PEC 3: Predicción de secuencias promotoras en E.coli

María Ajenjo Bauzá

11/1/2021

Contents

1 Lectura y transformación de los datos

Se realiza la lectura de los datos.

```
promotores <- read.csv(file = file.path(params$folder.data, params$file.data),
                       header = FALSE,
                       col.names = c("class", "name_seq", "sequence"))
head(promotores)
```

```
  class  name_seq                                                 sequence
1     +       S10 tactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcgt
2     +      AMPC tgctatcctgacagttgtcacgctgattggtgtcgttacaatctaacgcatcgccaa
3     +      AROH gtactagagaactagtgcattagcttatttttttgttatcatgctaaccacccggcg
4     +     DEOP2 aattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaatactaacaaactc
5     + LEU1_TRNA tcgataattaactattgacgaaaagctgaaaaccactagaatgcgcctccgtggtag
6     +    MALEFG aggggcaaggaggatggaaagaggttgccgtataaagaaactagagtccgtttaggt
```

1.1 Codificación one-hot

En primer lugar, se creará un diccionario en el que las claves sean los 4 nucleótidos posibles y los valores serán las representaciones en 1 y 0 correspondientes a cada uno.

```
# Se crea el diccionario
dict_nucl <- hash()
# Se añaden los valores
dict_nucl['t'] <- c(1,0,0,0)
dict_nucl['c'] <- c(0,1,0,0)
dict_nucl['g'] <- c(0,0,1,0)
dict_nucl['a'] <- c(0,0,0,1)
dict_nucl
```

```
<hash> containing 4 key-value pair(s).
  a : 0 0 0 1
  c : 0 1 0 0
  g : 0 0 1 0
  t : 1 0 0 0
```

Posteriormente, se creará una función que tome como argumento el diccionario creado anteriormente y el fichero en el que se encuentren los promotores que queramos codificar.

```
# Argumentos: diccionario nucleótidos y fichero con seq a codificar (tabla)
onehot_encoding <- function(dict_nucleotides, file){
  # Se crea una tabla vacía con 228 columnas (57*4)
  nucl_table <- matrix(ncol = 228)
  # Para cada fila de la columna 3 del fichero
  for (row in 1:length(file[,3])){
    # La secuencia será el valor de esa fila y la columna 3
    seq = file[row,3]
    # Se crea una lista vacía donde añadir el resultado de la codificación
    lista <- c()
    # Se parte la secuencia en sus diferentes nucleótidos
```

```
    prom_split <- strsplit(seq,"")[[1]]
    # Para cada nucleótido de los de la secuencia
    for (nucl in prom_split){
      # Se añade a la lista el resultado de la codificación
      lista = append(lista, dict_nucleotides[[nucl]])
    }
    # Se añade a la tabla el resultado de la codificación
    nucl_table <- rbind(nucl_table, lista)
  }
  # Se elimina la primera fila vacía de la matriz nueva para que no aparezcan
  # los NA que aparecían al crear la matriz
  nucl_table <- nucl_table[-1,]
  # Se pasa de matriz a data.frame
  nucl_table <- as.data.frame(nucl_table)
  # Se juntan la nueva tabla y el fichero
  file <- cbind(file, nucl_table)
}
```

Ahora utilizaremos la función para realizar la codificación one-hot de nuestros promotores.

```
prom_oh <- onehot_encoding(dict_nucl, promotores)
head(prom_oh)[1:6]
```

```
        class  name_seq
lista       +       S10
lista.1     +      AMPC
lista.2     +      AROH
lista.3     +     DEOP2
lista.4     + LEU1_TRNA
lista.5     +    MALEFG
                                                        sequence V1 V2 V3
lista   tactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcgt  1  0  0
lista.1 tgctatcctgacagttgtcacgctgattggtgtcgttacaatctaacgcatcgccaa  1  0  0
lista.2 gtactagagaactagtgcattagcttatttttttgttatcatgctaaccacccggcg  0  0  1
lista.3 aattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaatactaacaaactc  0  0  0
lista.4 tcgataattaactattgacgaaaagctgaaaaccactagaatgcgcctccgtggtag  1  0  0
lista.5 aggggcaaggaggatggaaagaggttgccgtataaagaaactagagtccgtttaggt  0  0  0
```

```
# Comprobamos que las dimensiones sean correctas
dim(prom_oh)
```

```
[1] 105 231
```

1.2 Separación de los datos en train y test

En primer lugar, realizamos algunas modificaciones pertinentes en nuestro dataset.

```
# Eliminamos las columnas 2 y 3 correspondientes al nombre y a la secuencia
prom_oh <- prom_oh[c(-2,-3)]

# Pasamos a factor la variable class
prom_oh$class <- factor(prom_oh$class)
```

