Regresión Logística vs. Random Forest

Jaqueline Girabel

El conjunto de datos **cleveland** contiene la información de un estudio sobre enfermedades cardíacas. En función de las variables 1 a 13, se trata de diagnosticar el estrechamiento de las arterias (variable 14). Algunas de estas variables explicativas son de tipo categóricas y sus clases están representadas con valores enteros, como lo son **cp**, **slope**, **thal**, entre otras. La variable respuesta, **diagnosis**, toma valores enteros entre 0 y 3, pero nos interesa entenderla como una variable binaria, siendo "0" cuando tome el valor 0, y "1" cuando tome valores mayores a 0.

Con el objetivo de ajustar un modelo que permita predecir un diagnóstico, usamos la regresión logística para construir este clasificador, dado que este es un método que maneja datos con variables de todo tipo (como es el caso de los datos Cleveland).

```
head(cleveland)
```

```
##   Age Sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1  63   1  1      145  233   1       2     150     0     2.3     3  0    6
## 2  67   1  4      160  286   0       2     108     1     1.5     2  3    3
## 3  67   1  4      120  229   0       2     129     1     2.6     2  2    7
## 4  37   1  3      130  250   0       0     187     0     3.5     3  0    3
## 5  41   0  2      130  204   0       2     172     0     1.4     1  0    3
## 6  56   1  2      120  236   0       0     178     0     0.8     1  0    3
##   diagnosis
## 1         0
## 2         2
## 3         1
## 4         0
## 5         0
## 6         0
```

Transformamos **diagnosis** en una variable binaria:

```
cleveland$diagnosis[cleveland$diagnosis>0]<-1
cleveland$diagnosis<-as.factor(cleveland$diagnosis)
```

Notamos, además, algunos valores ausentes en el dataset que serán removidos. Los contabilizamos por columna de la siguiente manera:

```
library(purrr)

valores_ausentes_por_columna <- map_dbl(cleveland, function(x){sum(x==-999)})
valores_ausentes_por_columna
```

```
##       Age       Sex        cp  trestbps      chol       fbs   restecg   thalach
##         0         0         0         0         0         0         0         0
##     exang   oldpeak     slope        ca      thal diagnosis
##         0         0         0         4         2         0
```

```
cleveland<-cleveland[ca!=-999 & thal!=-999,]
```

Ahora sí, separamos un 70%-30% de los datos para definir los conjuntos de entrenamiento, **cleveland.T**, y de testeo, **cleveland.V**.

```
set.seed(123)
indices<-sample (1: nrow(cleveland), round(nrow(cleveland)*0.7,0))
cleveland.T<-cleveland[indices,]
cleveland.V<-cleveland[-indices,]
```

Si ajustamos un modelo de regresión logistica sobre el set de entrenamiento y predecimos los resultados de la variable respuesta evaluando el modelo en el set de testeo con 0.5 como punto de corte, obtenemos el siguiente error de predicción:

```
modelo<- glm( diagnosis ~., data =cleveland.T , family = binomial )
modelo.probas<- predict(modelo,type ="response", cleveland.V)

predicciones<-ifelse(modelo.probas > 0.5, 1, 0)

#error de prediccion
round(mean(predicciones != cleveland.V$diagnosis),4)
```

```
## [1] 0.1236
```

En el siguiente resumen podemos ver algunos datos relevantes.

```
summary(modelo)
```

```
##
## Call:
## glm(formula = diagnosis ~ ., family = binomial, data = cleveland.T)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.79368  -0.43946  -0.09915   0.30680   2.72892
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.565793   4.016871  -2.132 0.032970 *
## Age         -0.019799   0.031475  -0.629 0.529319
## Sex1         2.011121   0.740273   2.717 0.006593 **
## cp2          1.593341   0.917730   1.736 0.082533 .
## cp3          0.266661   0.795550   0.335 0.737482
## cp4          2.300731   0.794773   2.895 0.003794 **
## trestbps     0.031736   0.014325   2.215 0.026734 *
## chol         0.004947   0.005874   0.842 0.399703
## fbs         -1.265594   0.815807  -1.551 0.120820
## restecg      0.274082   0.246607   1.111 0.266390
## thalach     -0.014203   0.015844  -0.896 0.370024
## exang        0.650763   0.567303   1.147 0.251333
## oldpeak      0.656160   0.291461   2.251 0.024368 *
## slope2       1.140469   0.615288   1.854 0.063803 .
## slope3       1.160098   1.077395   1.077 0.281587
## ca           1.749101   0.397998   4.395 1.11e-05 ***
## thal6        0.079548   0.995458   0.080 0.936308
## thal7        1.859570   0.540069   3.443 0.000575 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 286.42  on 207  degrees of freedom
## Residual deviance: 124.21  on 190  degrees of freedom
## AIC: 160.21
##
## Number of Fisher Scoring iterations: 6
```

