Práctica 3 - Aprendizaje Automático

David López Pretel y Néstor Rodriguez Vico

27 de mayo de 2017

Problema de clasificacion

Para este problema hemos elegido la base de datos *email spam*.

1. Comprender al problema a resolver.

Esta base de consta de 4061 ejemplos, con 57 características cada ejemplo (sin contar la clase). Dentro de la base de datos, la última columna representa la clase, siendo 1 si el email es considerado como spam y siendo 0 en caso contrario. Las 57 características son valores reales que miden información presentes en los emails, como puede ser el número de ocurrencias de una palabra en el email.

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

El conjunto de train y test nos viene dados por la propia base de datos. A la vez que se descarga el conjunto de datos obtenemos un fichero formado por 0's y 1's, el cual nos indica si un dato va a train o va a test.

```
trainTest = as.vector(read.table("./datos/spam.traintest"))
cantidadUnos = sum(trainTest)
cat("Numero de unos:", cantidadUnos, "\n")
```

```
## Numero de unos: 1536
```

Como podemos ver, hay 1536 unos y 3065 ceros, por lo tanto vamos a considerar como conjunto de train los datos que tengan un 0 en el fichero *spam.traintest*.

```
datos = datos = read.csv("./datos/spam.data", header = F, sep = " ")
indicesTrain = which(trainTest == 0)
train = (as.matrix(datos))[indicesTrain,]
test = (as.matrix(datos))[-indicesTrain,]
cat("Tamaño train:", dim(train), "\n")
```

```
## Tamaño train: 3065 58
```

```
cat("Tamaño test:", dim(test), "\n")
```

```
## Tamaño test: 1536 58
```

No vamos a aplicar validación porque los datos vienen ya separados en train y test, así que no será necesario generar nuevos conjuntos ni realizar ejecuciones para distintos conjuntos de datos.

3. Preprocesado los datos: Falta de datos, categorización, normalización, reducción de dimensionalidad, etc.

En cuanto a preprocesado de datos vamos a aplicar lo explicado en el fichero *paraTrabajo3.pdf*.

```
# Guardamos las clases de los conjuntos
clasesTrain = train[, dim(train)[2]]
clasesTest = test[, dim(test)[2]]
# Eliminamos las clases de los conjuntos
train = train[, -dim(train)[2]]
```

```
test = test[, -dim(test)[2]]

library("caret")
```

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

```
## Loading required package: ggplot2
```

```
# Se usa el método YeoJohnson en vez de BoxCox ya que el segundo no es adecuado
# cuando las características pueden tomar valores negativos o valores iguales
# a cero, y esto ocurre en nuestro dataset.
ObjetoTrans = preProcess(train, method = c("YeoJohnson", "center", "scale", "pca"),
                         thresh=0.95)
ObjetoTrans_sinPCA = preProcess(train, method = c("YeoJohnson", "center", "scale"),
                                thresh=0.95)

trainPCA = predict(ObjetoTrans, train)
cat("Nuevo tamaño de train (con PCA):", dim(trainPCA), "\n")
```

```
## Nuevo tamaño de train (con PCA): 3065 46
```

```
testPCA = predict(ObjetoTrans, test)
cat("Nuevo tamaño de test (con PCA):", dim(testPCA), "\n")
```

```
## Nuevo tamaño de test (con PCA): 1536 46
```

```
trainSinPCA = predict(ObjetoTrans_sinPCA, train)
cat("Nuevo tamaño de train (sin PCA):", dim(trainSinPCA), "\n")
```

```
## Nuevo tamaño de train (sin PCA): 3065 57
```

```
testSinPCA = predict(ObjetoTrans_sinPCA, test)
cat("Nuevo tamaño de test (sin PCA):", dim(testSinPCA), "\n")
```

```
## Nuevo tamaño de test (sin PCA): 1536 57
```

4. Selección de clases de funciones a usar.

Las funciones que vamos a usar son las funciones lineales.

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

La regularización, por lo general, siempre suele ser buena, así que la vamos a aplicar. Más adelante veremos si merece la pena o no.

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

Para ver que modelos vamos a usar, vamos a seguir una estrategia en la cual vamos a tratar de distintas maneras los datos y vamos a aprender distintos modelos en base a ellos. Por eso tenemos 4 conjuntos de datos distintos: *trainPCA*, *testPCA*, *trainSinPCA* y *testSinPCA*. En los dos primeros se ha aplicado la técina PCA (para obtener las características principales) y en los dos segundos no. A continuación, vamos a usar la función *regsubsets* para ver cuáles son las características más importantes y ajustar distintos modelos en función de las mismas. Finalmente, vamos a intentar aplicar transformaciones sobre los datos para intentar mejorar los resultados obtenidos.

Veamos cuales son las características más importantes de los conjuntos con PCA aplicado:

```
library("leaps")

trainPCA = as.data.frame(cbind(clasesTrain, trainPCA))
testPCA = as.data.frame(cbind(clasesTest, testPCA))
trainSinPCA = as.data.frame(cbind(clasesTrain, trainSinPCA))
testSinPCA = as.data.frame(cbind(clasesTest, testSinPCA))

subsetPCA = regsubsets(trainPCA$clasesTrain ~ . , data = trainPCA,
                       nvmax = dim(trainPCA)[2], method = "forward")
summaryPCA = summary(subsetPCA)

subsetSinPCA = regsubsets(trainSinPCA$clasesTrain ~ . , data = trainSinPCA,
                          nvmax = dim(testSinPCA)[2], method = "forward")
summarySinPCA = summary(subsetSinPCA)
```

Las características que nos interesan aparecen en las siguiente tablas. Primero para el caso de usar PCA y a continuación si no lo usamos:

```
head(summaryPCA$outmat)
```

```
##          PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12 PC13 PC14 PC15
## 1  ( 1 ) "*" " " " " " " " " " " " " " " " " " "  " "  " "  " "  " "  " "
## 2  ( 1 ) "*" " " "*" " " " " " " " " " " " " " " " "  " "  " "  " "  " "  " "
## 3  ( 1 ) "*" " " "*" "*" " " " " " " " " " " " " " " " "  " "  " "  " "  " "  " "
## 4  ( 1 ) "*" " " "*" "*" " " " " " " " " " " " " " " " "  "*"  " "  " "  " "  " "
## 5  ( 1 ) "*" " " "*" "*" " " "*" " " " " " " " " " " " "  "*"  " "  " "  " "  " "
## 6  ( 1 ) "*" " " "*" "*" " " "*" "*" " " " " " " " " " "  "*"  " "  " "  " "  " "
##          PC16 PC17 PC18 PC19 PC20 PC21 PC22 PC23 PC24 PC25 PC26 PC27 PC28
## 1  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 2  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 3  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 4  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 5  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 6  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
##          PC29 PC30 PC31 PC32 PC33 PC34 PC35 PC36 PC37 PC38 PC39 PC40 PC41
## 1  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 2  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 3  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 4  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 5  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
## 6  ( 1 ) " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "  " "
##          PC42 PC43 PC44 PC45 PC46
## 1  ( 1 ) " "  " "  " "  " "  " "
## 2  ( 1 ) " "  " "  " "  " "  " "
## 3  ( 1 ) " "  " "  " "  " "  " "
## 4  ( 1 ) " "  " "  " "  " "  " "
## 5  ( 1 ) " "  " "  " "  " "  " "
## 6  ( 1 ) " "  " "  " "  " "  " "
```

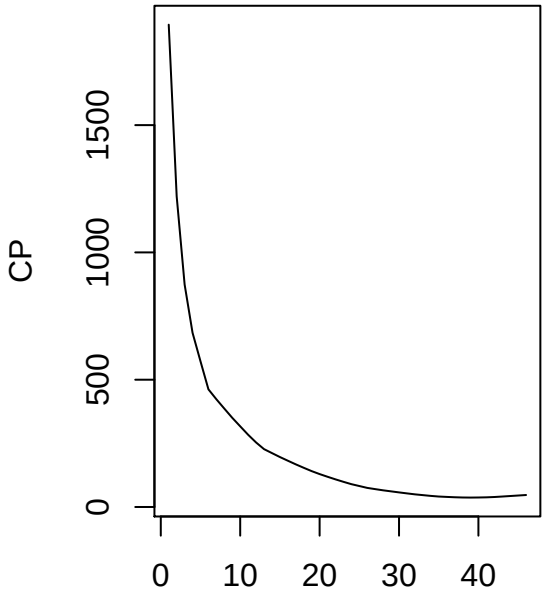
```
head(summarySinPCA$outmat)
```

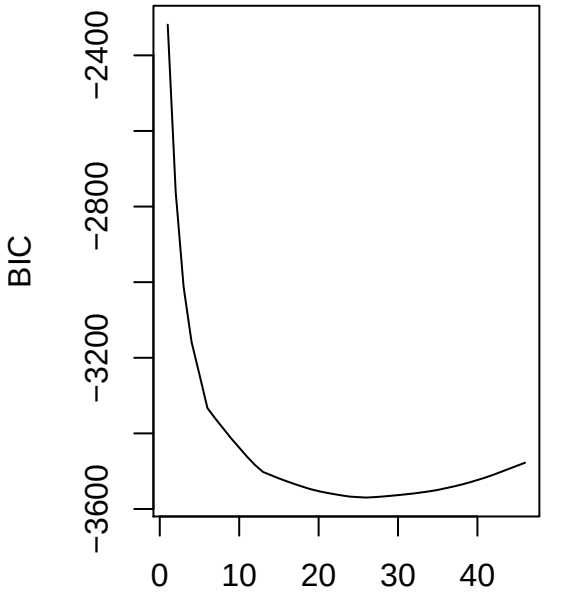
```
##          V1  V2  V3  V4  V5  V6  V7  V8  V9  V10 V11 V12 V13 V14 V15 V16
## 1  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 3  ( 1 ) " " " " " " " " " " " " " " " " "*" " " " " " " " " " " " " " " " "
```

```
## 4  ( 1 ) " " " " " " " " " " " " " " "*" " " " " " " " " " " " " " " " " " " " " " "
## 5  ( 1 ) " " " " " " " " " " " " " " "*" " " " " " " " " " " " " " " " " " " " " " "
## 6  ( 1 ) " " " " " " " " " " " " " " "*" " " " " " " " " " " " " " " " " " " " " " "
##          V17 V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32
## 1  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 3  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 4  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 5  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 6  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
##          V33 V34 V35 V36 V37 V38 V39 V40 V41 V42 V43 V44 V45 V46 V47 V48
## 1  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 3  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 4  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 5  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 6  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "*" " " " "
##          V49 V50 V51 V52 V53 V54 V55 V56 V57
## 1  ( 1 ) " " " " " " " " "*" " " " " " " " "
## 2  ( 1 ) " " " " " " " "*" "*" " " " " " " "
## 3  ( 1 ) " " " " " " " "*" "*" " " " " " " "
## 4  ( 1 ) " " " " " " " "*" "*" " " " " " " "
## 5  ( 1 ) " " " " " " " "*" "*" " " " " "*" " "
## 6  ( 1 ) " " " " " " " "*" "*" " " " " "*" " "
```