Ahora procedemos a la separación en train y test.

```
# n es el número de filas del conjunto total de datos
n <- nrow(prom_oh)

# Fijamos la semilla de aleatoriedad
set.seed(params$seed.train)

# Dividimos los datos en train y test
# n_train = 2/3 = params$p.train
train <- sample(n,floor(n*params$p.train))
data_train <- prom_oh[train,]
data_test  <- prom_oh[-train,]

# Comprobamos que los datos se han partido bien
dim(data_train)
```

```
[1]  70 229
```

```
dim(data_test)
```

```
[1]  35 229
```

```
head(data_train)[1:6]
```

```
         class V1 V2 V3 V4 V5
lista.30     +  0  1  0  0  1
lista.78     -  1  0  0  0  0
lista.50     +  0  1  0  0  0
lista.13     +  0  1  0  0  1
lista.66     -  1  0  0  0  1
lista.41     +  0  0  0  1  0
```

Ahora tenemos las columnas de las secuencias codificadas junto a la columna "class", que muestra la clase (si son secuencias promotoras, +, o no). Pero en algunos algoritmos debemos tener los datos sin esta columna, por lo que prepararemos los datasets necesarios para esos algoritmos. Necesitaremos también unas variables donde almacenar estos valores.

```
# Variables con las clases de los datos
data_train_labels <- data_train$class
data_test_labels <- data_test$class

# Datasets sin etiquetar
data_train_nolab <- data_train[-1]
data_test_nolab <- data_test[-1]
```

2 Aplicación de diferentes algoritmos para la clasificación de datos de secuencias promotoras

2.1 k-Nearest Neighbour (k-NN)

2.1.1 Entrenamiento del modelo

k = 1

```
k = 1
pred_test_1 <- knn(train = data_train_nolab, test = data_test_nolab,
                   cl = data_train_labels, k = k, prob = TRUE)

# Se almacenan los valores de probabilidad de cada predicción obtenida
knn_scores1 <- attr(pred_test_1, "prob")
```

Dado que el algoritmo knn (función knn()) da la probabilidad de ser la clase ganadora, debemos transformar la probabilidad de la clase que no sea la ganadora para obtener las probabilidades para poder hacer la curva ROC correctamente. En este caso y según se marca el enunciado, la clase ganadora es la codificada como "+", es decir, el promotor.

```
knn_scores1[pred_test_1 == "+"] <- 1 - knn_scores1[pred_test_1 == "+"]
```

k = 3

```
k = 3
pred_test_3 <- knn(train = data_train_nolab, test = data_test_nolab,
                   cl = data_train_labels, k = k, prob = TRUE)

# Se almacenan los valores de probabilidad de cada predicción obtenida
knn_scores3 <- attr(pred_test_3, "prob")

knn_scores3[pred_test_3 == "+"] <- 1 - knn_scores3[pred_test_3 == "+"]
```

k = 5

```
k = 5
pred_test_5 <- knn(train = data_train_nolab, test = data_test_nolab,
                   cl = data_train_labels, k = k, prob = TRUE)

# Se almacenan los valores de probabilidad de cada predicción obtenida
knn_scores5 <- attr(pred_test_5, "prob")

knn_scores5[pred_test_5 == "+"] <- 1 - knn_scores5[pred_test_5 == "+"]
```

k = 7

```
k = 7
pred_test_7 <- knn(train = data_train_nolab, test = data_test_nolab,
                   cl = data_train_labels, k = k, prob = TRUE)
```

```r
# Se almacenan los valores de probabilidad de cada predicción obtenida
knn_scores7 <- attr(pred_test_7, "prob")

knn_scores7[pred_test_7 == "+"] <- 1 - knn_scores7[pred_test_7 == "+"]
```

2.1.2 Predicción y evaluación

k = 1

Realizamos ahora la matriz de confusión.

