Trabajo Final: Programación

Gema Correa Fernández y Ana Puertas Olea

7 de Junio de 2017

Ajuste de Modelos No Lineales

Antes de nada, debemos cargar las siguientes librerías que usaremos a lo largo de la práctica:

```
library("caret", quietly=TRUE) # Para usar preProcess
library("ggplot2", quietly=TRUE) # Para usar plot
library("e1071", quietly=TRUE) # Para usar SVM (Máquina de Soporte de Vectores)
library("ada", quietly=TRUE) # Para usar AdaBoost
library("nnet", quietly=TRUE) # Para usar nnet (Red Neuronal)
library("chemometrics", quietly=TRUE) # Parar usar nnetEval (NN by cross-validation)
library("ROCR", quietly=TRUE) # Para hacer la Curva ROC
```

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

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

```
library("randomForest", quietly=TRUE) # Para usar Random Forest
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

En este trabajo, nos vamos a centrar en el ajuste del mejor predictor (lineal o no lineal) para un conjunto de datos. En concreto, usaremos la BBDD Breast Cancer Wisconsin (Diagnostic) para un problema de clasificación binaria. Las características del data.frame han sido calculadas a partir de una imagen digitalizada de una extracción con aguja fina (FNA) de una masa mamaria. Por lo tanto, describen las características de los núcleos celulares presentes en la imagen.

Información del atributo:

1. **ID number**
2. **Diagnóstico** (M = maligno, B = benigno)

- Del atributo 3 al atributo 32, calculamos para cada núcleo de célula, diez características de valor real:

1. **Radio** (media de las distancias desde el centro hasta los puntos del perímetro)
2. **Textura** (desviación estándar de los valores de escala de grises)
3. **Perímetro**
4. **Área**
5. **Suavidad** (variación local en longitudes de radio)
6. **Compacidad** ($perímetro^2/área - 1.0$)
7. **Concavidad** (gravedad de las partes cóncavas del contorno)

8. **Puntos cóncavos** (número de partes cóncavas del contorno)
9. **Simetría**
10. **Dimensión fractal**

Nuestro objetivo será buscar el mejor modelo y justificar si son representativos dichos resultados. Al ser un problema de clasificación, clasificaremos en función de la segunda columna **diagnóstico**, que nos dice si el cáncer es maligno o benigno.

Para realizar una comparación entre todos los modelos, usaremos la curva ROC.

A continuación, vamos a leer nuestro data.frame:

```
# Guardamos el data frame en una varible
BWcancer <- read.csv("datos/wdbc.data", header = F, sep = ",")

# Visualizamos la cabecera del data frame
head(BWcancer)
```

```
##         V1 V2    V3    V4     V5     V6      V7      V8     V9     V10
## 1   842302  M 17.99 10.38 122.80 1001.0 0.11840 0.27760 0.3001 0.14710
## 2   842517  M 20.57 17.77 132.90 1326.0 0.08474 0.07864 0.0869 0.07017
## 3 84300903  M 19.69 21.25 130.00 1203.0 0.10960 0.15990 0.1974 0.12790
## 4 84348301  M 11.42 20.38  77.58  386.1 0.14250 0.28390 0.2414 0.10520
## 5 84358402  M 20.29 14.34 135.10 1297.0 0.10030 0.13280 0.1980 0.10430
## 6   843786  M 12.45 15.70  82.57  477.1 0.12780 0.17000 0.1578 0.08089
##      V11     V12    V13    V14   V15    V16      V17     V18     V19
## 1 0.2419 0.07871 1.0950 0.9053 8.589 153.40 0.006399 0.04904 0.05373
## 2 0.1812 0.05667 0.5435 0.7339 3.398  74.08 0.005225 0.01308 0.01860
## 3 0.2069 0.05999 0.7456 0.7869 4.585  94.03 0.006150 0.04006 0.03832
## 4 0.2597 0.09744 0.4956 1.1560 3.445  27.23 0.009110 0.07458 0.05661
## 5 0.1809 0.05883 0.7572 0.7813 5.438  94.44 0.011490 0.02461 0.05688
## 6 0.2087 0.07613 0.3345 0.8902 2.217  27.19 0.007510 0.03345 0.03672
##       V20     V21      V22   V23   V24    V25    V26    V27    V28    V29
## 1 0.01587 0.03003 0.006193 25.38 17.33 184.60 2019.0 0.1622 0.6656 0.7119
## 2 0.01340 0.01389 0.003532 24.99 23.41 158.80 1956.0 0.1238 0.1866 0.2416
## 3 0.02058 0.02250 0.004571 23.57 25.53 152.50 1709.0 0.1444 0.4245 0.4504
## 4 0.01867 0.05963 0.009208 14.91 26.50  98.87  567.7 0.2098 0.8663 0.6869
## 5 0.01885 0.01756 0.005115 22.54 16.67 152.20 1575.0 0.1374 0.2050 0.4000
## 6 0.01137 0.02165 0.005082 15.47 23.75 103.40  741.6 0.1791 0.5249 0.5355
##      V30    V31     V32
## 1 0.2654 0.4601 0.11890
## 2 0.1860 0.2750 0.08902
## 3 0.2430 0.3613 0.08758
## 4 0.2575 0.6638 0.17300
## 5 0.1625 0.2364 0.07678
## 6 0.1741 0.3985 0.12440
```

```
# Mostramos la dimensión (569 instancias y 32 características)
dim(BWcancer)
```

```
## [1] 569  32
```

Como sabemos **V1** se corresponde con un identificador, así que prescindimos de ella. Por consiguiente, el nombre de cada columna ha variado, por lo que hacemos una modificación para poder realizar un mejor tratamiento a los datos. Por ello, establecemos la variable de clasificación en la primera columna llamada **class**, y a partir de ahí iremos contando atributos.