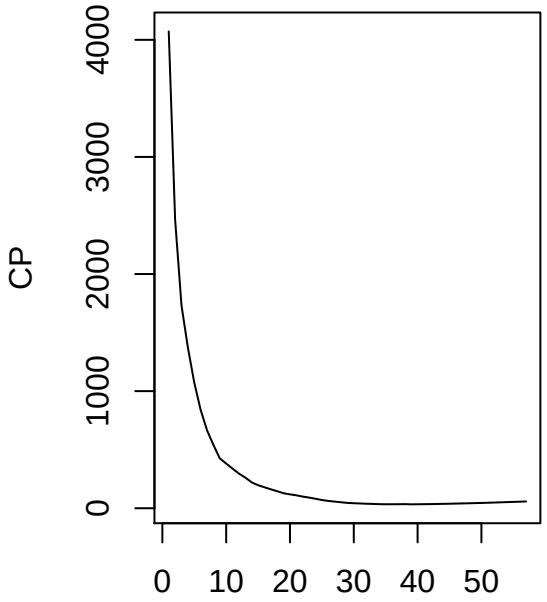
Una vez sabemos cuales son las mejores características en función del número de las mismas que deseemos usar, vamos a ver cual es la cantidad óptima de características que debemos usar. Para ello vamos a pintar un par de gráficas (en función de los valores CP y BIC) y vamos a calcular donde se alcanza el valor mínimo:

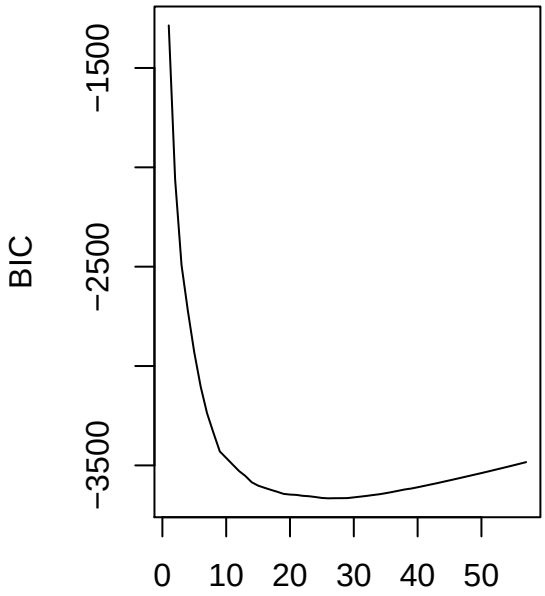
```
par(mfrow=c(1,2))
plot(summaryPCA$cp, xlab = "Número de variables (con PCA).", ylab = "CP", type = "l")
plot(summaryPCA$bic, xlab = "Número de variables (con PCA).", ylab = "BIC", type = "l")
```





```r
plot(summarySinPCA$cp, xlab = "Número de variables (sin PCA).", ylab = "CP", type = "l")
plot(summarySinPCA$bic, xlab = "Número de variables (sin PCA).", ylab = "BIC", type = "l")
```






```r
cat("Mejor número de características - CP (con PCA):", which.min(summaryPCA$cp), "\n")
```

```
## Mejor número de características - CP (con PCA): 39
```

```r
cat("Mejor número de características - BIC (con PCA):", which.min(summaryPCA$bic), "\n")
```

```
## Mejor número de características - BIC (con PCA): 26
```

```r
cat("Mejor número de características - CP (sin PCA):", which.min(summarySinPCA$cp), "\n")
```

```
## Mejor número de características - CP (sin PCA): 39
```

```r
cat("Mejor número de características - BIC (sin PCA):", which.min(summarySinPCA$bic), "\n")
```

```
## Mejor número de características - BIC (sin PCA): 26
```

Podemos ver que los mejores resultados se obtiene usando 39 características según CP y 26 según BIC, se haya usado PCA o no. Las características son estas:

```r
as.vector(which(summaryPCA$outmat[29,] == "*"))
```

```
##  [1]  1  2  3  4  5  6  7 10 11 12 13 16 17 18 19 22 23 24 26 27 28 31 32
## [24] 33 34 35 37 39 43
```

```r
as.vector(which(summaryPCA$outmat[36,] == "*"))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 15 16 17 18 19 22 23 24 26 27
## [24] 28 29 31 32 33 34 35 37 38 39 41 43 45
```

```r
as.vector(which(summarySinPCA$outmat[29,] == "*"))
```

```
##  [1]  1  5  7  8 12 13 16 17 19 21 22 24 25 27 32 33 36 37 42 44 45 46 48
## [24] 52 53 54 55 56 57
```

```r
as.vector(which(summarySinPCA$outmat[36,] == "*"))
```

```
##  [1]  1  2  5  7  8  9 11 12 13 16 17 19 21 22 23 24 25 27 31 32 33 36 37
## [24] 39 42 43 44 45 46 48 52 53 54 55 56 57
```

Si quisiesemos usar una única característica sería:

```r
as.vector(which(summaryPCA$outmat[1,] == "*"))
```

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

```r
as.vector(which(summaryPCA$outmat[1,] == "*"))
```

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

```r
as.vector(which(summarySinPCA$outmat[1,] == "*"))
```

```
## [1] 53
```

```r
as.vector(which(summarySinPCA$outmat[1,] == "*"))
```

```
## [1] 53
```

Como podemos ver, aunque se use el mísmo número de características, no son las mismas. Una vez sabemos que características usar, podemos ir construyendo nuestros primeros modelos:

```r
datosTrainModeloPCA_1 = trainPCA[, c(1, as.vector(which(summaryPCA$outmat[1,]
                                                        == "*")) + 1)]
datosTrainModeloSinPCA_1 = trainSinPCA[, c(1, as.vector(which(summarySinPCA$outmat[1,]
                                                              == "*")) + 1)]

datosTrainModeloPCA_29 = trainPCA[, c(1, as.vector(which(summaryPCA$outmat[29,]
                                                        == "*")) + 1)]
datosTrainModeloSinPCA_29 = trainSinPCA[, c(1, as.vector(which(summarySinPCA$outmat[29,]
                                                               == "*")) + 1)]

datosTrainModeloPCA_36 = trainPCA[, c(1, as.vector(which(summaryPCA$outmat[36,]
                                                        == "*")) + 1)]
datosTrainModeloSinPCA_36 = trainSinPCA[, c(1, as.vector(which(summarySinPCA$outmat[36,]
                                                               == "*")) + 1)]

datosTrainModeloPCA_57 = trainPCA
datosTrainModeloSinPCA_57 = trainSinPCA

# En el caso de los conjuntos de test, no vamos a guardar la clase en el dataset
datosTestModeloPCA_1 = as.data.frame(testPCA[, as.vector(
  which(summaryPCA$outmat[1,] == "*")) + 1])
datosTestModeloSinPCA_1 = as.data.frame(testSinPCA[, as.vector(
  which(summarySinPCA$outmat[1,] == "*")) + 1])

# Al crear un data.frame con una única columna se pierde el nombre de la columna
names1 = names(datosTrainModeloPCA_1)[-1]
names2 = names(datosTrainModeloSinPCA_1)[-1]
names(datosTestModeloPCA_1) = names1
names(datosTestModeloSinPCA_1) = names2

datosTestModeloPCA_29 = as.data.frame(testPCA[, as.vector(
  which(summaryPCA$outmat[29,] == "*")) + 1])
```

```r
datosTestModeloSinPCA_29 = as.data.frame(testSinPCA[, as.vector(
  which(summarySinPCA$outmat[29,] == "*")) + 1])

datosTestModeloPCA_36 = as.data.frame(testPCA[, as.vector(
  which(summaryPCA$outmat[36,] == "*")) + 1])
datosTestModeloSinPCA_36 = as.data.frame(testSinPCA[, as.vector(
  which(summarySinPCA$outmat[36,] == "*")) + 1])

datosTestModeloPCA_57 = testPCA[,-1]
datosTestModeloSinPCA_57 = testSinPCA[,-1]

# Si en family no pones nada es lineal
# Si pones family = binomial(logit) es regresion logistica

#Una vez tenemos los conjuntos de datos, aprendemos los modelos
glmPCA_1 = glm(datosTrainModeloPCA_1$clasesTrain ~ .,
                 data = datosTrainModeloPCA_1, family = binomial(logit))
glmSinPCA_1 = glm(datosTrainModeloSinPCA_1$clasesTrain ~ .,
                    data = datosTrainModeloSinPCA_1, family = binomial(logit))

glmPCA_29 = glm(datosTrainModeloPCA_29$clasesTrain ~ .,
                  data = datosTrainModeloPCA_29, family = binomial(logit))
glmSinPCA_29 = glm(datosTrainModeloSinPCA_29$clasesTrain ~ .,
                     data = datosTrainModeloSinPCA_29, family = binomial(logit))

glmPCA_36 = glm(datosTrainModeloPCA_36$clasesTrain ~ .,
                  data = datosTrainModeloPCA_36, family = binomial(logit))
glmSinPCA_36 = glm(datosTrainModeloSinPCA_36$clasesTrain ~ .,
                     data = datosTrainModeloSinPCA_36, family = binomial(logit))

glmPCA_57 = glm(datosTrainModeloPCA_57$clasesTrain ~ .,
                  data = datosTrainModeloPCA_57, family = binomial(logit))
glmSinPCA_57 = glm(datosTrainModeloSinPCA_57$clasesTrain ~ .,
                     data = datosTrainModeloSinPCA_57, family = binomial(logit))
```