Lo primero que notamos es que, si bien la regresión logistica admite predictores tanto numéricos como categóricos, la funcionalidad glm necesitó crear variables nuevas transformando cada variable categórica de 3 o más clases en dos o más variables binarias.

Por otro lado, la columna de z-values nos indica cierto nivel de relevancia de las variables explicativas, en términos de la varianza. Estos valores se corresponden con los resultados de un test cuya hipótesis nula es que los coeficientes son iguales a 0, por lo que z-values menores a 2 en módulo son significativos al 95%. Así, si el objetivo es hacer una selección de variables únicamente con esta información, entonces Sex, cp4, trestbps, oldpeak, ca y thal7 podrían ser cosideradas poco significativas y descartadas.

Las técnicas de regularización en normas L1 y L2, aplicables a modelos de regresión, permiten ajustar el modelo propuesto a fines de lograr un balance entre sesgo y varianza. En particular, la penalización impuesta en norma L1 (lasso) tiene la propiedad de forzar a los coeficientes menos significativos a ser exactamente iguales a 0; de manera que esta es una técnica que podemos emplear para hacer una selección de variables y, simultaneamente, ajustar el modelo en función de ellas. Dado que esta penalización depende de un parámetro λ, buscamos primero un valor de λ que resulte apropiado a fines de mejorar la exactitud del modelo, esto lo hacemos usando cross-validation (K-folds, con K=10) sobre el set de datos de entrenamiento para cada lambda en una grilla dada.

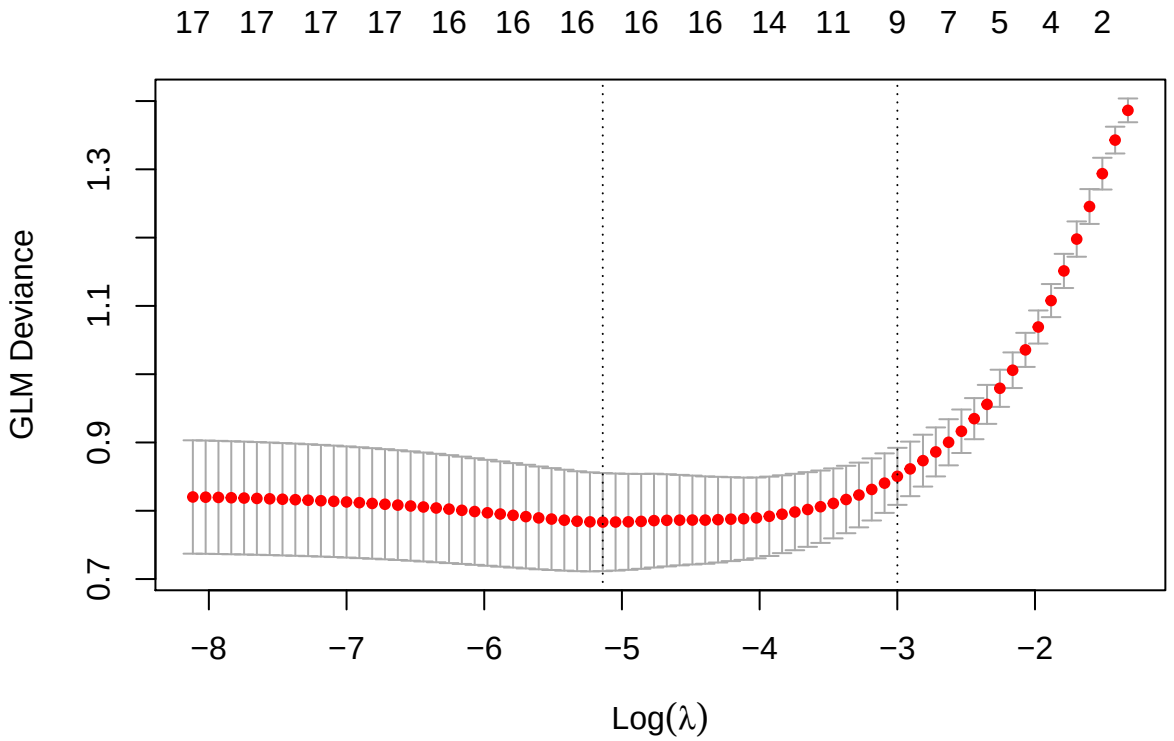
```
library(glmnet)

# model.matrix crea una matriz con las variables predictoras y tranforma cada variable categórica en un
x<-model.matrix(diagnosis~., cleveland )[,-1]
y<-ifelse(cleveland$diagnosis=="0",0,1)

x.T<-x[indices,]
x.V<-x[-indices,]
y.T<-y[indices]
y.V<-y[-indices]


set.seed(123)
modelo.cv<-cv.glmnet(x.T,y.T, alpha =1,family = binomial)

plot(modelo.cv)
```