```r
# Eliminamos la primera columna
BWcancer <- BWcancer[,-1]

# Modificamos el nombre de los atributo
names(BWcancer) = c("class", 1:30)

# Visualizamos la cabecera del data frame
head(BWcancer)
```

```
##   class     1     2      3      4       5       6      7       8      9
## 1     M 17.99 10.38 122.80 1001.0 0.11840 0.27760 0.3001 0.14710 0.2419
## 2     M 20.57 17.77 132.90 1326.0 0.08474 0.07864 0.0869 0.07017 0.1812
## 3     M 19.69 21.25 130.00 1203.0 0.10960 0.15990 0.1974 0.12790 0.2069
## 4     M 11.42 20.38  77.58  386.1 0.14250 0.28390 0.2414 0.10520 0.2597
## 5     M 20.29 14.34 135.10 1297.0 0.10030 0.13280 0.1980 0.10430 0.1809
## 6     M 12.45 15.70  82.57  477.1 0.12780 0.17000 0.1578 0.08089 0.2087
##         10     11     12    13     14       15      16      17      18
## 1 0.07871 1.0950 0.9053 8.589 153.40 0.006399 0.04904 0.05373 0.01587
## 2 0.05667 0.5435 0.7339 3.398  74.08 0.005225 0.01308 0.01860 0.01340
## 3 0.05999 0.7456 0.7869 4.585  94.03 0.006150 0.04006 0.03832 0.02058
## 4 0.09744 0.4956 1.1560 3.445  27.23 0.009110 0.07458 0.05661 0.01867
## 5 0.05883 0.7572 0.7813 5.438  94.44 0.011490 0.02461 0.05688 0.01885
## 6 0.07613 0.3345 0.8902 2.217  27.19 0.007510 0.03345 0.03672 0.01137
##         19       20    21    22     23     24     25     26     27     28
## 1 0.03003 0.006193 25.38 17.33 184.60 2019.0 0.1622 0.6656 0.7119 0.2654
## 2 0.01389 0.003532 24.99 23.41 158.80 1956.0 0.1238 0.1866 0.2416 0.1860
## 3 0.02250 0.004571 23.57 25.53 152.50 1709.0 0.1444 0.4245 0.4504 0.2430
## 4 0.05963 0.009208 14.91 26.50  98.87  567.7 0.2098 0.8663 0.6869 0.2575
## 5 0.01756 0.005115 22.54 16.67 152.20 1575.0 0.1374 0.2050 0.4000 0.1625
## 6 0.02165 0.005082 15.47 23.75 103.40  741.6 0.1791 0.5249 0.5355 0.1741
##       29      30
## 1 0.4601 0.11890
## 2 0.2750 0.08902
## 3 0.3613 0.08758
## 4 0.6638 0.17300
## 5 0.2364 0.07678
## 6 0.3985 0.12440
```

Como podemos comprobar, nuestra variable de clasificación es una variable que toma valores entre M o B, por tanto, la vamos a poner en función de 0 y 1 para poder trabajar correctamente con ella.

- 0: Benigno
- 1: Maligno

```r
# Cambiamos a numérico la variable diágnostico
BWcancer[,1] <- as.numeric(BWcancer[,1]) # Obtenemos un 2 en Menigno y un 1 para Benigno
BWcancer[,1][BWcancer[,1] == 1] = 0 # Cambiamos el 1 por el 0 (Benigno) --> No padece cáncer
BWcancer[,1][BWcancer[,1] == 2] = 1 # Cambiamos el 2 por el 1 (Maligno) --> Padece cáncer

# Por último, transformamos en factores nuestra variable class
BWcancer$class = as.factor(BWcancer$class)
```

Antes de transformar los atributos debemos comprobar el rango de cada columna, ya que el método *BoxCox* no funciona para variables que son ceros o negativos, por tanto, en las variables que tengan ceros, haremos una traslación sumando uno a todos los valores para poder realizar la trasformación que mitiga la asimetría.

```r
# Contamos desde la segunda columna hasta el final, ya que la primera es nuestra 'class'
for(i in 2:ncol(BWcancer)) {
  # Obtenemos el mínimo
  if (min(BWcancer[i]) == 0)
    cat(i-1, "\t")
}
```

```
## 7    8   17  18  27  28
```

Por tanto, a las variables "7", "8", "17", "18", "27", "27" debemos sumarle un uno. Esta modificación, no afectará a los resultados y realizamos tal cambio, a todo el cojunto de datos de ese atributo.

```r
# Sumamos uno en las siguientes atributos
BWcancer$"7" = (BWcancer$"7" + 1)
BWcancer$"8" = (BWcancer$"8" + 1)
BWcancer$"17" = (BWcancer$"17" + 1)
BWcancer$"18" = (BWcancer$"18" + 1)
BWcancer$"27" = (BWcancer$"27" + 1)
BWcancer$"28" = (BWcancer$"28" + 1)
```

Ahora, vamos hacer un particionado de los datos. Como se nos proporciona solo un archivo, somos nosotros quiénes tenemos que hacer tal partición. Para ello, vamos a dividir el data.frame en un 80% para el training y en un 20% para el test.

```r
# Establecemos una semilla por defecto para hacer el mismo tipo de partición
set.seed(127)

# Nos quedamos con los indices para el training
train = sample (nrow(BWcancer), round(nrow(BWcancer)*0.8))

# Reservamos por separado training y test
BWcancer.train = BWcancer[train,]
BWcancer.test = BWcancer[-train,]
```

Ya podemos proceder a transformar los datos usando *preProcess()*. Para ello establecemos el umbral del PCA a 0.9. Para realizar este cambio, hemos seguido el mismo procedimiento del documento *paraTrabajo3.pdf* disponible en DECSAI.

Aplicaremos el preprocesamiento a los datos del training y del test.

```r
# Establecemos una semilla por defecto
set.seed(127)

# Transformamos los datos
resultado_preprocesamiento = preProcess(BWcancer.train,
                                        method = c("BoxCox", "center", "scale", "pca"),
                                        thres = 0.9)

# Aplicar el preprocesamiento a los datos del training y del test
BW_cancer_train_preprocesado = predict(resultado_preprocesamiento, BWcancer.train)
BW_cancer_test_preprocesado = predict(resultado_preprocesamiento, BWcancer.test)
```

```r
attach(BWcancer.train) # Para prescendir y simplificar el prefijo BWcancer.train
```

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Puesto que consideramos que tenemos suficientes datos, optaremos por no realizar validación cruzada (excepto

para la red neuronal). También, debemos tener en cuenta que no vamos a penalizar ni los falsos positivos o ni los falsos negativos, es decir, ambos tendrán el mismo valor.

Una vez arreglados nuestros datos, ya podemos pasar a la selección del modelo. Para empezar a seleccionar el mejor modelo, nos basaremos en la gráfica de la curva ROC, ofrecida por el modelo y en su área bajo la curva. Por consiguiente, no calcularemos E_{out}, ya que no nos basaremos en el error para realizar la comparación entre los modelos.

Modelo Lineal

GLM (Regresión Logística)

Previamente, ajustaremos un modelo lineal de regresión logística sobre los datos, para ver si es mejor predictor que un modelo no lineal.