Primero vamos a ver si hace falta aplicar regularización. Para ello vamos a comparar las desviaciones en el caso de no usar regularización (usando lambda = 0) y en el caso de usarla (usando como lambda el mejor lambda estimado).

```r
library("glmnet")
```

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

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

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

```r
a = cv.glmnet(x = as.matrix(datosTrainModeloPCA_29[,-1]),
              y = as.matrix(datosTrainModeloPCA_29[,1]), alpha = 0)
mejorLambda = a$lambda.min
# Sin regularización
glmnet(x = as.matrix(datosTrainModeloPCA_29[,-1]),
       y = as.matrix(datosTrainModeloPCA_29[,1]), alpha = 0,
       lambda = 0, standardize = FALSE)
```

```
##
```

```
## Call:  glmnet(x = as.matrix(datosTrainModeloPCA_29[, -1]), y = as.matrix(datosTrainModeloPCA_29[,
##
##      Df   %Dev Lambda
## [1,] 29 0.7111      0
```

```
# Con regularización
glmnet(x = as.matrix(datosTrainModeloPCA_29[,-1]),
       y = as.matrix(datosTrainModeloPCA_29[,1]), alpha = 0,
       lambda = mejorLambda, standardize = FALSE)
```

```
##
## Call:  glmnet(x = as.matrix(datosTrainModeloPCA_29[, -1]), y = as.matrix(datosTrainModeloPCA_29[,
##
##      Df   %Dev  Lambda
## [1,] 29 0.7104 0.03922
```

Como podemos ver, obtenemos unos valores de %Dev similares, lo cual nos lleva a pensar que no es necesario aplicar regularización, aún así vamos a aplicarla para ver si obtenemos unos mejores resultados o se confirma que no es necesario hacerla. Los modelos que vamos a ajustar son con regularización (weight decay, ya que ponemos el parámetro alpha a 0).

```
a = cv.glmnet(x = as.matrix(datosTrainModeloPCA_29[,-1]),
              y = as.matrix(datosTrainModeloPCA_29[,1]), alpha = 0)
mejorLambda = a$lambda.min
# Si usamos alpha = 0 hace WD
glmnetPCA_29 = glmnet(x = as.matrix(datosTrainModeloPCA_29[,-1]),
                      y = as.matrix(datosTrainModeloPCA_29[,1]), alpha = 0,
                      lambda = mejorLambda, standardize = FALSE)

a = cv.glmnet(x = as.matrix(datosTrainModeloSinPCA_29[,-1]),
              y = as.matrix(datosTrainModeloSinPCA_29[,1]), alpha = 0)
mejorLambda = a$lambda.min
glmnetSinPCA_29 = glmnet(x = as.matrix(datosTrainModeloSinPCA_29[,-1]),
                         y = as.matrix(datosTrainModeloSinPCA_29[,1]), alpha = 0,
                         lambda = mejorLambda, standardize = FALSE)

a = cv.glmnet(x = as.matrix(datosTrainModeloPCA_36[,-1]),
              y = as.matrix(datosTrainModeloPCA_36[,1]), alpha = 0)
mejorLambda = a$lambda.min
glmnetPCA_36 = glmnet(x = as.matrix(datosTrainModeloPCA_36[,-1]),
                      y = as.matrix(datosTrainModeloPCA_36[,1]), alpha = 0,
                      lambda = mejorLambda, standardize = FALSE)

a = cv.glmnet(x = as.matrix(datosTrainModeloSinPCA_36[,-1]),
              y = as.matrix(datosTrainModeloSinPCA_36[,1]), alpha = 0)
mejorLambda = a$lambda.min
glmnetSinPCA_36 = glmnet(x = as.matrix(datosTrainModeloSinPCA_36[,-1]),
                         y = as.matrix(datosTrainModeloSinPCA_36[,1]), alpha = 0,
                         lambda = mejorLambda, standardize = FALSE)

a = cv.glmnet(x = as.matrix(datosTrainModeloPCA_57[,-1]),
              y = as.matrix(datosTrainModeloPCA_57[,1]), alpha = 0)
mejorLambda = a$lambda.min
glmnetPCA_57 = glmnet(x = as.matrix(datosTrainModeloPCA_57[,-1]),
                      y = as.matrix(datosTrainModeloPCA_57[,1]), alpha = 0,
                      lambda = mejorLambda, standardize = FALSE)
```

```
a = cv.glmnet(x = as.matrix(datosTrainModeloSinPCA_57[,-1]),
              y = as.matrix(datosTrainModeloSinPCA_57[,1]), alpha = 0)
mejorLambda = a$lambda.min
glmnetSinPCA_57 = glmnet(x = as.matrix(datosTrainModeloSinPCA_57[,-1]),
                         y = as.matrix(datosTrainModeloSinPCA_57[,1]), alpha = 0,
                         lambda = mejorLambda, standardize = FALSE)
```

Finalmente, en clase hemos visto que se pueden hacer transformaciones sobre los datos, vamos a intentarlo:

```
datos = as.data.frame(datosTrainModeloPCA_29[,-1])
# Recuperamos los nombres del data.frame
names = names(datosTrainModeloPCA_29)[-1]
names(datos) = names

#Intentamos aplicar una transformación polinomial
#poly(as.matrix(datos), 2)
```

Para poder generar el PDF he tenido que comentar la línea que intenta hacer la transformación, ya que da error y no se puede generar. El error que da es *Error: no se puede ubicar un vector de tamaño 767004.2 Gb*. El error se da porque son muchísimas características y por lo tanto el polinomio es enorme.

7. Selección y ajuste modelo final.

A la hora de seleccionar los modelos vamos a elegir como el mejor el que nos de un menor error para el conjunto de test. También vamos a ver las curvas ROC de cada modelo y la matriz de confusión.

Para ver si nuestros modelos sobreaprenden podemos ver los errores que se obtienen clasificando el mismo conjunto con el que hemos aprendido (E_{in}) y ver si dichos errores difieren mucho de los errores fuera de la muestra (E_{test}).

```
dfAux = as.data.frame(datosTrainModeloPCA_1[,-1])
dfAux2 = as.data.frame(datosTrainModeloSinPCA_1[,-1])
names1 = names(datosTrainModeloPCA_1)[-1]
names2 = names(datosTrainModeloSinPCA_1)[-1]
names(dfAux) = names1
names(dfAux2) = names2

probTrainModeloPCA_1 = predict(glmPCA_1, dfAux, type="response")
predTrainModeloPCA_1 = rep(0, length(probTrainModeloPCA_1))
predTrainModeloPCA_1[probTrainModeloPCA_1 >=0.5] = 1
errorTrainModeloPCA_1 = mean(predTrainModeloPCA_1 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmPCA_1:",
    errorTrainModeloPCA_1, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmPCA_1: 0.1236542
```

```
probTrainModeloSinPCA_1 = predict(glmSinPCA_1, dfAux2, type="response")
predTrainModeloSinPCA_1 = rep(0, length(probTrainModeloSinPCA_1))
predTrainModeloSinPCA_1[probTrainModeloSinPCA_1 >=0.5] = 1
errorTrainModeloSinPCA_1 = mean(predTrainModeloSinPCA_1 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_1:",
    errorTrainModeloSinPCA_1, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_1: 0.2071778
```

```
probTrainModeloPCA_29 = predict(glmPCA_29, as.data.frame(datosTrainModeloPCA_29[,-1]),
                                type="response")
predTrainModeloPCA_29 = rep(0, length(probTrainModeloPCA_29))
predTrainModeloPCA_29[probTrainModeloPCA_29 >=0.5] = 1
errorTrainModeloPCA_29 = mean(predTrainModeloPCA_29 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmPCA_29:",
    errorTrainModeloPCA_29, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmPCA_29: 0.05611746
```

```
probTrainModeloSinPCA_29 = predict(glmSinPCA_29, as.data.frame(datosTrainModeloSinPCA_29[,-1]),
                                   type="response")
predTrainModeloSinPCA_29 = rep(0, length(probTrainModeloSinPCA_29))
predTrainModeloSinPCA_29[probTrainModeloSinPCA_29 >=0.5] = 1
errorTrainModeloSinPCA_29 = mean(predTrainModeloSinPCA_29 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_29:",
    errorTrainModeloSinPCA_29, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_29: 0.05220228
```

```
probTrainModeloPCA_36 = predict(glmPCA_36, as.data.frame(datosTrainModeloPCA_36[,-1]),
                                type="response")
predTrainModeloPCA_36 = rep(0, length(probTrainModeloPCA_36))
predTrainModeloPCA_36[probTrainModeloPCA_36 >=0.5] = 1
errorTrainModeloPCA_36 = mean(predTrainModeloPCA_36 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmPCA_36:",
    errorTrainModeloPCA_36, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmPCA_36: 0.05546493
```

```
probTrainModeloSinPCA_36 = predict(glmSinPCA_36, as.data.frame(datosTrainModeloSinPCA_36[,-1]),
                                   type="response")
predTrainModeloSinPCA_36 = rep(0, length(probTrainModeloSinPCA_36))
predTrainModeloSinPCA_36[probTrainModeloSinPCA_36 >=0.5] = 1
errorTrainModeloSinPCA_36 = mean(predTrainModeloSinPCA_36 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_36:",
    errorTrainModeloSinPCA_36, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_36: 0.05187602
```

```
probTrainModeloPCA_57 = predict(glmPCA_57, as.data.frame(datosTrainModeloPCA_57[,-1]),
                                type="response")
predTrainModeloPCA_57 = rep(0, length(probTrainModeloPCA_57))
predTrainModeloPCA_57[probTrainModeloPCA_57 >=0.5] = 1
errorTrainModeloPCA_57 = mean(predTrainModeloPCA_57 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmPCA_57:",
    errorTrainModeloPCA_57, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmPCA_57: 0.05709625
```

```
probTrainModeloSinPCA_57 = predict(glmSinPCA_57, as.data.frame(datosTrainModeloSinPCA_57[,-1]),
                                   type="response")
predTrainModeloSinPCA_57 = rep(0, length(probTrainModeloSinPCA_57))
predTrainModeloSinPCA_57[probTrainModeloSinPCA_57 >=0.5] = 1
errorTrainModeloSinPCA_57 = mean(predTrainModeloSinPCA_57 != clasesTrain)
cat("Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_57:",
    errorTrainModeloSinPCA_57, "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmSinPCA_57: 0.05187602
```