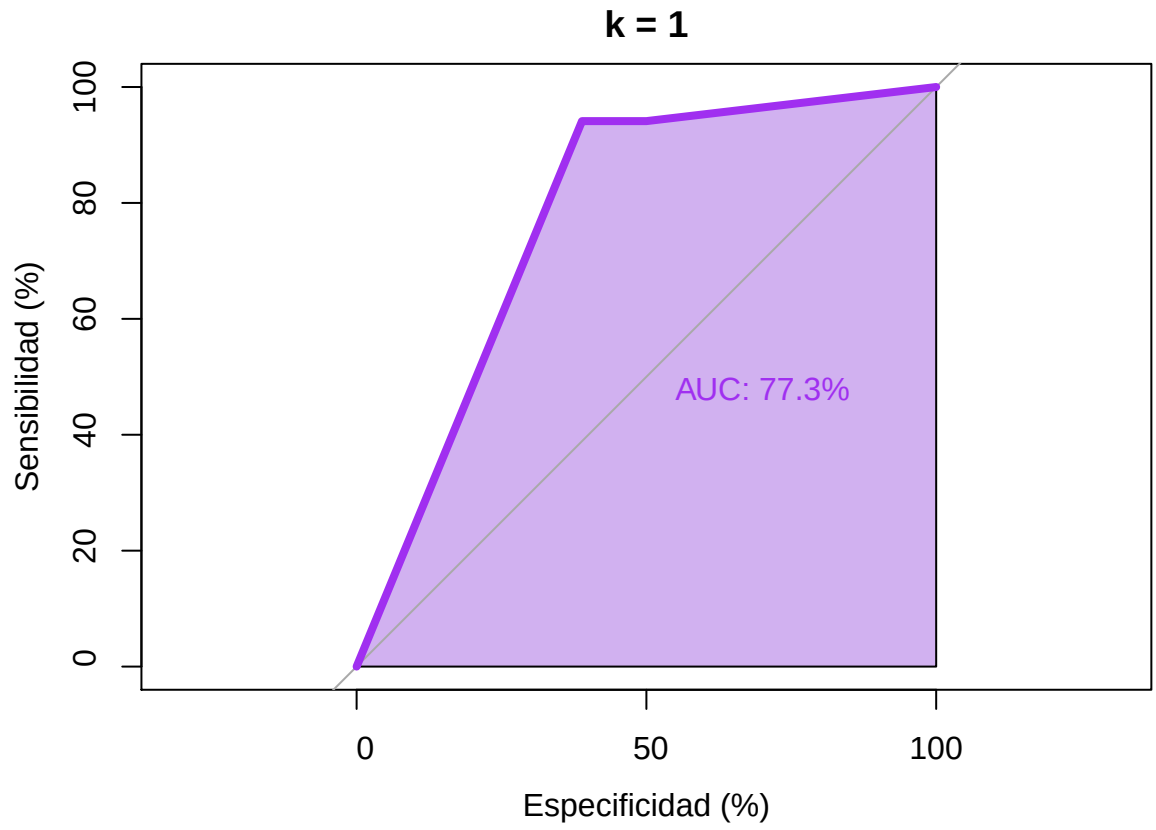
```r
# Matriz de confusión
conf_matrix1 <- confusionMatrix(pred_test_1, data_test_labels,
                                positive = "+",
                                dnn = c("Predicted", "Actual"))
conf_matrix1
```

```
Confusion Matrix and Statistics

         Actual
Predicted  -  +
        - 10  1
        +  8 16

               Accuracy : 0.7429
                 95% CI : (0.5674, 0.8751)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 0.004919

                  Kappa : 0.4911

 Mcnemar's Test P-Value : 0.045500

            Sensitivity : 0.9412
            Specificity : 0.5556
         Pos Pred Value : 0.6667
         Neg Pred Value : 0.9091
             Prevalence : 0.4857
         Detection Rate : 0.4571
   Detection Prevalence : 0.6857
      Balanced Accuracy : 0.7484

       'Positive' Class : +
```

También evaluaremos el modelo a través de la curva ROC.

```r
roc_curve1 <- roc(data_test_labels, knn_scores1, plot = TRUE, legacy.axes = TRUE,
                  percent = TRUE, xlab = "Especificidad (%)",
                  ylab = "Sensibilidad (%)", col = "#A02FF0", lwd = 4,
                  print.auc = TRUE, print.auc.x = 45,
                  auc.polygon = TRUE, auc.polygon.col = "#D1B1F0",
                  auc = TRUE, main = "k = 1")
```

```
Setting levels: control = -, case = +

Setting direction: controls > cases
```