```
# Función que calcula los errores y E_test para regresión logística
errorres_regresion_logistica <- function(m){

  probTr = predict(m, type="response")
  probTst = predict(m, data.frame(BW_cancer_test_preprocesado), type="response")

  predTst = rep(0, length(probTst)) # predicciones por defecto 0
  predTst[probTst >= 0.5] = 1 # >= 0.5 clase 1

  predTr = rep(0, length(probTr)) # predicciones por defecto 0
  predTr[probTr >= 0.5] = 1 # >= 0.5 clase 1

  # Para el calculo del Etest
  print(table(pred=predTst, real=BW_cancer_test_preprocesado$class))

  # Calculamos Etest
  Etest = mean(predTst != BW_cancer_test_preprocesado$class)

  list(Etest=Etest)
}
```

```
# Creamos el modelo
ml = glm(class ~ ., family = binomial(logit), data = BW_cancer_train_preprocesado)

# Realizamos un análisis del modelo
summary(ml)
```

```
##
## Call:
## glm(formula = class ~ ., family = binomial(logit), data = BW_cancer_train_preprocesado)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2413  -0.0368  -0.0015   0.0035   3.8220
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.5642     0.5147  -4.982 6.29e-07 ***
## PC1          -2.5758     0.4296  -5.997 2.02e-09 ***
## PC2           1.6439     0.3360   4.893 9.92e-07 ***
```

```
## PC3           0.3392     0.1861   1.822  0.06839 .
## PC4           1.1658     0.2653   4.395 1.11e-05 ***
## PC5          -1.1440     0.3879  -2.949  0.00319 **
## PC6           0.9352     0.3392   2.757  0.00584 **
## PC7           0.4523     0.3781   1.196  0.23162
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 600.338  on 454  degrees of freedom
## Residual deviance:  69.852  on 447  degrees of freedom
## AIC: 85.852
##
## Number of Fisher Scoring iterations: 9
```

```
# Obtenemos el error
errorres_regresion_logistica(ml)
```

```
##     real
## pred  0  1
##    0 70  1
##    1  1 42
```

```
## $Etest
## [1] 0.01754386
```

Como podemos apreciar, el E_{test} nos devuelve un valor de 0.0175439, un error aceptable pero que posiblemente podamos reducir aplicando sobre los datos otros modelos que veremos más adelante.

Además, nuestra matriz de confusión nos muestra que se producen dos fallos, un falso positivo y un falso negativo. Por eso, nuestro error se puede considerar tolerable, ya que no es excesivamente grande.

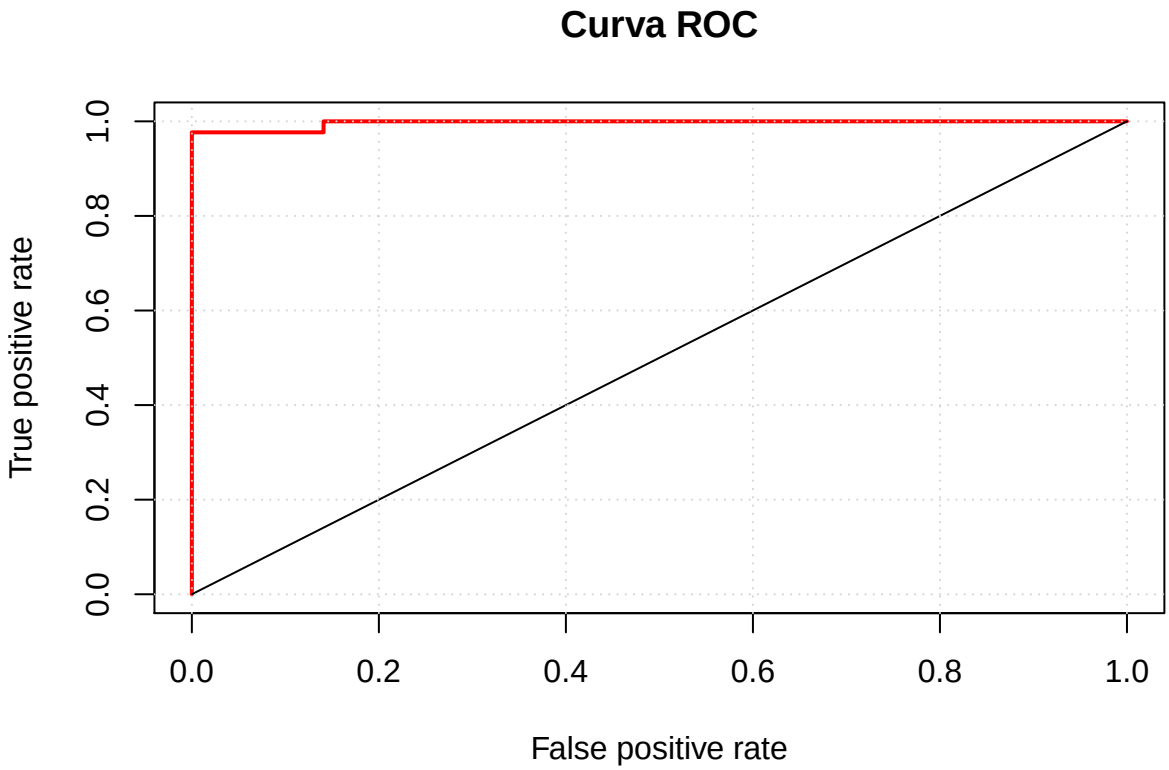
Según el resultado obtenido, podemos ver que las variables más relevantes son *PC1*, *PC2*, *PC4*, *PC5* y *PC6*, así que cuanto mayor sea el valor absoluto de $Z - value$ más información nos aportará dicha variable, ya que representa la distancia entre el dato y la media de la población.

A continuación, realizamos la curva ROC para el modelo lineal. Primero deberemos obtener la probabilidad de nuestra predicción.

```
# Obtenemos las probabilidades
prob_GLM = predict(ml, data.frame(BW_cancer_test_preprocesado), type=c("response"))
```

Para mecanizar el proceso, nos creamos una función que nos dibuja la curva ROC.