Una vez obtenemos los errores dentro de la muestra, veamos lo que sucede fuera.

```
probTestModeloPCA_1 = predict(glmPCA_1, datosTestModeloPCA_1, type="response")
predTestModeloPCA_1 = rep(0, length(probTestModeloPCA_1))
predTestModeloPCA_1[probTestModeloPCA_1 >=0.5] = 1
errorModeloPCA_1 = mean(predTestModeloPCA_1 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmPCA_1:",
    errorModeloPCA_1, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmPCA_1: 0.1197917
```

```
probTestModeloSinPCA_1 = predict(glmSinPCA_1, datosTestModeloSinPCA_1, type="response")
predTestModeloSinPCA_1 = rep(0, length(probTestModeloSinPCA_1))
predTestModeloSinPCA_1[probTestModeloSinPCA_1 >=0.5] = 1
errorModeloSinPCA_1 = mean(predTestModeloSinPCA_1 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmSinPCA_1:",
    errorModeloSinPCA_1, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmSinPCA_1: 0.2057292
```

```
probTestModeloPCA_29 = predict(glmPCA_29, datosTestModeloPCA_29, type="response")
predTestModeloPCA_29 = rep(0, length(probTestModeloPCA_29))
predTestModeloPCA_29[probTestModeloPCA_29 >=0.5] = 1
errorModeloPCA_29 = mean(predTestModeloPCA_29 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmPCA_29:",
    errorModeloPCA_29, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmPCA_29: 0.06119792
```

```
probTestModeloSinPCA_29 = predict(glmSinPCA_29, datosTestModeloSinPCA_29, type="response")
predTestModeloSinPCA_29 = rep(0, length(probTestModeloSinPCA_29))
predTestModeloSinPCA_29[probTestModeloSinPCA_29 >=0.5] = 1
errorModeloSinPCA_29 = mean(predTestModeloSinPCA_29 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmSinPCA_29:",
    errorModeloSinPCA_29, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmSinPCA_29: 0.06575521
```

```
probTestModeloPCA_36 = predict(glmPCA_36, datosTestModeloPCA_36, type="response")
predTestModeloPCA_36 = rep(0, length(probTestModeloPCA_36))
predTestModeloPCA_36[probTestModeloPCA_36 >=0.5] = 1
errorModeloPCA_36 = mean(predTestModeloPCA_36 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmPCA_36:",
    errorModeloPCA_36, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmPCA_36: 0.0625
```

```
probTestModeloSinPCA_36 = predict(glmSinPCA_36, datosTestModeloSinPCA_36, type="response")
predTestModeloSinPCA_36 = rep(0, length(probTestModeloSinPCA_36))
predTestModeloSinPCA_36[probTestModeloSinPCA_36 >=0.5] = 1
errorModeloSinPCA_36 = mean(predTestModeloSinPCA_36 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmSinPCA_36:",
    errorModeloSinPCA_36, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmSinPCA_36: 0.06835938
```

```
probTestModeloPCA_57 = predict(glmPCA_57, datosTestModeloPCA_57, type="response")
predTestModeloPCA_57 = rep(0, length(probTestModeloPCA_57))
predTestModeloPCA_57[probTestModeloPCA_57 >=0.5] = 1
errorModeloPCA_57 = mean(predTestModeloPCA_57 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmPCA_57:",
    errorModeloPCA_57, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmPCA_57: 0.06575521
```

```
probTestModeloSinPCA_57 = predict(glmSinPCA_57, datosTestModeloSinPCA_57, type="response")
predTestModeloSinPCA_57 = rep(0, length(probTestModeloSinPCA_57))
predTestModeloSinPCA_57[probTestModeloSinPCA_57 >=0.5] = 1
errorModeloSinPCA_57 = mean(predTestModeloSinPCA_57 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmSinPCA_57:",
    errorModeloSinPCA_57, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmSinPCA_57: 0.06445312
```

Como podemos ver, obtenemos unos errores similares tanto dentro de la muestra como fuera de la misma, así que podemos suponer que nuestro modelo no ha sobreaprendido.

A continuación, vamos a calcular los errores de los modelos usando regularizacion:

```
probRegTestModeloPCA_29 = predict(glmnetPCA_29, s = mejorLambda,
                                  newx = as.matrix(datosTestModeloPCA_29))
predRegTestModeloPCA_29 = rep(0, length(probRegTestModeloPCA_29))
predRegTestModeloPCA_29[probRegTestModeloPCA_29 >=0.5] = 1
errorRegTestModeloPCA_29 = mean(predRegTestModeloPCA_29 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmnetPCA_29:",
    errorRegTestModeloPCA_29, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmnetPCA_29: 0.07291667
```

```
probRegTestModeloSinPCA_29 = predict(glmnetSinPCA_29, s = mejorLambda,
                                     newx = as.matrix(datosTestModeloSinPCA_29))
predRegTestModeloSinPCA_29 = rep(0, length(probRegTestModeloSinPCA_29))
predRegTestModeloSinPCA_29[probRegTestModeloSinPCA_29 >=0.5] = 1
errorRegTestModeloSinPCA_29 = mean(predRegTestModeloSinPCA_29 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmnetSinPCA_29:",
    errorRegTestModeloSinPCA_29, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmnetSinPCA_29: 0.06966146
```

```
probRegTestModeloPCA_36 = predict(glmnetPCA_36, s = mejorLambda,
                                  newx = as.matrix(datosTestModeloPCA_36))
predRegTestModeloPCA_36 = rep(0, length(probRegTestModeloPCA_36))
predRegTestModeloPCA_36[probRegTestModeloPCA_36 >=0.5] = 1
errorRegTestModeloPCA_36 = mean(predRegTestModeloPCA_36 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmnetPCA_36:",
    errorRegTestModeloPCA_36, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmnetPCA_36: 0.07877604
```

```
probRegTestModeloSinPCA_36 = predict(glmnetSinPCA_36, s = mejorLambda,
                                     newx = as.matrix(datosTestModeloSinPCA_36))
predRegTestModeloSinPCA_36 = rep(0, length(probRegTestModeloSinPCA_36))
predRegTestModeloSinPCA_36[probRegTestModeloSinPCA_36 >=0.5] = 1
errorRegTestModeloSinPCA_36 = mean(predRegTestModeloSinPCA_36 != clasesTest)
```

```
cat("Error fuera de la muestra (E_test) usando el modelo glmnetSinPCA_36:",
    errorRegTestModeloSinPCA_36, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmnetSinPCA_36: 0.06705729
```

```
probRegTestModeloPCA_57 = predict(glmnetPCA_57, s = mejorLambda,
                                  newx = as.matrix(datosTestModeloPCA_57))
predRegTestModeloPCA_57 = rep(0, length(probRegTestModeloPCA_57))
predRegTestModeloPCA_57[probRegTestModeloPCA_57 >=0.5] = 1
errorRegTestModeloPCA_57 = mean(predRegTestModeloPCA_57 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmnetPCA_57:",
    errorRegTestModeloPCA_57, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmnetPCA_57: 0.07486979
```

```
probRegTestModeloSinPCA_57 = predict(glmnetSinPCA_57, s = mejorLambda,
                                     newx = as.matrix(datosTestModeloSinPCA_57))
predRegTestModeloSinPCA_57 = rep(0, length(probRegTestModeloSinPCA_57))
predRegTestModeloSinPCA_57[probRegTestModeloSinPCA_57 >=0.5] = 1
errorRegTestModeloSinPCA_57 = mean(predRegTestModeloSinPCA_57 != clasesTest)
cat("Error fuera de la muestra (E_test) usando el modelo glmnetSinPCA_57:",
    errorRegTestModeloSinPCA_57, "\n")
```

```
## Error fuera de la muestra (E_test) usando el modelo glmnetSinPCA_57: 0.06640625
```

Como podemos ver, tal y como habiamos estimado al comparar las desviaciones, obtenemos unos errores mayores usando regularización frente a no usarla, por lo tanto descartamos los modelos que usan regularización como candidatos.