$k = 3$

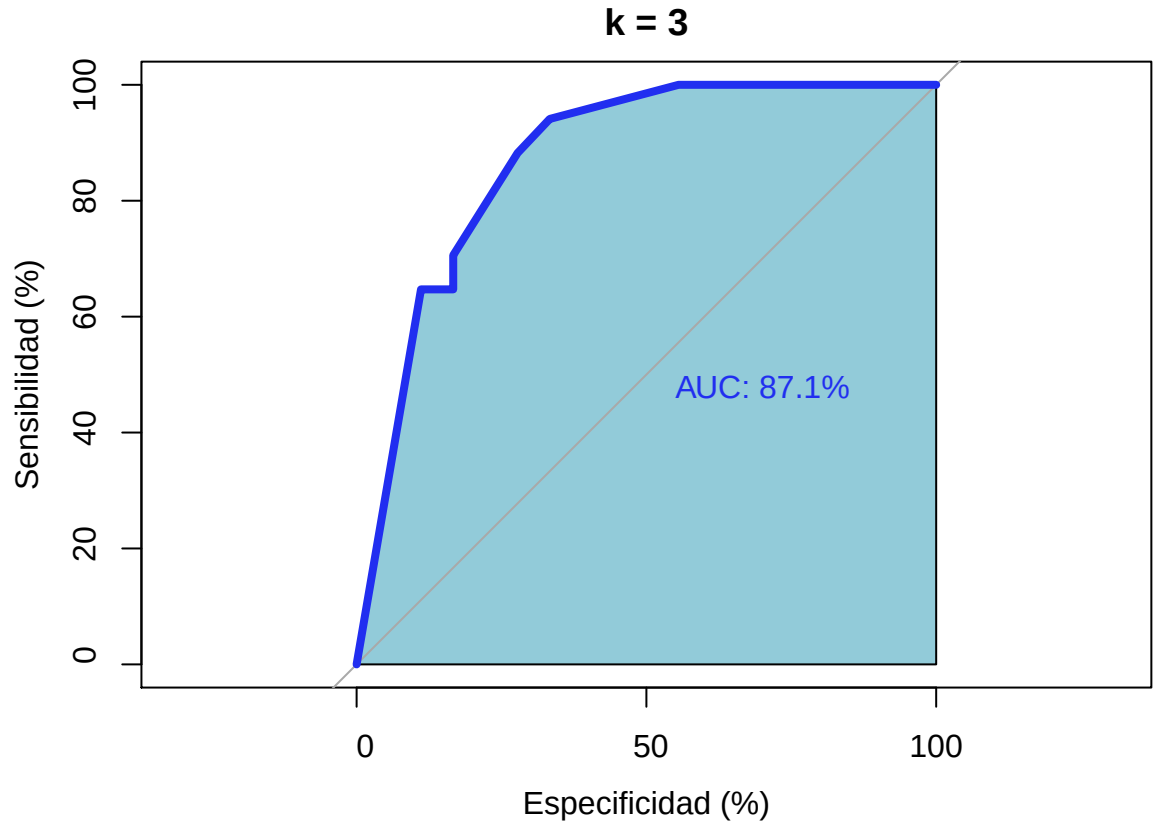
```
# Matriz de confusión
conf_matrix3 <- confusionMatrix(pred_test_3, data_test_labels,
                                positive = "+",
                                dnn = c("Predicted", "Actual"))
conf_matrix3
```

```
Confusion Matrix and Statistics

          Actual
Predicted  -  +
        - 13  2
        +  5 15

               Accuracy : 0.8
                 95% CI : (0.6306, 0.9156)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 0.0004642

                  Kappa : 0.6016
```

```
 Mcnemar's Test P-Value : 0.4496918

            Sensitivity : 0.8824
            Specificity : 0.7222
         Pos Pred Value : 0.7500
         Neg Pred Value : 0.8667
             Prevalence : 0.4857
         Detection Rate : 0.4286
   Detection Prevalence : 0.5714
      Balanced Accuracy : 0.8023

       'Positive' Class : +
```

```r
# Curva ROC
roc_curve3 <- roc(data_test_labels, knn_scores3, plot = TRUE, legacy.axes = TRUE,
                  percent = TRUE, xlab = "Especificidad (%)",
                  ylab = "Sensibilidad (%)", col = "#212EF0", lwd = 4,
                  print.auc = TRUE, print.auc.x = 45,
                  auc.polygon = TRUE, auc.polygon.col = "#92CBD9",
                  auc = TRUE, main = "k = 3")
```

```
Setting levels: control = -, case = +

Setting direction: controls > cases
```