El gráfico anterior nos muestra el error de la validación cruzada en función del logaritmo de lambda. Las rectas verticales punteadas indican dos valores del parámetro lambda a considerar: a la izquierda, **lambda.min**, es el que minimiza error de predicción; por otro lado, a la derecha, **lambda.max** es el que nos devuelve un modelo más simple (a fines de evitar overfitting) a cambio de un poco de exactitud. Teóricamente, cuanto más grande sea el valor del parámetro λ, más chicos en módulo son los coeficientes del modelo.

```
lambda.min<-modelo.cv$lambda.min
lambda.max<-modelo.cv$lambda.1se

lambda.min
```

```
## [1] 0.005862114
```

```
lambda.max
```

```
## [1] 0.04981352
```

Ajustamos, entonces, un **modelo.L1** usando **lambda.max** como mejor parámetro y calculamos el error de predicción sobre el set de testing, habiendo clasificado con el umbral 0.5.

```
set.seed(123)
modelo.L1<-glmnet(x.T, y.T, alpha =1, family = binomial, lambda = lambda.max)
probabilidades<-predict(modelo.L1,type ="response", newx =x.V)
predicciones.L1<-ifelse(probabilidades > 0.5, 1, 0)

error<-round(mean(predicciones.L1 != y.V),4)
error
```

```
## [1] 0.1461
```

Los coeficientes asociados a los predictores se muestran a continuación

```
coef(modelo.L1)
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept) -1.071198636
## Age          .
## Sex1         0.173801335
## cp2          .
## cp3          .
## cp4          0.959735049
## trestbps     .
## chol         .
## fbs          .
## restecg      .
## thalach     -0.002842517
## exang        0.133006643
## oldpeak      0.392394121
## slope2       0.024187243
## slope3       .
## ca           0.612601303
## thal3       -0.656803858
## thal6        .
## thal7        0.629598866
```

Por otro lado, si hubiésemos tomado lambda.min como mejor parámetro, los coeficientes del modelo serían

```
coef(modelo.cv, lambda.min)
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -6.835152499
## Age         -0.001020302
## Sex1         1.475634398
## cp2          0.775914487
## cp3          .
## cp4          1.729574670
## trestbps     0.020691572
## chol         0.003144998
## fbs         -0.762041694
## restecg      0.184963804
## thalach     -0.008494794
## exang        0.492595267
## oldpeak      0.597918396
## slope2       0.772382309
## slope3       0.640396633
## ca           1.337909220
## thal3       -0.216323187
## thal6        .
## thal7        1.409475722
```

Podemos concluir, entonces, que las variables más relevantes a la hora de predecir un diagnóstico son *Sex*, *cp4*, *thalach*, *exang*, *oldpeak*, *slope2*, *ca* y *thal7*.

Lo que queremos ahora es comparar el modelo.L1 con algún modelo basado en árboles de decisión, ya que estos últimos también admiten variables predictoras de todo tipo.