Puede darse el caso de que no sea igual un tipo de error que otro, es decir, que sea más grave predecir la clase uno y que realmente sea la clase dos que predecir que sea la clase dos y que realmente sea la clase uno. Para ello, vamos a ver las matrices de confusión, las cuales representan como se han predecido las muestras y como son realmente:

```
table(predTestModeloPCA_1, clasesTest)
```

```
##                    clasesTest
## predTestModeloPCA_1   0   1
##                   0 868 111
##                   1  73 484
```

```
table(predTestModeloSinPCA_1, clasesTest)
```

```
##                       clasesTest
## predTestModeloSinPCA_1   0   1
##                       0 883 258
##                       1  58 337
```

```
table(predTestModeloPCA_29, clasesTest)
```

```
##                     clasesTest
## predTestModeloPCA_29   0   1
##                     0 898  51
##                     1  43 544
```

```
table(predTestModeloSinPCA_29, clasesTest)
```

```
##                       clasesTest
## predTestModeloSinPCA_29   0   1
```

```
##                       0 890  50
##                       1  51 545
```

```
table(predTestModeloPCA_36, clasesTest)
```

```
##                     clasesTest
## predTestModeloPCA_36   0   1
##                    0 899  54
##                    1  42 541
```

```
table(predTestModeloSinPCA_36, clasesTest)
```

```
##                       clasesTest
## predTestModeloSinPCA_36   0   1
##                       0 890  54
##                       1  51 541
```

```
table(predTestModeloPCA_57, clasesTest)
```

```
##                     clasesTest
## predTestModeloPCA_57   0   1
##                    0 899  59
##                    1  42 536
```

```
table(predTestModeloSinPCA_57, clasesTest)
```

```
##                       clasesTest
## predTestModeloSinPCA_57   0   1
##                       0 892  50
##                       1  49 545
```

Una forma más de determinar si un modelo es mejor que otro es usando las curvas ROC. Veamos las curvas ROC de nuestros modelos:

```
library("ROCR")
```

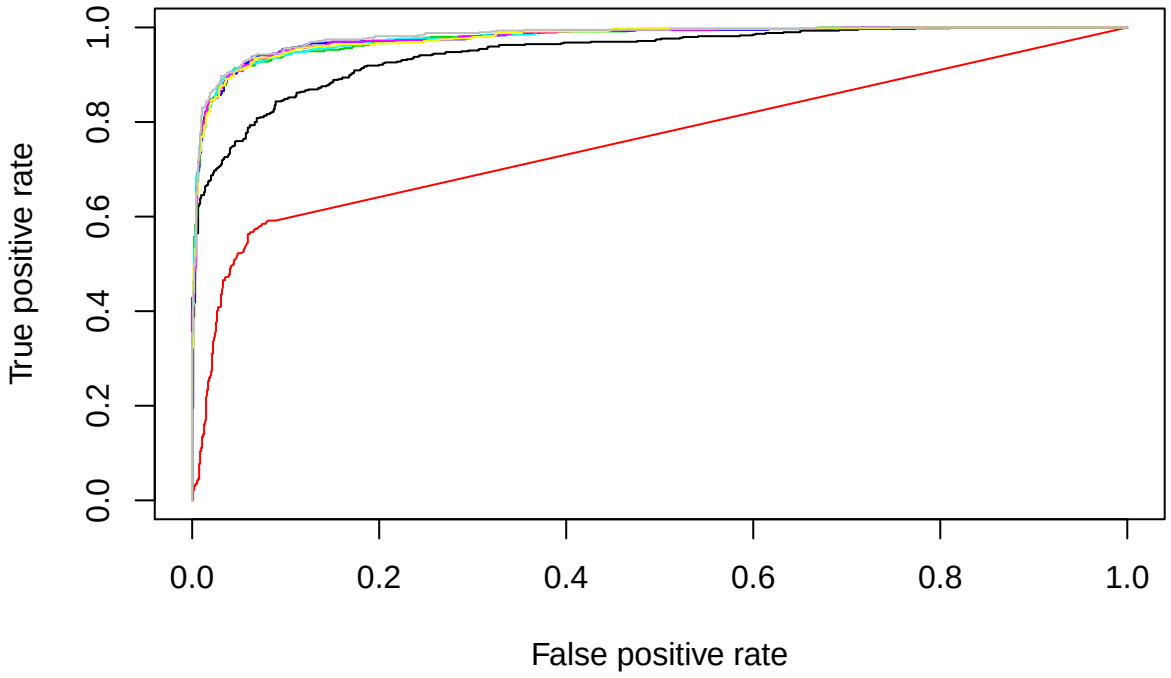
```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
prediccionTestModeloPCA_1 = prediction(probTestModeloPCA_1, clasesTest)
rendTestModeloPCA_1 = performance(prediccionTestModeloPCA_1,"tpr","fpr")

prediccionTestModeloSinPCA_1 = prediction(probTestModeloSinPCA_1, clasesTest)
rendTestModeloSinPCA_1 = performance(prediccionTestModeloSinPCA_1,"tpr","fpr")

prediccionTestModeloPCA_29 = prediction(probTestModeloPCA_29, clasesTest)
rendTestModeloPCA_29 = performance(prediccionTestModeloPCA_29,"tpr","fpr")

prediccionTestModeloSinPCA_29 = prediction(probTestModeloSinPCA_29, clasesTest)
rendTestModeloSinPCA_29 = performance(prediccionTestModeloSinPCA_29,"tpr","fpr")

prediccionTestModeloPCA_36 = prediction(probTestModeloPCA_36, clasesTest)
rendTestModeloPCA_36 = performance(prediccionTestModeloPCA_36,"tpr","fpr")
```

```
prediccionTestModeloSinPCA_36 = prediction(probTestModeloSinPCA_36, clasesTest)
rendTestModeloSinPCA_36 = performance(prediccionTestModeloSinPCA_36,"tpr","fpr")

prediccionTestModeloPCA_57 = prediction(probTestModeloPCA_57, clasesTest)
rendTestModeloPCA_57 = performance(prediccionTestModeloPCA_57,"tpr","fpr")

prediccionTestModeloSinPCA_57 = prediction(probTestModeloSinPCA_57, clasesTest)
rendTestModeloSinPCA_57 = performance(prediccionTestModeloSinPCA_57,"tpr","fpr")


plot(rendTestModeloPCA_1, col = 1)
plot(rendTestModeloSinPCA_1, add = T, col = 2)
plot(rendTestModeloPCA_29, add = T, col = 3)
plot(rendTestModeloSinPCA_29, add = T, col = 4)
plot(rendTestModeloPCA_36, add = T, col = 5)
plot(rendTestModeloSinPCA_36, add = T, col = 6)
plot(rendTestModeloPCA_57, add = T, col = 7)
plot(rendTestModeloSinPCA_57, add = T, col = 8)
```



Otra forma de elegir un modelo es el que tenga un área mayor por debajo de la curva ROC. Vamos a calcular las areas de las curvas obtenidas:

```
library("pracma")
```

```
##
## Attaching package: 'pracma'
```

```
## The following objects are masked from 'package:Matrix':
##
##     expm, lu, tril, triu
```

```
cat("Area debajo de la curva rendTestModeloPCA_1:",
    trapz(unlist(rendTestModeloPCA_1@x.values), unlist(rendTestModeloPCA_1@y.values)),
```

```
    "\n")
```

```
## Area debajo de la curva rendTestModeloPCA_1: 0.9463123
```

```
cat("Area debajo de la curva rendTestModeloSinPCA_1:",
    trapz(unlist(rendTestModeloSinPCA_1@x.values), unlist(rendTestModeloSinPCA_1@y.values)),
    "\n")
```

```
## Area debajo de la curva rendTestModeloSinPCA_1: 0.7627234
```

```
cat("Area debajo de la curva rendTestModeloPCA_29:",
    trapz(unlist(rendTestModeloPCA_29@x.values), unlist(rendTestModeloPCA_29@y.values)),
    "\n")
```

```
## Area debajo de la curva rendTestModeloPCA_29: 0.9777664
```

```
cat("Area debajo de la curva rendTestModeloSinPCA_29:",
    trapz(unlist(rendTestModeloSinPCA_29@x.values), unlist(rendTestModeloSinPCA_29@y.values)),
    "\n")
```

```
## Area debajo de la curva rendTestModeloSinPCA_29: 0.9786451
```

```
cat("Area debajo de la curva rendTestModeloPCA_36:",
    trapz(unlist(rendTestModeloPCA_36@x.values), unlist(rendTestModeloPCA_36@y.values)),
    "\n")
```

```
## Area debajo de la curva rendTestModeloPCA_36: 0.9782343
```

```
cat("Area debajo de la curva rendTestModeloSinPCA_36:",
    trapz(unlist(rendTestModeloSinPCA_36@x.values), unlist(rendTestModeloSinPCA_36@y.values)),
    "\n")
```

```
## Area debajo de la curva rendTestModeloSinPCA_36: 0.9785147
```

```
cat("Area debajo de la curva rendTestModeloPCA_58:",
    trapz(unlist(rendTestModeloPCA_57@x.values), unlist(rendTestModeloPCA_57@y.values)),
    "\n")
```

```
## Area debajo de la curva rendTestModeloPCA_58: 0.9780253
```

```
cat("Area debajo de la curva rendTestModeloSinPCA_58:",
    trapz(unlist(rendTestModeloSinPCA_57@x.values), unlist(rendTestModeloSinPCA_57@y.values)),
    "\n")
```

```
## Area debajo de la curva rendTestModeloSinPCA_58: 0.9820636
```

Veamos una tabla resumen para comparar los errores obtenidos:

Modelo	E_test
glmPCA_1	0.1197917
glmSinPCA_1	0.2057292
glmPCA_29	0.06119792
glmSinPCA_29	0.06575521
glmPCA_36	0.0625
glmSinPCA_36	0.06835938
glmPCA_57	0.06575521
glmSinPCA_57	0.06445312
glmnetPCA_29	0.07291667
glmnetSinPCA_29	0.06966146
glmnetPCA_36	0.07877604

Modelo	E_test
glmnetSinPCA_36	0.06705729
glmnetPCA_57	0.07486979
glmnetSinPCA_57	0.06640625

Viendo los resultados obtenidos, nuestro mejor modelo es aquel que nos proporciona un menor error fuera de la muestra. En nuestro caso, el modelo *glmPCA_29*.