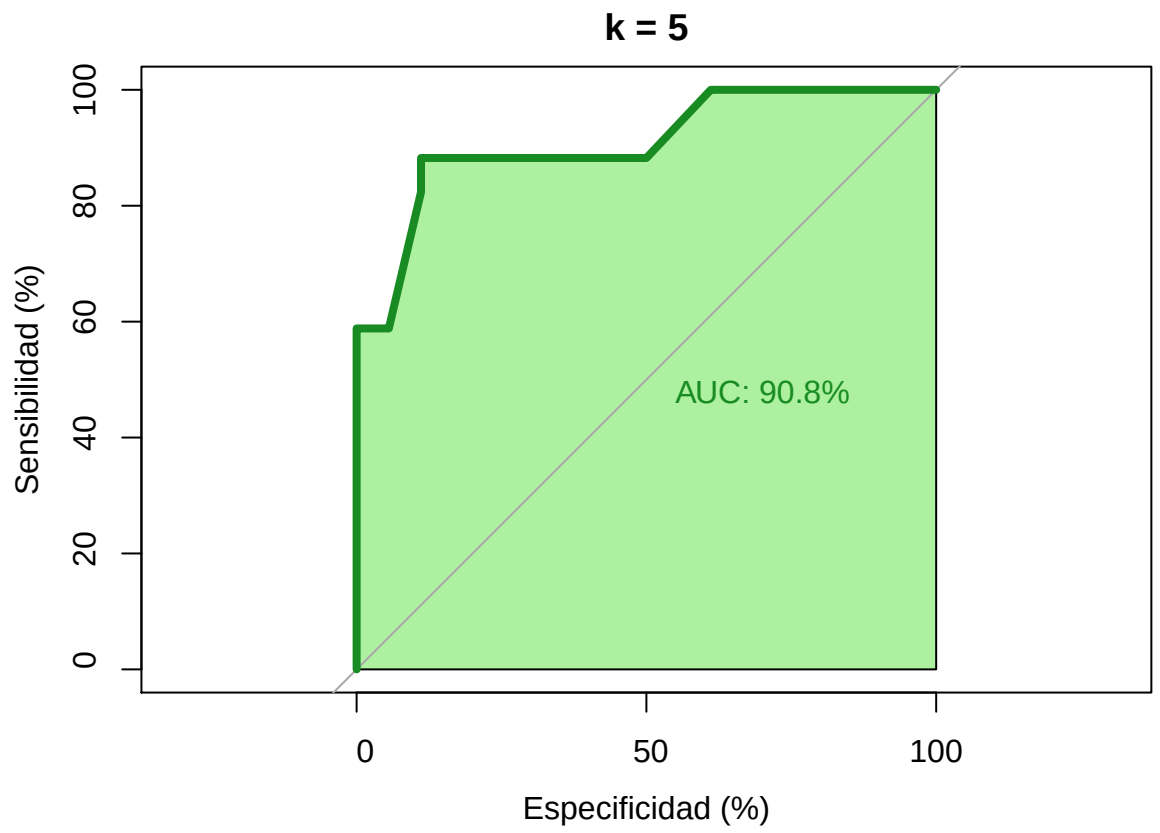
k = 5

```
# Matriz de confusión
conf_matrix5 <- confusionMatrix(pred_test_5, data_test_labels,
                                positive = "+",
                                dnn = c("Predicted", "Actual"))
conf_matrix5
```

```
Confusion Matrix and Statistics

         Actual
Predicted  -  +
        - 10  2
        +  8 15

               Accuracy : 0.7143
                 95% CI : (0.537, 0.8536)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 0.01301

                  Kappa : 0.4337

 Mcnemar's Test P-Value : 0.11385

            Sensitivity : 0.8824
            Specificity : 0.5556
         Pos Pred Value : 0.6522
         Neg Pred Value : 0.8333
             Prevalence : 0.4857
         Detection Rate : 0.4286
   Detection Prevalence : 0.6571
      Balanced Accuracy : 0.7190

       'Positive' Class : +
```

```
# Curva ROC
roc_curve5 <- roc(data_test_labels, knn_scores5, plot = TRUE, legacy.axes = TRUE,
                  percent = TRUE, xlab = "Especificidad (%)",
                  ylab = "Sensibilidad (%)", col = "#188C22", lwd = 4,
                  print.auc = TRUE, print.auc.x = 45,
                  auc.polygon = TRUE, auc.polygon.col = "#ADF09F",
                  auc = TRUE, main = "k = 5")
```

```
Setting levels: control = -, case = +

Setting direction: controls > cases
```