```
# Función que dibuja uan curva ROC
plotROC <- function(modelo, etiq_real, adicionar=FALSE,color="red") {
  pred <- prediction(modelo, etiq_real)
  perf <- performance(pred,"tpr","fpr")
  plot(perf, col=color, add=adicionar, main="Curva ROC", lwd = 2)
  segments(0, 0, 1, 1, col='black')
  grid()
}

# Cruva ROC para el modelo lineal
plotROC(prob_GLM, BW_cancer_test_preprocesado$class)
```




```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Modelo No Lineal

Máquina de Soporte de Vectores (SVM)

Usar solo el núcleo RBF-Gaussiano. Encontrar el mejor valor para el parámetro libre hasta una precisión de 2 cifras (enteras o decimales)

Antes de comenzar, debemos encontrar el mejor ajuste para **SVM** haciendo uso de un núcleo RBF-Gaussiano que especificaremos en la función mediante *kernel="radial"*.

```
# Establecemos una semilla
set.seed(127)

# Usamos tune.svm para encontrar el mejor coste y gamma para svm
x = subset(BW_cancer_train_preprocesado, select=-class)
svm_tune = tune.svm(x = x, y = class, kernel = "radial")
# print(svm_tune)

# Obtenemos el mejor modelo
svm_tune$best.model
```

```
##
## Call:
## best.svm(x = x, y = class, kernel = "radial")
##
##
## Parameters:
```

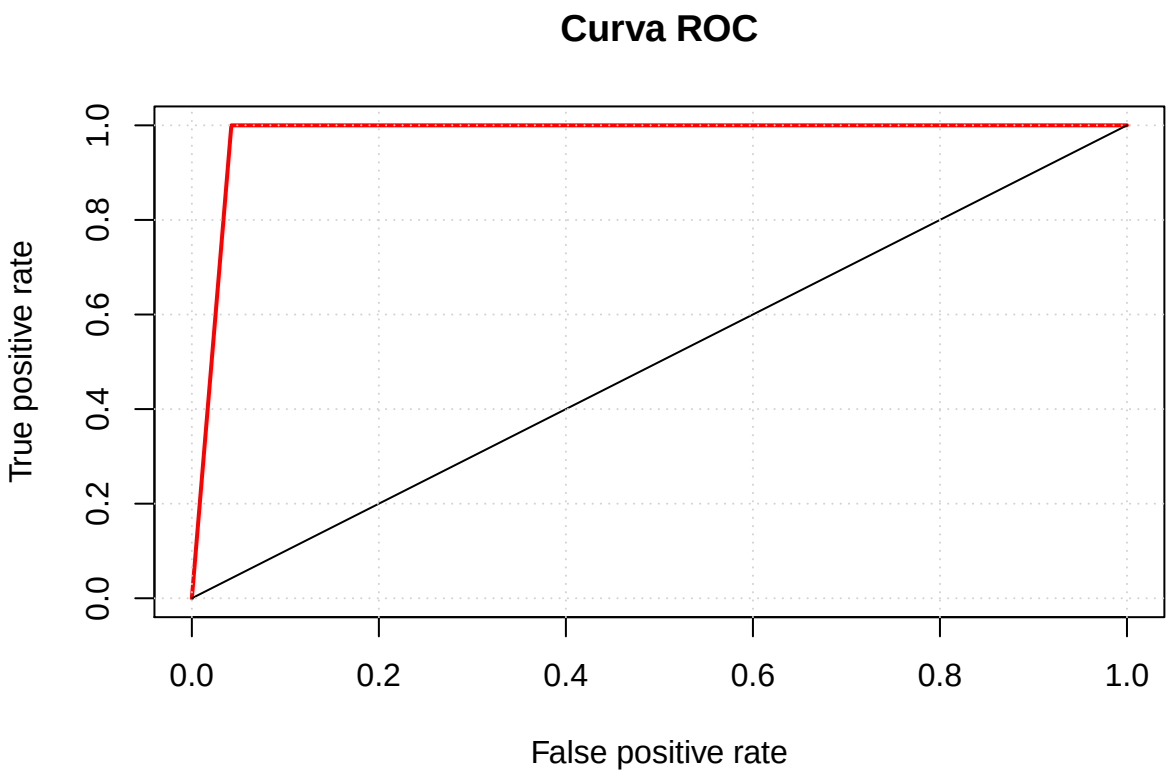
```
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.1428571
##
## Number of Support Vectors:  141
```

Ahora ya podemos calcular el modelo (*Machines Vector Support*) con los mejores parámetros encontrados, que son $cost = 1$ y $gamma = 0.1428571$. Además, el mejor resultado devuelve un número de 141 vectores soporte. Para ello, usamos la función implementada en R llamada *svm()* con los mejores parámetros encontrados.

```r
# Establecemos una semilla
set.seed(127)

# Guardamos el mejor gamma y cost
mejor_gamma = svm_tune$best.model$gamma
mejor_coste = svm_tune$best.model$cost

# Obtenemos el modelo
modelo_svm = svm(class ~., data = BW_cancer_train_preprocesado, kernel = "radial",
                 cost = mejor_coste, gamma = mejor_gamma, decision.values = TRUE,
                 probability = TRUE)

# Obtenemos la predición tanto para train como para test (Nos devuelve las etiquetas)
prediccion_train = predict(modelo_svm, data.frame(BW_cancer_train_preprocesado),
                           decision.values = TRUE, probability = TRUE)
prediccion_test = predict(modelo_svm, data.frame(BW_cancer_test_preprocesado),
                          decision.values = TRUE, probability = TRUE)

# Obtenemos nuestra matriz de confusión
table(pred=prediccion_test, real=BW_cancer_test_preprocesado$class)
```

```
##     real
## pred  0  1
##    0 69  0
##    1  2 43
```

```r
# Obtenemos la probabilidad entre las clases positivas y negativas
test_prob = attributes(predict(modelo_svm, data.frame(BW_cancer_test_preprocesado),
                               decision.values = T))$decision.values

# Como la probabilidad no se encuentra entre [0,1], vamos a normalizarla
# donde el mínimo seaa 0 y el máximo seaa 1
test_prob_normalizado = (test_prob - min(test_prob)) / ( max(test_prob) - min(test_prob) )
# summary(test_prob_normalizado)

# Ponemos que las clases que tengan el mismo peso
pred_test_SVM = rep(0, length(test_prob_normalizado)) # Predicciones por defecto 0
pred_test_SVM[test_prob_normalizado >= 0.5] = 1 # Clase 1 -> cáncer maligno

# Pintamos la curva ROC para SVM
plotROC(pred_test_SVM, BW_cancer_test_preprocesado$class)
```



Vamos a calcular el error E_{test} que nos proporciona el modelo SVM.

```
# Calculamos el error Etest
Etest_SVM = 0
Etest_SVM = mean(pred_test_SVM != BW_cancer_test_preprocesado$class)
cat("E_test para SVM", Etest_SVM)
```

```
## E_test para SVM 0.02631579
```

Como podemos apreciar, el modelo lineal previo ofrece un mejor resultado que el que nos ha devuelto SVM.