8. Estimacion del error E_{out} del modelo lo más ajustada posible.

Dado que hemos hecho train y test y que este venía prefijado, el único error que podemos proporcionar es el obtenido en el apartado enterior. Por lo tanto, nuestro E_{out} para el mejor modelo, *glmPCA_29*, es 0.06119792, es decir, un 6.1% de error.

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

Estamos ante un modelo de buena calidad por distintos motivos. El primero de ellos es que hemos obtenido una tasa de error baja, lo cual es bueno de cara a clasificar futuros datos. La otra parte interesante es que hemos obtenido un modelo bastante bueno usando sólo 29 características frente a las 57 originales. Esto último es bueno de cara a extender esta idea a problemas de mayores características. Como conclusión nos gustaría decir que nos ha sorprendido bastante este hecho, que usando sólo parte de las características se obtengan mejores resultados que usando todas las características posibles. Otra de las cosas que nos ha impresionado también ha sido que, en los modelos en los que se ha usado una única característica, no se ha obtenido un error extremadamente alto. Esto nos ha parecido interesante en el sentido de que, si la característica usada es buena (lo cual hemos visto que si lo es gracias a *regsubsets*) podemos obtener una buena tasa de clasificación usando una única característica en vez de las 57 que tiene nuestro problema original.

A la vista de los resultados de las matrices de confusión y teniendo en cuenta que nuestro mejor modelo es el de 29 características con PCA (es el que vamos a comentar), observamos que fallamos practicamente lo mismo al clasificar una clase u otra y teniendo en cuenta que no tiene mayor importancia fallar en una clase u otra pues solo indica si un correo es spam o no, podemos concluir que el modelo es bueno.

Finalmente, nuestro mejor modelo, como ya se ha comentado anteriormente, ha sido *glmPCA_29*. Este modelo se ha obtenido usando un análisis de componentes principales, cogiendo las 29 características más importantes según *regsubsets* y aplicando regresión logística.

Problema de regresión

Para este problema hemos elegido la base de datos *prostate*.

1. Comprender al problema a resolver.

Esta base de datos consta de 97 ejemplos, con 8 características cada ejemplo (sin contar la clase). Dentro de la base de datos, la última columna representa el valor que queremos predecir. Las 8 características son valores que miden información obtenida de los sujetos, como puede ser la edad o el logaritmo del volumen del cancer observado.

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

En esta base de datos, al igual que en la base usada en clasificación, nos da la partición de los conjuntos de train y test, por lo que no será necesario usar validación u otras técnicas.

```
prostate = read.table("./datos/prostate.data", sep=" ", head=T)
# Quitamos el identificador
prostate = prostate[,-1]
# y el indicar de si va a train o no
trainTest = prostate[,dim(prostate)[2]]
prostate = prostate[,-dim(prostate)[2]]
# Vemos como debemos particionar los conjuntos
indexTrain = which(trainTest == T)
indexTest = which(trainTest == F)
train = prostate[indexTrain,]
test = prostate[indexTest,]
```

3. Preprocesado los datos: Falta de datos, categorización, normalización, reducción de dimensionalidad, etc.

En la información del dataset se nos dice que debemos centrar y escalar los datos antes de procesarlos. En el apartado de regresión líneal lo hicimos usando la función *preProcess*, así que aquí hacemos lo mismo. En este caso no vamos a aplicar la técnica PCA ya que tenemos muy pocas características y no será necesario eliminar más para trabajar con los datos, no obstante veremos posteriormente las características más relevantes que nos indica la función *regsubsets*.

```
library("caret")

ObjetoTrans = preProcess(train, method = c("YeoJohnson", "center", "scale"), thresh=1)
train = predict(ObjetoTrans, train)
test = predict(ObjetoTrans, test)
```

4. Selección de clases de funciones a usar.

Al igual que en el problema anterior las clases de funciones que vamos a usar son las funciones líneales.

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

Vamos a ser positivos y vamos a seguir pensando que viene bien hacer regularización a pesar de que en el apartado de clasificación no hayamos obtenido buenos resultados.

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

Como se ha comentado en el apartado del preprocesamiento, aplicar PCA no tiene sentido en este problema debido al reducido número de características que tenemos, así que vamos a continuar por el uso de la función *regsubsets* para ver cuáles son las características más importantes y ajustar distintos modelos en función de las mismas.

Veamos cuales son las características más importantes nuestro dataset:

```
library("leaps")

subset = regsubsets(train$lpsa ~ . , data = train, nvmax = dim(train)[2], method = "forward")
summary = summary(subset)
```

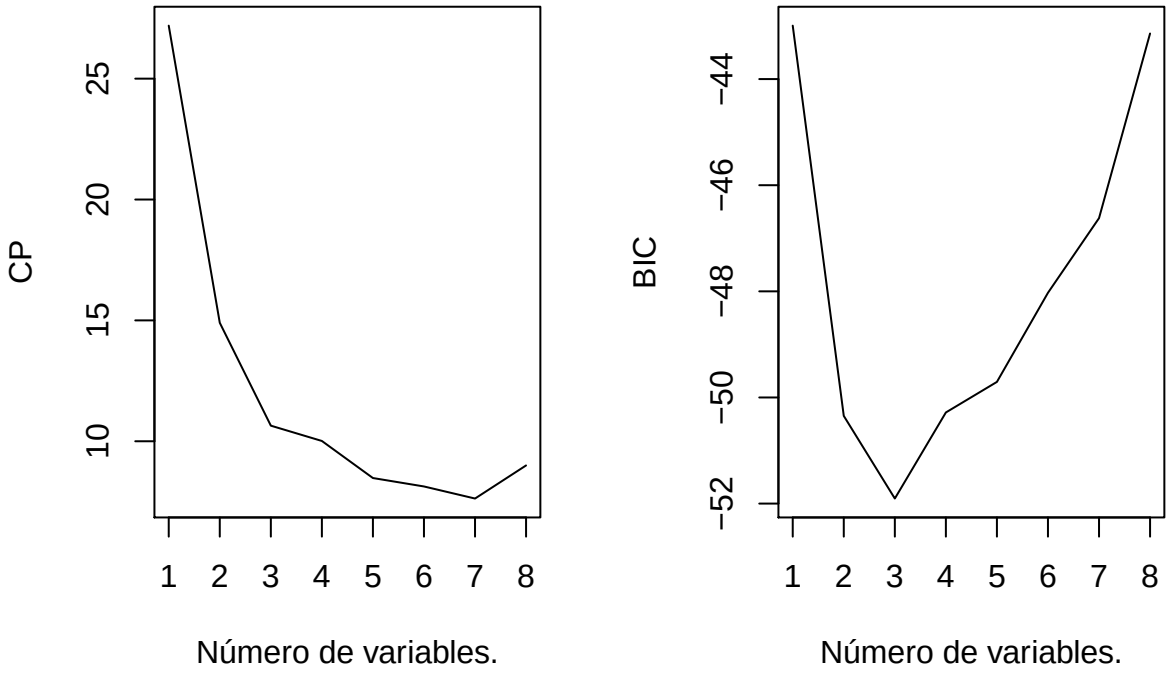
Las características que nos interesan aparecen en la siguiente tabla:

```
summary$outmat
```

```
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  " " " " " "     "*"
## 4  ( 1 ) "*"    "*"     " " " "  "*" " " " "     "*"
## 5  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     "*"
## 6  ( 1 ) "*"    "*"     "*" "*"  "*" " " " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     "*"
```

Una vez sabemos cuales son las mejores características en función del número de las mismas que deseemos usar, vamos a ver cual es la cantidad óptima de características que debemos usar. Para ello vamos a pintar dos gráficas (en función de los valores CP y BIC) y vamos a calcular donde se alcanza el valor mínimo:

```
par(mfrow=c(1,2))
plot(summary$cp, xlab = "Número de variables.", ylab = "CP", type = "l")
plot(summary$bic, xlab = "Número de variables.", ylab = "BIC", type = "l")
```




```
cat("Mejor número de características - CP:", which.min(summary$cp), "\n")
```

```
## Mejor número de características - CP: 7
```

```
cat("Mejor número de características - BIC:", which.min(summary$bic), "\n")
```

```
## Mejor número de características - BIC: 3
```

Podemos ver que los mejores resultados se obtiene usando 7 características según CP y 3 según BIC. Las características son las siguientes:

```
as.vector(which(summary$outmat[7,] == "*"))
```

```
## [1] 1 2 3 4 5 6 8
```

```r
as.vector(which(summary$outmat[3,] == "*"))
```

```
## [1] 1 2 8
```

Si quisiéramos usar una única característica sería:

```r
as.vector(which(summary$outmat[1,] == "*"))
```