$k = 7$

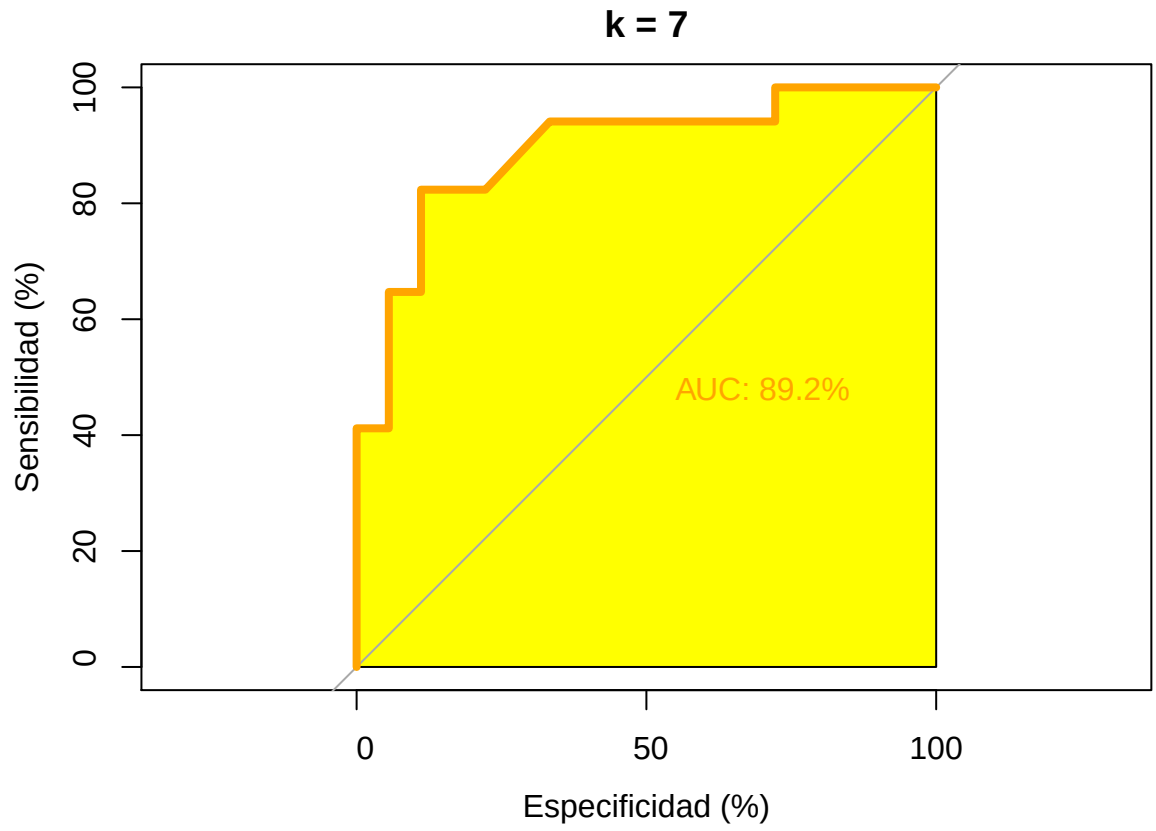
```
# Matriz de confusión
conf_matrix7 <- confusionMatrix(pred_test_7, data_test_labels,
                                positive = "+",
                                dnn = c("Predicted", "Actual"))
conf_matrix7
```

```
Confusion Matrix and Statistics

          Actual
Predicted  -  +
        - 13  1
        +  5 16

               Accuracy : 0.8286
                 95% CI : (0.6635, 0.9344)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 0.0001126

                  Kappa : 0.6591

 Mcnemar's Test P-Value : 0.2206714

            Sensitivity : 0.9412
            Specificity : 0.7222
```

```
         Pos Pred Value : 0.7619
         Neg Pred Value : 0.9286
             Prevalence : 0.4857
         Detection Rate : 0.4571
   Detection Prevalence : 0.6000
      Balanced Accuracy : 0.8317

       'Positive' Class : +
```

```r
# Curva ROC
roc_curve7 <- roc(data_test_labels, knn_scores7, plot = TRUE, legacy.axes = TRUE,
                  percent = TRUE, xlab = "Especificidad (%)",
                  ylab = "Sensibilidad (%)", col = "orange", lwd = 4,
                  print.auc = TRUE, print.auc.x = 45,
                  auc.polygon = TRUE, auc.polygon.col = "yellow",
                  auc = TRUE, main = "k = 7")
```

```
Setting levels: control = -, case = +

Setting direction: controls > cases
```



2.1.3 Conclusión algoritmo

En primer lugar, realizaremos una tabla comparativa en la que se verán los principales valores (en porcentaje) a comparar de forma visual.

k	AUC	Accuracy	Error rate	Sensivity	Specificity	Kappa
1	77.3	74.29	25.71	94.12	55.56	49.11
3	87.1	80.0	20.0	88.24	72.22	60.16
5	90.8	71.43	28.57	88.24	55.56	43.37
7	89.2	82.86	17.14	94.12	72.22	65.91

Tras observar detenidamente los resultados obtenidos y compararlos, podríamos aventurarnos a decir que de las cuatro k's probadas, k = 7 es la que mejores resultados obtiene, ya que es con la que se obtiene la mayor tasa de éxito (y por tanto, menor tasa de error). Además, presenta unos buenos valores de sensibilidad, especificidad AUC y el mejor valor del estadístico kappa.