En lugar de ajustar un modelo simple - es decir, en lugar de construir un único árbol y aplicar un criterio de poda para controlar su complejidad-, directamente empleamos un modelo basado en réplicas bootstrapping. Los árboles obtenidos de las muestras bootstrap del set de entrenamiento no son podados, por lo que cada uno de ellos presenta poco sesgo pero varianza alta. Tomando el promedio de todos ellos, logramos reducir la varianza (siempre y cuando los árboles no estén correlacionados). Así, los resultados obtenidos no dependen tanto de la muestra original y podemos acercarnos más a la distribución de la que esta proviene.

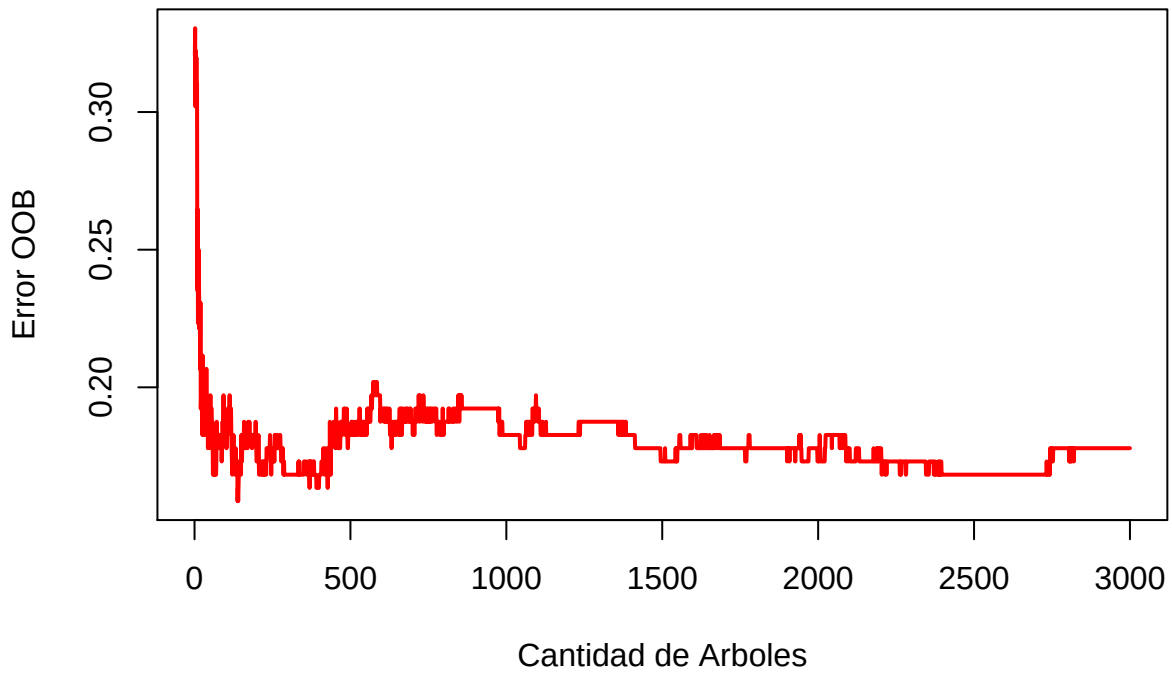
Usamos *random forest* para ajustar este modelo.

```
library(randomForest)

set.seed(123)
modelo.rf<-randomForest(diagnosis~., data=cleveland.T, ntree=3000 )
```

El siguiente gráfico describe al error Out Of Bag (OOB) en función de la cantidad de árboles, en el conjunto de entrenamiento. El error OOB es similar al error de testeo de tipo "leave-one-out" en la validación cruzada.

```
plot(modelo.rf$err.rate[,1], type="l", col="red", ylab = "Error OOB", xlab = "Cantidad de Arboles", lwd=
```



Incrementar la cantidad de árboles (réplicas bootstrap) no produce overfitting, por lo que podemos tomar un numero de árboles suficientemente grande a partir del cual el error se estabilice. Nosotros tomamos **ntree=500** :

```
set.seed(123)
modelo.rf<-randomForest(diagnosis~., data=cleveland.T, ntree=500, importance=TRUE)
```