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

Una vez sabemos qué características usar, podemos ir construyendo nuestros primeros modelos con regresión lineal ya que con logistica se nos indica que la característica a predecir debe estar entre 0 y 1 (son probabilidades) y en nuestro dataset esto no ocurre :

```r
datosTrainModelo_1 = as.data.frame(train[, c(as.vector(which(summary$outmat[1,] == "*")),
                                             dim(train)[2])])
datosTrainModelo_3 = as.data.frame(train[, c(as.vector(which(summary$outmat[3,] == "*")),
                                             dim(train)[2])])
datosTrainModelo_7 = as.data.frame(train[, c(as.vector(which(summary$outmat[7,] == "*")),
                                             dim(train)[2])])
datosTrainModelo_8 = as.data.frame(train)

datosTestModelo_1 = as.data.frame(test[, c(as.vector(which(summary$outmat[1,] == "*")),
                                           dim(train)[2])])

datosTestModelo_3 = as.data.frame(test[, c(as.vector(which(summary$outmat[3,] == "*")),
                                           dim(train)[2])])
datosTestModelo_7 = as.data.frame(test[, c(as.vector(which(summary$outmat[7,] == "*")),
                                           dim(train)[2])])
datosTestModelo_8 = as.data.frame(test)

# Una vez tenemos los conjuntos de datos, aprendemos los modelos
# Usamos regresión lineal
glm_1 = glm(datosTrainModelo_1$lpsa ~ ., data = datosTrainModelo_1)
glm_3 = glm(datosTrainModelo_3$lpsa ~ ., data = datosTrainModelo_3)
glm_7 = glm(datosTrainModelo_7$lpsa ~ ., data = datosTrainModelo_7)
glm_8 = glm(datosTrainModelo_8$lpsa ~ ., data = datosTrainModelo_8)
```

Primero vamos a ver si hace falta aplicar regularización. Para ello vamos a comparar las desviaciones en el caso de no usar regularización (usando lambda = 0) y en el caso de usarla (usando como lambda el mejor lambda estimado).

```r
library("glmnet")
a = cv.glmnet(x = as.matrix(datosTrainModelo_3[,-dim(datosTrainModelo_3)[2]]),
              y = as.matrix(datosTrainModelo_3$lpsa), alpha = 0)
mejorLambda_3 = a$lambda.min
# Sin regularización
glmnet(x = as.matrix(datosTrainModelo_3[,-dim(datosTrainModelo_3)[2]]),
       y = as.matrix(datosTrainModelo_3$lpsa), alpha = 0,
       lambda = 0, standardize = FALSE)
```

```
##
## Call:  glmnet(x = as.matrix(datosTrainModelo_3[, -dim(datosTrainModelo_3)[2]]),      y = as.matrix(da
##
##      Df   %Dev Lambda
## [1,]  3 0.6415      0
```

```
# Con regularización
glmnet(x = as.matrix(datosTrainModelo_3[,-dim(datosTrainModelo_3)[2]]),
       y = as.matrix(datosTrainModelo_3$lpsa), alpha = 0,
       lambda = mejorLambda_3, standardize = FALSE)
```

```
##
## Call:  glmnet(x = as.matrix(datosTrainModelo_3[, -dim(datosTrainModelo_3)[2]]),      y = as.matrix(da
##
##      Df  %Dev  Lambda
## [1,]  3 0.639 0.09603
```

Como podemos ver, obtenemos unos valores de %Dev similares aunque un poco más bajos si usamos regularización, lo cual nos lleva a pensar que no es necesario aplicar regularización, aún así vamos a aplicarla para ver si obtenemos unos mejores resultados o se confirma que no es necesario hacerla (al igual que en la parte de clasificación). Los modelos que vamos a ajustar son con regularización (weight decay, ya que ponemos el parámetro alpha a 0).

```
a = cv.glmnet(x = as.matrix(datosTrainModelo_3[,-dim(datosTrainModelo_3)[2]]),
              y = as.matrix(datosTrainModelo_3$lpsa), alpha = 0)
mejorLambda_3 = a$lambda.min
# Si usamos alpha = 0 hace WD
glmnet_3 = glmnet(x = as.matrix(datosTrainModelo_3[,-dim(datosTrainModelo_3)[2]]),
                  y = as.matrix(datosTrainModelo_3$lpsa), alpha = 0,
                  lambda = mejorLambda_3, standardize = FALSE)

a = cv.glmnet(x = as.matrix(datosTrainModelo_7[,-dim(datosTrainModelo_7)[2]]),
              y = as.matrix(datosTrainModelo_7$lpsa), alpha = 0)
mejorLambda_7 = a$lambda.min
glmnet_7 = glmnet(x = as.matrix(datosTrainModelo_7[,-dim(datosTrainModelo_7)[2]]),
                  y = as.matrix(datosTrainModelo_7$lpsa), alpha = 0,
                  lambda = mejorLambda_7, standardize = FALSE)

a = cv.glmnet(x = as.matrix(datosTrainModelo_8[,-dim(datosTrainModelo_8)[2]]),
              y = as.matrix(datosTrainModelo_8$lpsa), alpha = 0)
mejorLambda_8 = a$lambda.min
glmnet_8 = glmnet(x = as.matrix(datosTrainModelo_8[,-dim(datosTrainModelo_8)[2]]),
                  y = as.matrix(datosTrainModelo_8$lpsa), alpha = 0,
                  lambda = mejorLambda_8, standardize = FALSE)
```

Finalmente, en clase hemos visto que se pueden hacer transformaciones sobre los datos, vamos a hacer una transformación de segundo grado:

```
datosTransformadosTrainModelo_1 = cbind(as.data.frame(polym(datosTrainModelo_1$lcavol,
                                                            degree = 2, raw = T)),
                                        datosTrainModelo_1$lpsa)

datosTransformadosTrainModelo_3 = cbind(as.data.frame(polym(datosTrainModelo_3$lcavol,
                                                            datosTrainModelo_3$lweight,
                                                            datosTrainModelo_3$pgg45,
                                                            degree = 2, raw = T)),
                                        datosTrainModelo_3$lpsa)

datosTransformadosTrainModelo_7 = cbind(as.data.frame(polym(datosTrainModelo_7$lcavol,
                                                            datosTrainModelo_7$lweight,
                                                            datosTrainModelo_7$age,
```

```
                                                    datosTrainModelo_7$lbph,
                                                    datosTrainModelo_7$svi,
                                                    datosTrainModelo_7$lcp,
                                                    datosTrainModelo_7$pgg45,
                                                    degree = 2, raw = T)),
                                        datosTrainModelo_7$lpsa)

datosTransformadosTrainModelo_8 = cbind(as.data.frame(polym(datosTrainModelo_8$lcavol,
                                                    datosTrainModelo_8$lweight,
                                                    datosTrainModelo_8$age,
                                                    datosTrainModelo_8$lbph,
                                                    datosTrainModelo_8$svi,
                                                    datosTrainModelo_8$lcp,
                                                    datosTrainModelo_8$gleason,
                                                    datosTrainModelo_8$pgg45,
                                                    degree = 2, raw = T)),
                                        datosTrainModelo_8$lpsa)

names = names(datosTransformadosTrainModelo_1)
names(datosTransformadosTrainModelo_1) = c(names[-length(names)], "lpsa")
names = names(datosTransformadosTrainModelo_3)
names(datosTransformadosTrainModelo_3) = c(names[-length(names)], "lpsa")
names = names(datosTransformadosTrainModelo_7)
names(datosTransformadosTrainModelo_7) = c(names[-length(names)], "lpsa")
names = names(datosTransformadosTrainModelo_8)
names(datosTransformadosTrainModelo_8) = c(names[-length(names)], "lpsa")


datosTransformadosTestModelo_1 = cbind(as.data.frame(polym(datosTestModelo_1$lcavol,
                                                    degree = 2, raw = T)),
                                        datosTestModelo_1$lpsa)

datosTransformadosTestModelo_3 = cbind(as.data.frame(polym(datosTestModelo_3$lcavol,
                                                   datosTestModelo_3$lweight,
                                                    datosTestModelo_3$pgg45,
                                                    degree = 2, raw = T)),
                                        datosTestModelo_3$lpsa)

datosTransformadosTestModelo_7 = cbind(as.data.frame(polym(datosTestModelo_7$lcavol,
                                                    datosTestModelo_7$lweight,
                                                    datosTestModelo_7$age,
                                                    datosTestModelo_7$lbph,
                                                    datosTestModelo_7$svi,
                                                    datosTestModelo_7$lcp,
                                                    datosTestModelo_7$pgg45,
                                                    degree = 2, raw = T)),
                                        datosTestModelo_7$lpsa)

datosTransformadosTestModelo_8 = cbind(as.data.frame(polym(datosTestModelo_8$lcavol,
                                                    datosTestModelo_8$lweight,
                                                    datosTestModelo_8$age,
                                                    datosTestModelo_8$lbph,
                                                    datosTestModelo_8$svi,
```

```
                                                    datosTestModelo_8$lcp,
                                                    datosTestModelo_8$gleason,
                                                    datosTestModelo_8$pgg45,
                                                    degree = 2, raw = T)),
                                        datosTestModelo_8$lpsa)

names = names(datosTransformadosTestModelo_1)
names(datosTransformadosTestModelo_1) = c(names[-length(names)], "lpsa")
names = names(datosTransformadosTestModelo_3)
names(datosTransformadosTestModelo_3) = c(names[-length(names)], "lpsa")
names = names(datosTransformadosTestModelo_7)
names(datosTransformadosTestModelo_7) = c(names[-length(names)], "lpsa")
names = names(datosTransformadosTestModelo_8)
names(datosTransformadosTestModelo_8) = c(names[-length(names)], "lpsa")
```