A continuación, veamos la matriz de confusión obtenida:

```
print(table(pred=pred_test_SVM, real=BW_cancer_test_preprocesado$class))
```

```
##     real
## pred  0  1
##    0 68  0
##    1  3 43
```

Como podemos observar, nuestra matriz de confusión nos muestra que se producen tres fallos, concretamente tres falsos negativos. Por eso, nuestro error es ligeramente más grande que el modelo anterior, pero aún así se puede considerar tolerable, ya que no es excesivamente grande.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Random Forest

Usar los valores que por defecto se dan en la teoría y experimentar para obtener el número de árboles adecuado.

En este modelo, tenemos que ajustar el número de árboles a utilizar en el algoritmo. Tras probar entre 1 y 200 árboles, el que proporciona mejor resultado de E_{test} es 15 árboles, siendo además es el modelo más pequeño posible de árboles.

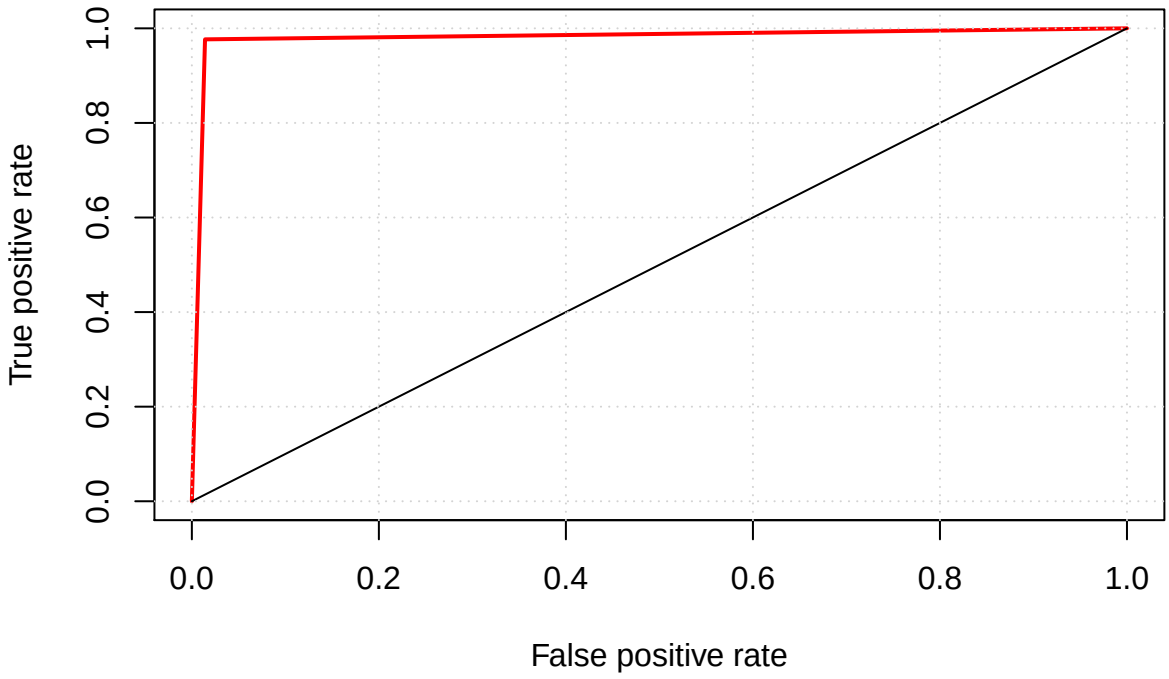
```
# Establecemos una semilla por defecto
set.seed(127)

# Obtenemos el modelo
modelo_rf = randomForest(class ~., data = BW_cancer_train_preprocesado, ntree=15)

# Realizamos las predicciones
prediccion_train = predict(modelo_rf, BW_cancer_train_preprocesado,type="prob")
prediccion_test = predict(modelo_rf, BW_cancer_test_preprocesado, type="prob")
RF <- prediccion_test[,2]

# Poner la probabilidad entre 0 y 1, ponemos que las clases que tengan el mismo peso
pred_test_RF = rep(0, length(RF)) # Predicciones por defecto 0
pred_test_RF[RF >= 0.5] = 1 # Clase 1 -> cáncer maligno

# Matriz de confusión
print(table(pred=pred_test_RF, real=BW_cancer_test_preprocesado$class))
```

```
##     real
## pred  0  1
##    0 70  1
##    1  1 42
```

```
# Calculamos el error
Etest_RF = 0
Etest_RF = mean(pred_test_RF != BW_cancer_test_preprocesado$class)
cat("Etest para Random Forest", Etest_RF)
```

```
## Etest para Random Forest 0.01754386
```

```
# Pintamos la curva ROC
plotROC(pred_test_RF,BW_cancer_test_preprocesado$class)
```



Con el algoritmo Random Forest, nuestro E_{test} es de 0.0175439. Como dijimos, el mejor número de árboles encontrado es de 15 tras la realización de varias pruebas, siendo el número más pequeño encontrado que ofrece el error más pequeño. Como podemos observar, el error es bastante parecido al obtenido con el modelo lineal. Además, hemos obtenido una mejora frente al modelo no lineal SVM.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Boosting

Para clasificación usar AdaBoost con funciones "stump".