Precedimos los resultados en el set cleveland.V y comparamos el error de predicción con el que obtuvimos usando regresión logística:

```
predicciones.rf<-predict (modelo.rf , newdata =cleveland.V)

round(mean(predicciones.rf!= cleveland.V$diagnosis),4)
```

```
## [1] 0.1798
```

En este modelo también es posible visualizar la importancia de las variables en términos de, en este caso, la exactitud. Estos valores se obtienen mediante permutaciones en los datos de cada una de las variables explicativas para ver cómo disminuye la predicción.

```
importance(modelo.rf, type=1)
```

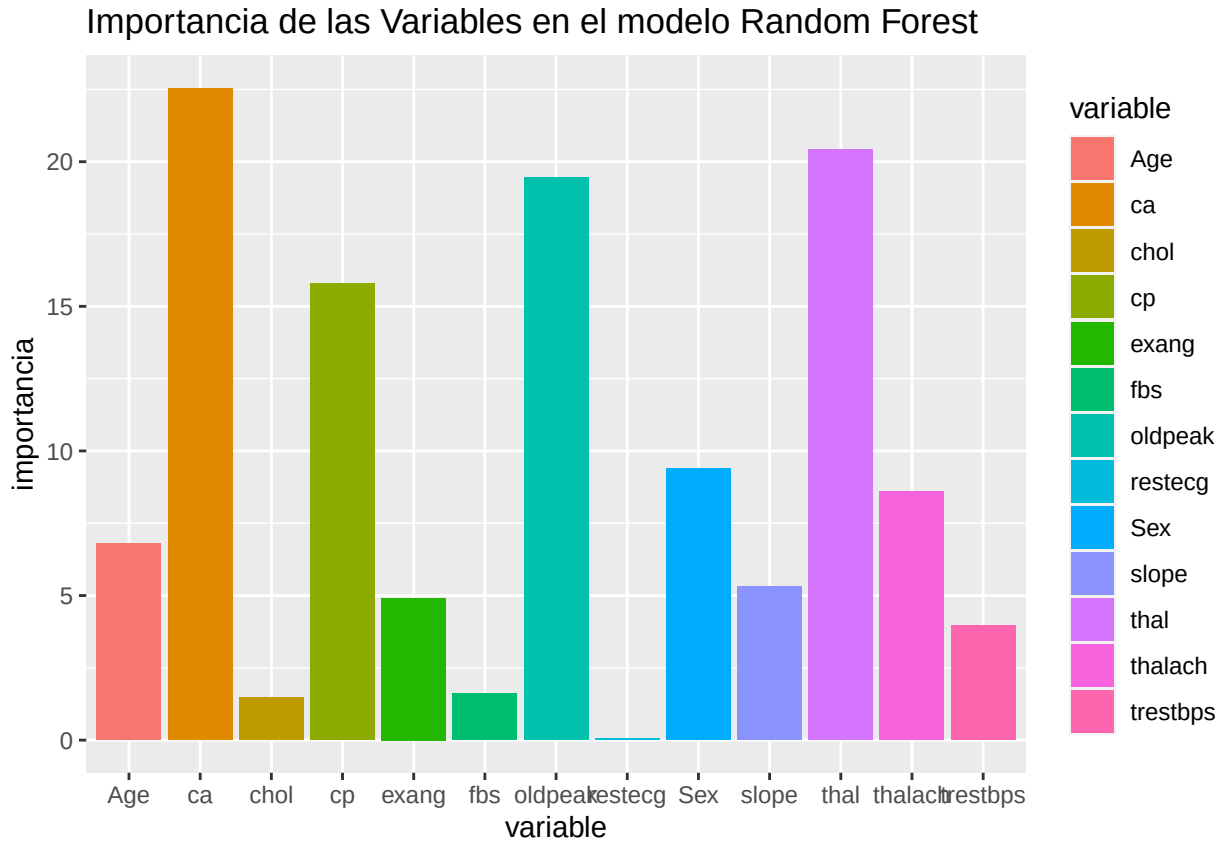
```
##          MeanDecreaseAccuracy
## Age                 6.7931137
## Sex                 9.3904779
## cp                 15.7831163
## trestbps            3.9584098
## chol                1.4782912
## fbs                 1.6151610
## restecg             0.0580079
## thalach             8.5899669
## exang               4.9134559
## oldpeak            19.4673378
## slope               5.3168134
## ca                 22.5436721
## thal               20.4133676
```