Una vez tenemos los datos tranformados, vamos a ajustar los modelos:

```
glmTransformados_1 = glm(datosTransformadosTrainModelo_1$lpsa ~ .,
                         data = datosTransformadosTrainModelo_1)
glmTransformados_3 = glm(datosTransformadosTrainModelo_3$lpsa ~ .,
                         data = datosTransformadosTrainModelo_3)
glmTransformados_7 = glm(datosTransformadosTrainModelo_7$lpsa ~ .,
                         data = datosTransformadosTrainModelo_7)
glmTransformados_8 = glm(datosTransformadosTrainModelo_8$lpsa ~ .,
                         data = datosTransformadosTrainModelo_8)
```

7. Selección y ajuste modelo final.

Como medida de que modelo es mejor que otro vamos a usar el error cuadrático medio, así que vamos a calcular dichos errores para todos los modelos que hemos ajustado anteriormente. Al igual que hemos hecho en el caso de clasificación, primero debemos ver si nuestros modelos han sobreaprendido. Para ello vamos a calcular el error dentro de la muestra y compararlo con el error en test.

```
probTrainModelo1 = predict(glm_1, as.data.frame(datosTrainModelo_1),
                           type="response", exact = T)
cat("Error dentro de la muestra (E_Train) usando el modelo glm_1:",
    mean((probTrainModelo1 - datosTrainModelo_1$lpsa)^2), "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glm_1: 0.4573822
```

```
probTrainModelo3 = predict(glm_3, as.data.frame(datosTrainModelo_3),
                           type="response", exact = T)
cat("Error dentro de la muestra (E_Train) usando el modelo glm_3:",
    mean((probTrainModelo3 - datosTrainModelo_3$lpsa)^2), "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glm_3: 0.3531859
```

```
probTrainModelo7 = predict(glm_7, as.data.frame(datosTrainModelo_7),
                           type="response", exact = T)
cat("Error dentro de la muestra (E_Train) usando el modelo glm_7:",
    mean((probTrainModelo7 - datosTrainModelo_7$lpsa)^2), "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glm_7: 0.2973287
```

```
probTrainModelo8 = predict(glm_8, as.data.frame(datosTrainModelo_8),
                           type="response", exact = T)
```

```
cat("Error dentro de la muestra (E_Train) usando el modelo glm_8:",
    mean((probTrainModelo8 - datosTrainModelo_8$lpsa)^2), "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glm_8: 0.294128
```

```
probTestModelo1 = predict(glm_1, as.data.frame(datosTestModelo_1),
                          type="response", exact = T)
cat("Error fuera de la muestra (E_Test) usando el modelo glm_1:",
    mean((probTestModelo1 - datosTestModelo_1$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glm_1: 0.3534174
```

```
probTestModelo3 = predict(glm_3, as.data.frame(datosTestModelo_3),
                          type="response", exact = T)
cat("Error fuera de la muestra (E_Test) usando el modelo glm_3:",
    mean((probTestModelo3 - datosTestModelo_3$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glm_3: 0.3748923
```

```
probTestModelo7 = predict(glm_7, as.data.frame(datosTestModelo_7),
                          type="response", exact = T)
cat("Error fuera de la muestra (E_Test) usando el modelo glm_7:",
    mean((probTestModelo7 - datosTestModelo_7$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glm_7: 0.3699996
```

```
probTestModelo8 = predict(glm_8, as.data.frame(datosTestModelo_8),
                          type="response", exact = T)
cat("Error fuera de la muestra (E_Test) usando el modelo glm_8:",
    mean((probTestModelo8 - datosTestModelo_8$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glm_8: 0.3906221
```

Como podemos ver obtenemos unos errores similares, por lo tanto no podemos decir que nuestros modelos hayan sobreaprendido. Calcularemos ahora el error para los modelos con regularización, tanto dentro como fuera de la muestra:

```
probTrainModeloReg3 = predict(glmnet_3, s = mejorLambda_3,
                              newx = as.matrix(datosTrainModelo_3[,-dim(datosTrainModelo_3)[2]]))
cat("Error dentro de la muestra (E_Train) usando el modelo glmnet_3:",
    mean((probTrainModeloReg3 - datosTrainModelo_3$lpsa)^2), "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmnet_3: 0.3556016
```

```
probTrainModeloReg7 = predict(glmnet_7, s = mejorLambda_7,
                              newx = as.matrix(datosTrainModelo_7[,-dim(datosTrainModelo_7)[2]]))
cat("Error dentro de la muestra (E_Train) usando el modelo glmnet_7:",
    mean((probTrainModeloReg7 - datosTrainModelo_7$lpsa)^2), "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmnet_7: 0.3020642
```

```
probTrainModeloReg8 = predict(glmnet_8, s = mejorLambda_8,
                              newx = as.matrix(datosTrainModelo_8[,-dim(datosTrainModelo_8)[2]]))
cat("Error dentro de la muestra (E_Train) usando el modelo glmnet_8:",
    mean((probTrainModeloReg8 - datosTrainModelo_8$lpsa)^2), "\n")
```

```
## Error dentro de la muestra (E_Train) usando el modelo glmnet_8: 0.3022247
```

```
probTestModeloReg3 = predict(glmnet_3, s = mejorLambda_3,
                              newx = as.matrix(datosTestModelo_3[,-dim(datosTestModelo_3)[2]]))
```

```
cat("Error fuera de la muestra (E_Test) usando el modelo glmnet_3:",
    mean((probTestModeloReg3 - datosTestModelo_3$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glmnet_3: 0.3809695
```

```
probTestModeloReg7 = predict(glmnet_7, s = mejorLambda_7,
                             newx = as.matrix(datosTestModelo_7[,-dim(datosTestModelo_7)[2]]))
cat("Error fuera de la muestra (E_Test) usando el modelo glmnet_7:",
    mean((probTestModeloReg7 - datosTestModelo_7$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glmnet_7: 0.3637184
```

```
probTestModeloReg8 = predict(glmnet_8, s = mejorLambda_8,
                             newx = as.matrix(datosTestModelo_8[,-dim(datosTestModelo_8)[2]]))
cat("Error fuera de la muestra (E_Test) usando el modelo glmnet_8:",
    mean((probTestModeloReg8 - datosTestModelo_8$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glmnet_8: 0.3649505
```

Volvemos a lo mismo, podemos decir que los modelos no han sobreaprendido. A diferencia que en el apartado de clasificación, aquí la regularización si sirve, ya que tenemos algunos modelos con regularización que obtienen un error en test menores que los modelos sin regularizar. Más adelante veremos si alguno de esos modelos con regularización es el mejor. Calcularemos ahora el error fuera de la muestra para los modelos con transformaciones sobre los datos :

```
probTestTransformadosModelo1 = predict(glmTransformados_1,
                                       as.data.frame(datosTransformadosTestModelo_1[,
                                       -dim(datosTransformadosTestModelo_1)[2]]),
                                       type="response", exact = T)
cat("Error fuera de la muestra (E_Test) usando el modelo glmTransformados_1:",
    mean((probTestTransformadosModelo1 - datosTransformadosTestModelo_1$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glmTransformados_1: 0.3561855
```

```
probTestTransformadosModelo3 = predict(glmTransformados_3,
                                       as.data.frame(datosTransformadosTestModelo_3[,
                                       -dim(datosTransformadosTestModelo_3)[2]]),
                                       type="response", exact = T)
cat("Error fuera de la muestra (E_Test) usando el modelo glmTransformados_3:",
    mean((probTestTransformadosModelo3 - datosTransformadosTestModelo_3$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glmTransformados_3: 0.3688455
```

```
probTestTransformadosModelo7 = predict(glmTransformados_7,
                                       as.data.frame(datosTransformadosTestModelo_7[,
                                       -dim(datosTransformadosTestModelo_7)[2]]),
                                       type="response", exact = T)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
cat("Error fuera de la muestra (E_Test) usando el modelo glmTransformados_7:",
    mean((probTestTransformadosModelo7 - datosTransformadosTestModelo_7$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glmTransformados_7: 0.8453743
```

```
probTestTransformadosModelo8 = predict(glmTransformados_8,
                                       as.data.frame(datosTransformadosTestModelo_8[,
```

```
                                        -dim(datosTransformadosTestModelo_8)[2]]),
                                        type="response", exact = T)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
cat("Error fuera de la muestra (E_Test) usando el modelo glmTransformados_8:",
    mean((probTestTransformadosModelo8 - datosTransformadosTestModelo_8$lpsa)^2), "\n")
```

```
## Error fuera de la muestra (E_Test) usando el modelo glmTransformados_8: 1.012884
```

Veamos una tabla resumen para comparar los errores obtenidos:

Modelo	E_test
glm_1	0.3534174
glm_3	0.3748923
glm_7	0.3699996
glm_8	0.3906221
glmnet_3	0.3799163
glmnet_7	0.3637184
glmnet_8	0.3656383
glmTransformados_1	0.3561855
glmTransformados_3	0.3688455
glmTransformados_7	0.8453743
glmTransformados_8	1.012884

Viendo los resultados obtenidos, nuestro mejor modelo es aquel que nos proporciona un menor error fuera de la muestra. En nuestro caso, el modelo *glm_1*.

8. Estimacion del error E_{out} del modelo lo más ajustada posible.

Dado que hemos hecho train y test y que este venía prefijado, el único error que podemos proporcionar es el obtenido en el apartado enterior. Por lo tanto, nuestro E_{out} para el mejor modelo, *glm_1*, es 0.3534174.

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

Como se ha comentado anteriormente el mejor modelo es *glm_1*, el cual no contiene regularización y solo tiene en cuenta una característica. Esto es bastante bueno puesto que con una sola características vamos a obtener buenas predicciones. No obstante los errores usando 1, 2, 7 y 8 características son muy similares. Además vemos que aplicar transformaciones de segundo grado a los datos no consigue aportar mejores resultados. En conclusión vemos que para esta base de datos, obtener un modelo con Regresión Lineal usando una sola característica es suficiente para obtener una buena predicción.