2.2 Naive Bayes

2.2.1 Entrenamiento modelo

```
# Construcción del clasificador con laplace = 0
bayes0 <- naiveBayes(class ~ .,
                     data = data_train,
                     data_train_labels,
                     laplace = 0)

# Construcción del clasificador con laplace = 1
bayes1 <- naiveBayes(class ~ .,
                     data = data_train,
                     data_train_labels,
                     laplace = 1)
```

2.2.2 Predicción y evaluación

```
pred_bayes0 <- predict(bayes0, data_test)
pred_bayes1 <- predict(bayes1, data_test)
```

Utilizaremos la función confusionMatrix() del paquete caret para construir la tabla de validación cruzada que nos servirá para la evaluación del algoritmo.

```
# Para Laplace = 0
confusionMatrix(pred_bayes0, data_test_labels,
                positive = "+",
                dnn = c("Predicted", "Actual"))
```

```
Confusion Matrix and Statistics

         Actual
Predicted  -  +
        - 17  3
        +  1 14

               Accuracy : 0.8857
                 95% CI : (0.7326, 0.968)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 3.724e-06

                  Kappa : 0.7705

 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.8235
            Specificity : 0.9444
         Pos Pred Value : 0.9333
         Neg Pred Value : 0.8500
             Prevalence : 0.4857
```

```
         Detection Rate : 0.4000
   Detection Prevalence : 0.4286
      Balanced Accuracy : 0.8840

       'Positive' Class : +
```

```r
# Para Laplace = 1
confusionMatrix(pred_bayes1, data_test_labels,
                positive = "+",
                dnn = c("Predicted", "Actual"))
```

```
Confusion Matrix and Statistics

         Actual
Predicted  -  +
        - 17  3
        +  1 14

               Accuracy : 0.8857
                 95% CI : (0.7326, 0.968)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 3.724e-06

                  Kappa : 0.7705

 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.8235
            Specificity : 0.9444
         Pos Pred Value : 0.9333
         Neg Pred Value : 0.8500
             Prevalence : 0.4857
         Detection Rate : 0.4000
   Detection Prevalence : 0.4286
      Balanced Accuracy : 0.8840

       'Positive' Class : +
```

CURVAS ROC

- Para Laplace = 0:

En primer lugar, se obtienen las probabilidades de ser o no promotor ("+").

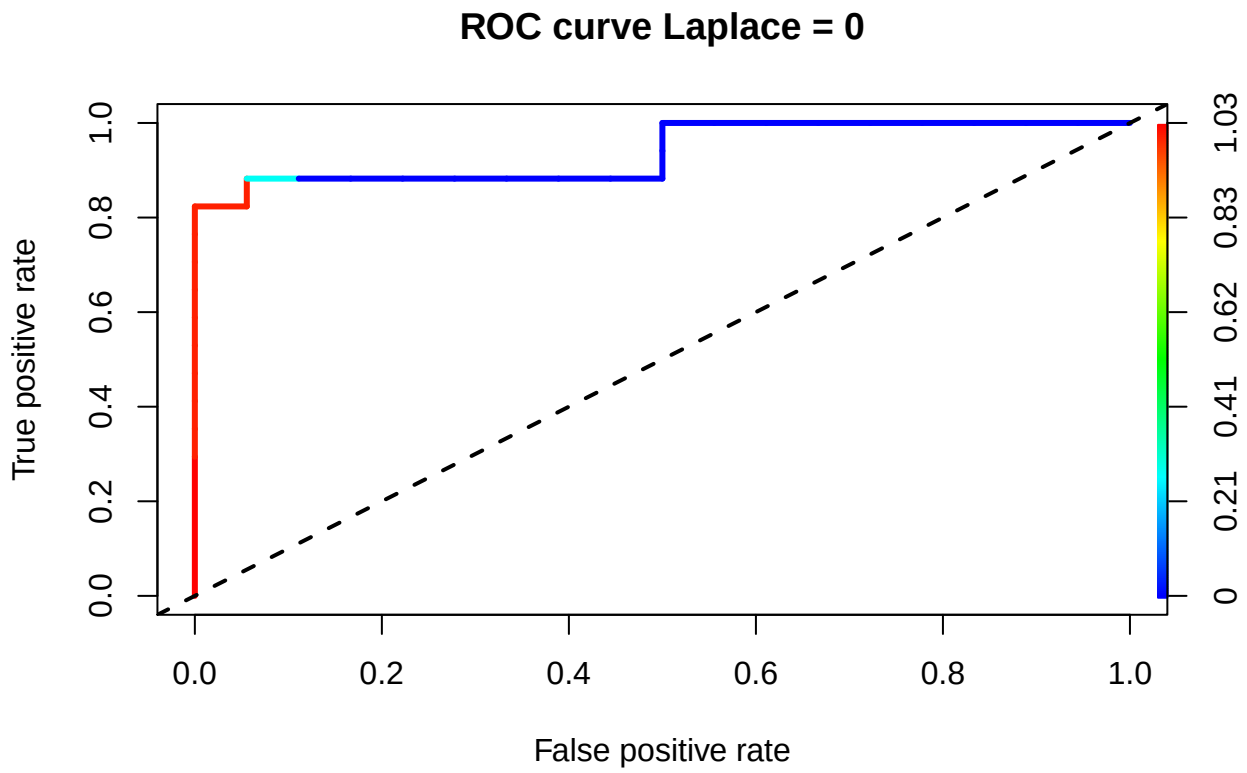
```r
test_pred <- predict(bayes0, data_test_nolab, type = "raw")
tail(test_pred)
```

```
      -            +
[30,] 1 1.461971e-15
[31,] 1 3.729971e-11
```

```
[32,] 1 1.347052e-09
[33,] 1 2.161493e-20
[34,] 1 2.387286e-21
[35,] 1 1.309736e-15
```