Para implementar **AdaBoost** con funciones "stump"" nos hemos basado en el pdf *"Journal of Statistical Software", October 2006, Volume 17, Issue 2.*

```
# Establecemos una semilla por defecto
set.seed(127)

library(rpart) # Librería necesaria para usar la función stump

# Según la referencia anterior, es stump (2-split)
stump = rpart.control(cp = -1 , maxdepth = 1 , minsplit = 0)

# Obtenemos el modelo
# type = "discrete" realiza Boosting.
adaboost_stump <- ada(class~., data = BW_cancer_train_preprocesado, iter = 50,
                      loss = "e", type = "discrete", control = stump)

# Obtenemos las predicciones
prediccion_train = predict(adaboost_stump, BW_cancer_train_preprocesado)
```

```r
prediccion_test = predict(adaboost_stump, data.frame(BW_cancer_test_preprocesado) )

# Matriz de confusión
print(table(pred=prediccion_test, real=BW_cancer_test_preprocesado$class))
```

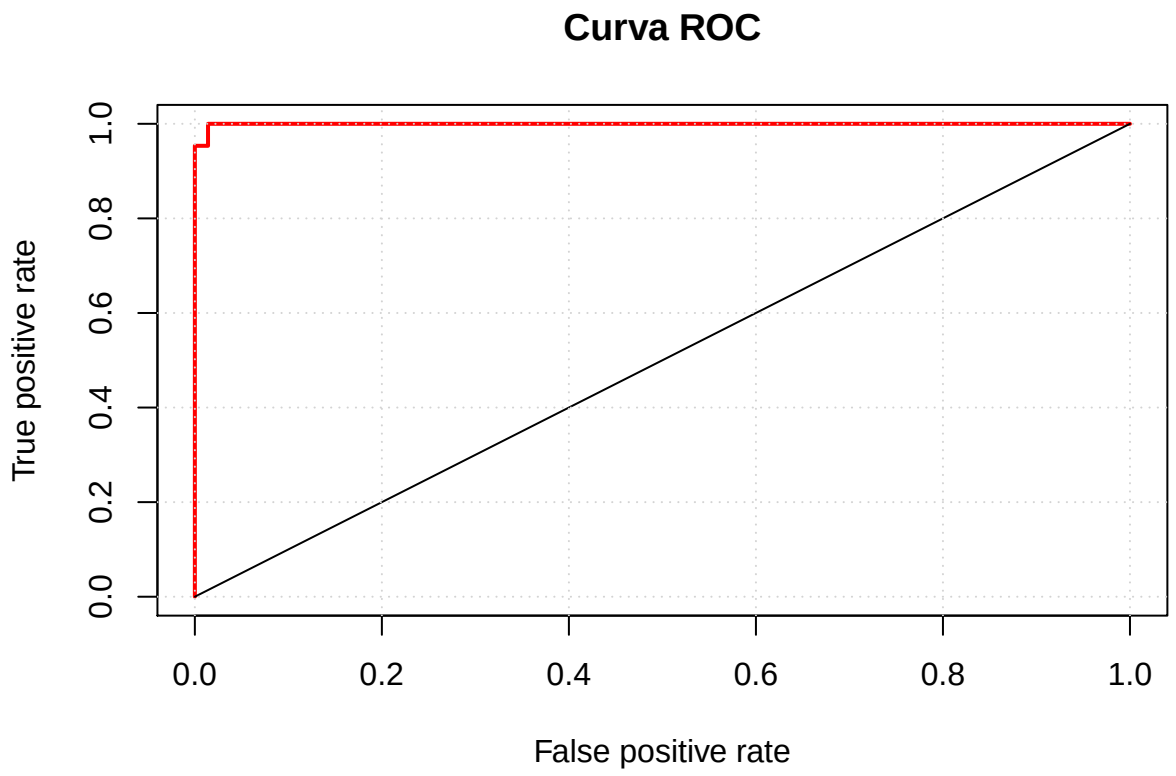
```
##     real
## pred  0  1
##    0 71  2
##    1  0 41
```

```r
# Calculamos el error Etest
Etest_ada = 0
Etest_ada = mean(prediccion_test != BW_cancer_test_preprocesado$class)
cat("Etest para AdaBoost", Etest_ada)
```

```
## Etest para AdaBoost 0.01754386
```

Pintamos la curva ROC, pero antes debemos obtener la probabilidad.

```r
# Obtenemos la probabilidad entre las clases positivas y negativas
test_prob_adab = predict(adaboost_stump, data.frame(BW_cancer_test_preprocesado),
                         type = "probs")
# Nos quedamos con una columna de test_prob_adab, da igual si es la 1 o la 2,
# ya que son la inversa respectivamente

# Dibujamos la curva ROC
plotROC(test_prob_adab[,2], BW_cancer_test_preprocesado$class)
```



Con el algoritmo AdaBoost, nuestro E_{test} es de 0.0175439, un error bastante aceptable que podemos comparar con el resultado obtenido con el modelo lineal.

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Redes Neuronales

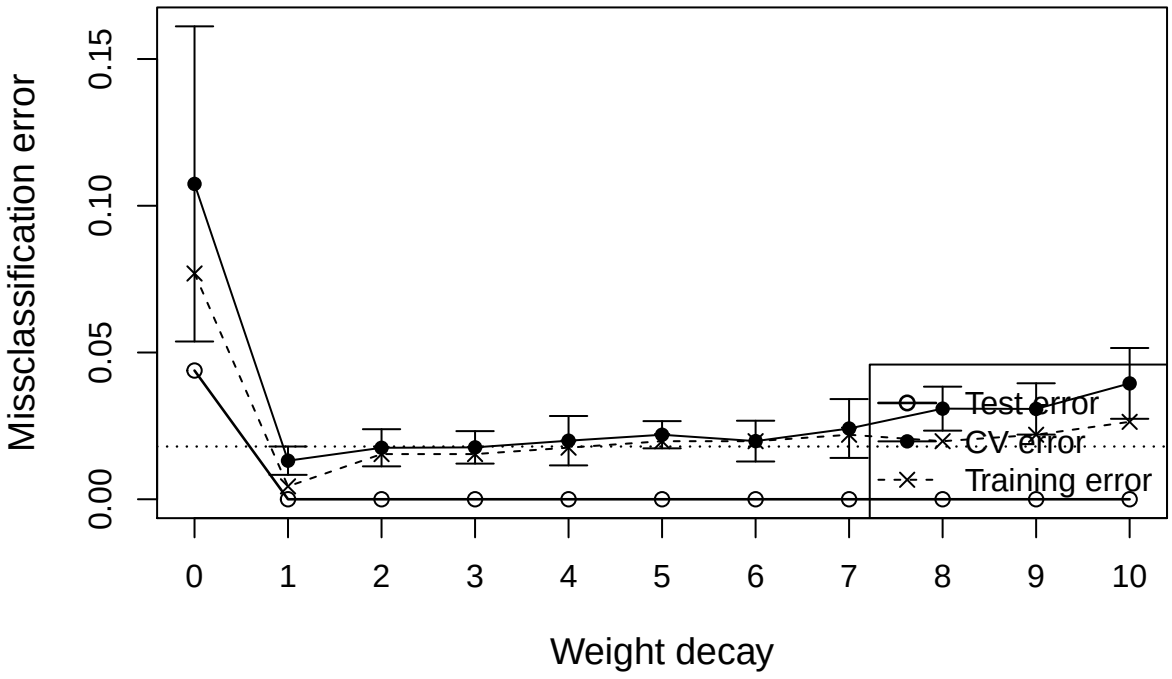
Considerar tres clases de funciones definidas por arquitecturas con 1, 2 y 3 capas de unidades ocultas y número de unidades por capa en el rango $0 - 50$. Definir un conjunto de modelos (arquitecturas) y elegir el mejor por validación cruzada. Recordar que a igualdad de E_{out} siempre es preferible la arquitectura más pequeña.