Representamos los valores de la tabla anterior en el siguiente gráfico. Las variables más significativas en el modelo de random forest son **ca**, **cp**, **oldpeak** y **thal**.

```
library(dplyr)
library(ggplot2)

data_importance <- tibble(variable=colnames(cleveland[,-14]),
                          importancia=importance(modelo.rf, type=1))
gg<- ggplot(data_importance, aes(x=variable, y=importancia, fill=variable))
gg<- gg + geom_col() + ggtitle("Importancia de las Variables en el modelo Random Forest")

gg
```



En términos del error en el set de testing, el modelo que mejor ajusta los datos es el **modelo.L1**.

Queremos ahora identificar el modelo que logre un mejor balance entre los indicadores de sensibilidad y especificidad. En el caso de la regresión logística, el umbral utilizado en la regla de Bayes para definir el diagnóstico fue 0.5, pero si las clases están muy desbalanceadas en el set de datos será conveniente analizar cuál es el umbral apropiado para la clasificación, en términos de optimizar los indicadores mencionados.

En el caso de random forest, obtenemos:

```
tabla.rf<-table(predicciones.rf, cleveland.V$diagnosis)
sensibilidad.rf<-tabla.rf[4]/(tabla.rf[4]+tabla.rf[3])
especificidad.rf<-tabla.rf[1]/(tabla.rf[1]+tabla.rf[2])

tabla.rf
```

```
##
## predicciones.rf  0  1
##               0 40 10
##               1  6 33
```

```
sensibilidad.rf
```

```
## [1] 0.7674419
```

```
especificidad.rf
```

```
## [1] 0.8695652
```

Veamos qué ocurre con la regresión:

```
tabla.L1<-table(predicciones.L1, cleveland.V$diagnosis)
sensibilidad.L1<-tabla.L1[4]/(tabla.L1[4]+tabla.L1[3])
especificidad.L1<-tabla.L1[1]/(tabla.L1[1]+tabla.L1[2])

tabla.L1
```

```
##
## predicciones.L1  0  1
##               0 41  8
##               1  5 35
```

```
sensibilidad.L1
```

```
## [1] 0.8139535
```

```
especificidad.L1
```

```
## [1] 0.8913043
```

Evaluamos la sensibilidad y la especificidad del modelo.L1 sobre una grilla de umbrales. El mejor umbral será el que maximice *simultaneamente* ambos indicadores. Luego de este proceso, obtenemos que el mejor punto de corte (en esa grilla) para la regla de Bayes es 0.48.

```
especificidad<-function(umbral){
  clasificador<-rep(0, dim(cleveland.V)[1])
  clasificador[probabilidades > umbral]<-1
  clasificador<-as.factor(clasificador)
  tabla<-table(clasificador, cleveland.V$diagnosis)
  return(tabla[1]/(tabla[1]+tabla[2]))
}


sensibilidad<-function(umbral){
  clasificador<-rep(0, dim(cleveland.V)[1])
  clasificador[probabilidades > umbral]<-1
  #print(clasificador)
  clasificador<-as.factor(clasificador)
  tabla<-table(clasificador, cleveland.V$diagnosis)
  #print(tabla[4]/(tabla[4]+tabla[3]))
  return(tabla[4]/(tabla[4]+tabla[3]))
}


distancias<-function(probas){
  distancias<-c()
  for (p in 1:length(probas)) {
    distancia<-sqrt(sum((c(1-especificidad(probas[p]), sensibilidad(probas[p]))-c(0,1))^2))
    distancias[p]<-distancia
  }
  return(distancias)
}

grilla<-seq(from=0.05, to=0.95, by=0.01)

#Devuelve algunos valores ausentes
#(provienen de tablas de una sola fila) pero no molestan
distancias<-map_dbl(grilla, distancias)
umbral<-grilla[which.min(distancias)]
```

```
umbral
```

```
## [1] 0.48
```

Podemos concluir que el mejor modelo, en términos del error y del balance obtenido entre especificidad y sensibilidad, es el modelo.L1.