRespuesta, indiceModelo, salidaPantalla = TRUE)
{
  # Calculamos la probabilidad del modelo lineal
  probTest = predict(modeloTrain, data.frame(datosTest), type="response")
  # Calculo del Eout con el modelo lineal
  numerador = abs(probTest - varRespuesta)
  denominador = max(varRespuesta) - min(varRespuesta)
  error = numerador/denominador
  error = mean(error)
  error = error*100
  if(salidaPantalla)
  {
    print(sprintf("Eout (media de errores normalizados) del ML-%i -> %f",
                  indiceModelo, error))
  }
  else
  {
    error
  }

}
```

Finalmente utilizamos la nueva metrica para explicar cual es nuestro mejor modelo

```
# Aplicacion de la metrica explicada anteriormente para explicar nuestro mejor modelo
errorNormalizadoIntervalo(modeloLinealMejora1, LAozone.TestProcessed, LAozone.Test$ozone,1,T)
```

```
## [1] "Eout (media de errores normalizados) del ML-1 -> 8.766600"
```

```
errorNormalizadoIntervalo(modeloLinealMejora2, LAozone.TestProcessed, LAozone.Test$ozone,2,T)
```

```
## [1] "Eout (media de errores normalizados) del ML-2 -> 8.985830"
```

```
errorNormalizadoIntervalo(modeloLinealMejora3, LAozone.TestProcessed, LAozone.Test$ozone,3,T)
```

```
## [1] "Eout (media de errores normalizados) del ML-3 -> 10.092354"
```

```r
errorNormalizadoIntervalo(modeloLinealMejora4, LAozone.TestProcessed, LAozone.Test$ozone,4,T)
```

```
## [1] "Eout (media de errores normalizados) del ML-4 -> 8.732578"
```

```r
errorNormalizadoIntervalo(modeloLinealMejora5, LAozone.TestProcessed, LAozone.Test$ozone,5,T)
```

```
## [1] "Eout (media de errores normalizados) del ML-5 -> 8.695157"
```

El mejor de todos los modelos es aquel que obtiene un 8.6% de error, por lo cual a priori será nuestra seleccion. En el siguiente punto corroboramos la seleccion del mismo frente a otros modelos.

2.8.1. Estimacion del Eout lo mas ajustada posible.

El objetivo de este punto, al igual que hicimos en el problema de clasificacion, sera ajustar la estimacion del Eout utilizando el mejor de los modelos anteriores para aprender con distintos conjuntos de datos train y test generados en varias ejecuciones independientes. Esta vez, para el calculo del Eout, haremos uso de la media del error normalizado en un intervalo en lugar del error de minimos cuadrados, ya que esta ultima es una medida poco interpretable para dicho error.

```r
set.seed(14)

ajusteEoutRegresion = function(dataset)
{
  # Utilizamos el 80% de las muestras del conjunto de datos
  # para crear el conjunto Train y las restantes para el Test
  porcentajeMuestrasTrain = 0.8
  train = sample(nrow(dataset), round(nrow(dataset)*porcentajeMuestrasTrain))
  dataset.Train = dataset[train,]
  dataset.Test = dataset[-train,]

  # No consideramos el atributo de historial familiar (famHist)
  # al ser una variable binaria
  objetoTrans = preProcess(dataset.Train[,!colnames(dataset.Train)=="ozone"],
                           method = c("BoxCox", "center", "scale"))
  dataset.TrainProcessed = predict(objetoTrans, dataset.Train)
  dataset.TestProcessed = predict(objetoTrans, dataset.Test)

  # Aprendemos el mejor modelo de regresion lineal
  modeloLineal = lm(ozone ~ humidity * temp * ibh + I(ibt^2) + I(vis^2), data = Laozone.TrainProcessed)

  # Calculamos la probabilidad del modelo lineal
  probTest = predict(modeloLineal, data.frame(dataset.TestProcessed), type="response")
  # Calculo del Eout con el modelo lineal
  errorNormalizadoIntervalo(modeloLineal, dataset.TestProcessed, dataset.TestProcessed$ozone, 1,F)
}

# Llamamos a la funcion para calcular el Eout medio con 100 iteraciones
errorMedio = mean(replicate(100,ajusteEoutRegresion(LAozone)))

print(sprintf("El error medio obtenido con 100 iteraciones es: %f", errorMedio))
```

```
## [1] "El error medio obtenido con 100 iteraciones es: 10.253654"
```

2.9. Conclusiones finales

Aunque el resultado medio de Eout obtenido nos permite concluir que el conjunto de datos utilizado en este problema no es linealmente separable, consideramos como un buen ajuste un valor del 10% de error fuera de la muestra que proporciona nuestro modelo. A continuacion explicaremos como hemos llegado a obtener esta conclusion, basandonos principalmente en los resultados de los experimentos realizados y sus interpretaciones.

Tras aplicar los mismos procesos sobre los datos que en el problema de clasificacion (generacion de particiones Train y Test, preprocesado, seleccion de clases de funciones, regularizacion y seleccion de mejores atributos) nos dispusimos a generar varios modelos de regresion lineal (y solo de regresioin lineal, ya que no tiene sentido aplicar regresion logistica por las razones expuestas anteriormente) basandonos en los subconjuntos de los atributos que mejor explicaban nuestros datos. Tras ello evaluamos dichos modelos con el error de minimos cuadrados, seleccionamos un subconjunto de los que ofrecian mejores resultados e intentamos reducir el error fuera de la muestra mediante transformaciones no lineales de tipo cuadratica y producto de potencias.

Los resultados obtenidos con el error de minimos cuadrados no ofrecian una medida clara e interpretable del error fuera de la muestra, por lo que decidimos cambiar la metrica de dicho error por la media de los errores normalizados en un intervalo. Esta nueva metrica, al estar normalizada en el rango definido por el maximo y el minimo de la variable respuesta nos ofrece un valor en el rango [0,1] del error cometido con el modelo lineal aprendido para separar el conjunto de datos Test, medidas extrapolables a un porcentaje de error,la cual es lo suficientemente representativa como para poder comparar los modelos lineales entre si y decidir cual de estos es el mejor. Usando esta medida evaluamos nuevamente nuestros modelos y, a la vista de los resultados, entre los 5 modelos distintos generados mediante transformaciones no lineales, el que mejor ajusta el error es aquel modelo que utiliza 5 de los mejores atributos con transformaciones cuadraticas y productos de potencias de los mismos, arrojando un error fuera de la muestra de 8.69%, lo cual pese a que el dataset no es linealmente separable, se puede considerar un buen ajuste.

Por ultimo, para ofrecer una medida mas representativa y con mayor poder de generalizacion hemos decidido repetir el experimento en su totalidad 100 veces, generando en cada iteracion un nuevo par de conjuntos Train y Test. Usaremos este nuevo conjunto de train para aprener un nuevo modelo lineal conteniendo los mismos atributos y transformaciones no lineales que el mejor de los modelos aprendidos en el experimento 0 y, posteriormente, validamos el mismo con el conjunto de test obteniendo un porcentaje de error fuera de la muestra. Tras repetir 100 veces el experimento y hacer la media de los sucesivos Eout obtenidos, concluimos que el error fuera de la muestra con el modelo lineal aprendido es de 10.25%, medida que avala nuestra postura acerca de que nuestro mejor modelo (de regresion lineal con 5 de los mejores atributos con transformaciones cuadraticas y productos de potencias) es capaz de realizar un ajuste de cierta calidad frente a un conjunto de datos que no es linealmente separable.