Para realizar una **Red Neuronal** mediante validación cruzada, existe una función *nnetEval()* que ya implementa eso, está disponible en la librería chemometrics

Para escoger el mejor modelo, haremos uso de la validación cruzada, trabajando con todos los datos y llevando a cabo el prepocesamiento sobre todo nuestro data.frame.

Tras probar distintos experimentos con un número diferente de unidades por capa entre el rango $[0, 50]$, obtuvimos que el mejor resultado de validación cruzada se consigue con 16 unidades ocultas (*size*).

```r
# Establecemos una semilla por defecto
set.seed(127)

# Transformamos los datos
resultado_preprocesamiento = preProcess(BWcancer, method = c("BoxCox", "center",
                                                             "scale", "pca"), thres = 0.9)

# Aplicar el preprocesamiento a los datos
BWcancer_preprocesado = predict(resultado_preprocesamiento, BWcancer)

# Nos quedamos con los indices para el training
train = sample (nrow(BWcancer), round(nrow(BWcancer)*0.8))

# Obtenemos nuestro modelo
# size -> number of hidden units
modelo_nn <- nnetEval(BWcancer_preprocesado, BWcancer_preprocesado$class, train,
                      kfold = 10, size = 16, maxit=100)
```




```
# Obtenemos el Eout
mean(modelo_nn$testerr)
```

```
## [1] 0.003987241
```

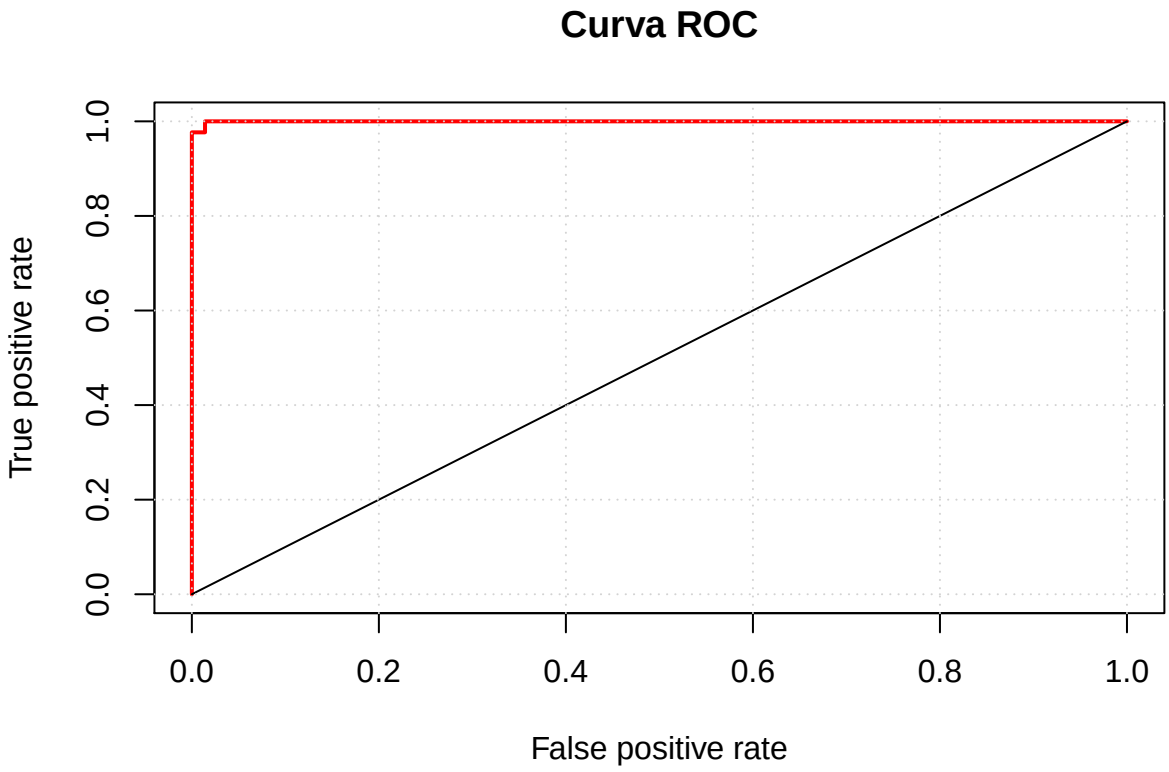
Sin lugar a duda, el modelo de Red Neuronal es el mejor predictor de todos con un error de 0.003987241.

A continuación, probaremos el modelo de red neuronal sin validación cruzada con el fin de obtener nuestra única curva ROC. Para ello hacemos uso de la función *nnet()*

```
# Establecemos una semilla por defecto
set.seed(127)

# Obtenemos el modelo
nn <- nnet(class ~., BW_cancer_train_preprocesado,size = 16, maxit=100)
```

```
## # weights:  145
## initial  value 381.780704
## iter  10 value 31.302964
## iter  20 value 5.144941
## iter  30 value 0.071842
## iter  40 value 0.002559
## final  value 0.000098
## converged
```

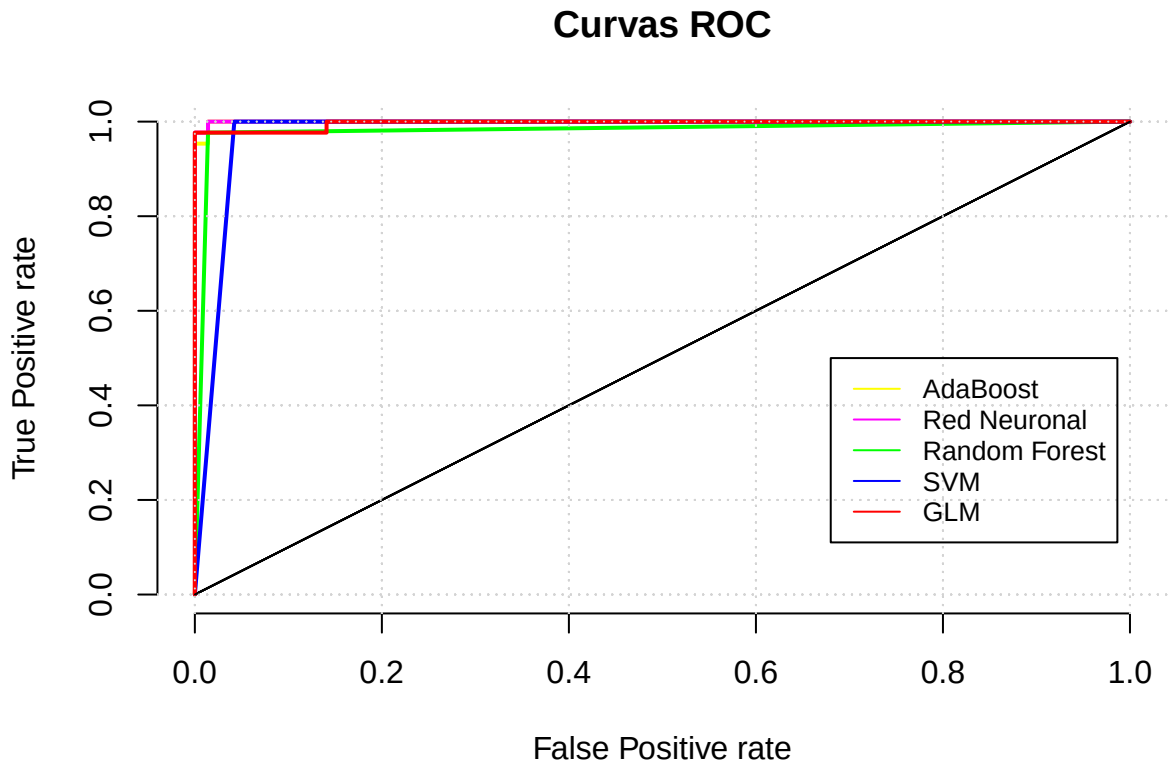
```
# test_prob_nn --> ya nos devuelve la probabilidad (nos quedamos con la primera columna)
test_prob_nn <- predict(nn,BW_cancer_test_preprocesado)

# Mostramos la Curva ROC
plotROC(test_prob_nn[,1], BW_cancer_test_preprocesado$class)
```



Después de haber estudiado por separado cada modelo, vamos a realizar un estudio comparando todas a la vez. Lo primero que haremos será mostrar una gráfica donde estén todas las curvas ROC.

```
# Para que nos muestre las gráficas juntas, abrimos plot antes
plot.new()
plot.window(xlim=c(0,1), ylim=c(0,1))
axis(1)
axis(2)
title(main = "Curvas ROC")
title(xlab= "False Positive rate")
title(ylab= "True Positive rate")

# Boosting
plotROC(test_prob_adab[,2], BW_cancer_test_preprocesado$class, color="yellow",
        adicionar = TRUE)
# Redes Neuroanles
plotROC(test_prob_nn[,1],BW_cancer_test_preprocesado$class, color = "magenta",
        adicionar = TRUE)
# RandomForest
plotROC(pred_test_RF,BW_cancer_test_preprocesado$class, color = "green",
        adicionar = TRUE)
# SVM
plotROC(pred_test_SVM,BW_cancer_test_preprocesado$class, color = "blue",
        adicionar = TRUE)
# GLM
plotROC(prob_GLM,BW_cancer_test_preprocesado$class, color = "red",
        adicionar = TRUE)


# Agregamos una leyenda
```

```
legend(0.68, 0.5, c("AdaBoost", "Red Neuronal",  "Random Forest", "SVM", "GLM"),
       cex = 0.8, col = c("yellow", "magenta", "green", "blue",  "red"),
       lty=c(1,1))
```




```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Como a simple vista no podemos distinguir con claridad qué modelo tiene mayor área bajo la curva, realizaremos una función para que nos diga cual es el mejor predictor, calculando el área bajo la curva de cada modelo.

```
# Función que calcula el área bajo la curva
areaROC <- function(prediccion, etiq_real) {
  pred <- prediction(prediccion, etiq_real)
  auc <- performance(pred,"auc")
  attributes(auc)$y.values[[1]]
}

# Guardamos los valores obtenidos
Area <- data.frame(
  modelos = c("AdaBoost","Red Neuronal","Random Forest", "SVM ","GLM"),
  valor =  c(areaROC(test_prob_adab[,2], BW_cancer_test_preprocesado$class),
             areaROC(pred_test_RF,BW_cancer_test_preprocesado$class),
             areaROC(pred_test_SVM,BW_cancer_test_preprocesado$class),
             areaROC(prob_GLM,BW_cancer_test_preprocesado$class) ) )
Area
```

```
##         modelos     valor
## 1      AdaBoost 0.9993449
## 2  Red Neuronal 0.9996725
## 3 Random Forest 0.9813298
```

```
## 4          SVM  0.9788732
## 5           GLM 0.9967245
```

Como era de esperar, el mejor resultado obtenido ha sido el de Red Neuronal, seguido de AdaBoost, GLM, Random Forest y por último, SVM. Por lo tanto, para nuestro problema de clasificación la mejor opción es sin duda este modelo no lineal de Red Neuronal, donde podemos obtener una mejor predicción de si la persona va a padecer o no cáncer de mama. Además, todos los modelos realizados obtienen un resultado aceptable, ya que el error que producen no es excesivamente alto, pero a igualdad de errores nos decantaremos por el modelo que dé menos falsos negativos, ya que consideramos que es peor que una persona que padezca cáncer sea diagnosticada como sana (la esperanza de vida de dicha persona sería prácticamente nula si no se le aplicara ningún tratamiento) que si en su lugar, la persona se encuentra sana y se le diagnostica esta enfermedad, seleccionando por lo tanto, los modelos que cumplen dichas condiciones.