Con la información de las probabilidades de la clase positiva ("+") se construye la curva ROC.

```
pred <- prediction(predictions = test_pred[,2], labels = data_test_labels)
perf <- performance(pred, measure = "tpr", x.measure = "fpr")

plot(perf, main = "ROC curve Laplace = 0", col = "blue", lwd=3, colorize = TRUE)
abline(a=0, b=1, lwd=2, lty=2)
```




```
# Área bajo la curva
perf.auc <- performance(pred, measure ="auc")
perf.auc@y.values
```

```
[[1]]
[1] 0.9379085
```

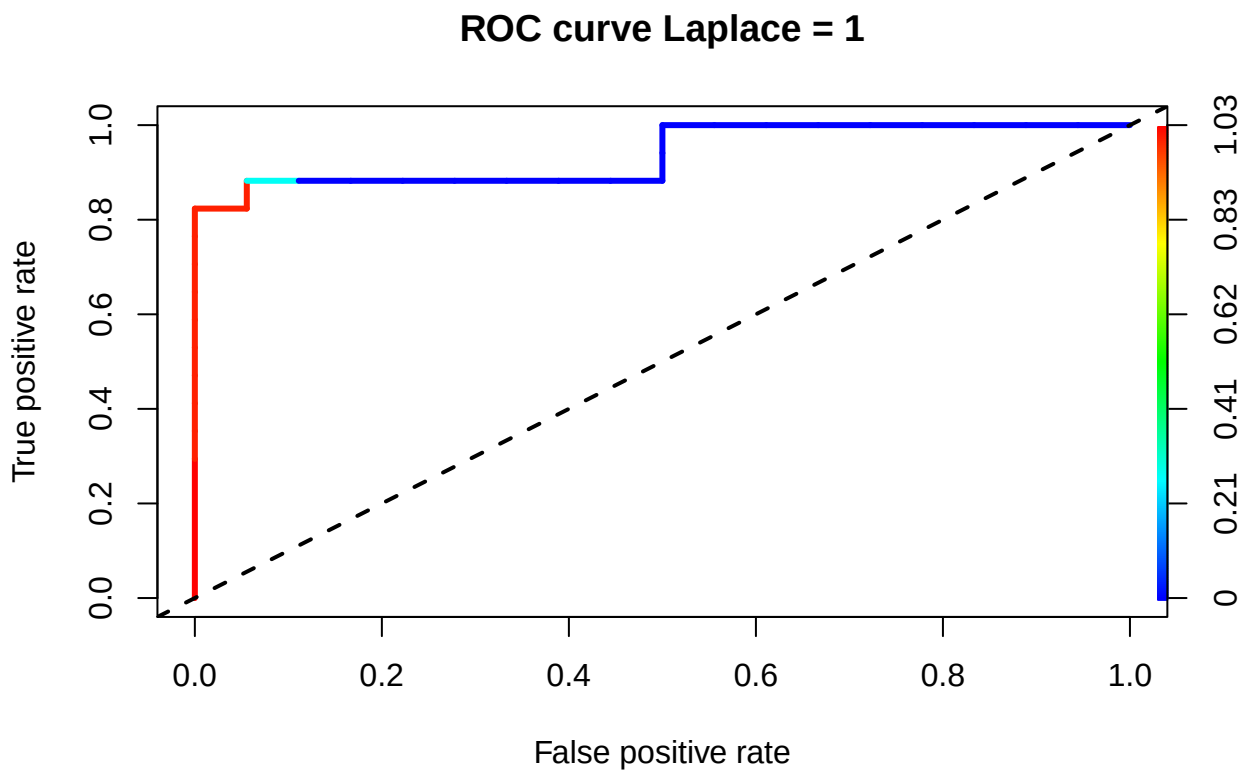
El area bajo la curva es **0.9379085**.

- Para Laplace = 1:

```r
# Calculamos probabilidades
test_pred1 <- predict(bayes1, data_test_nolab, type = "raw")
tail(test_pred1)
```

```
      -           +
[30,] 1 1.461971e-15
[31,] 1 3.729971e-11
[32,] 1 1.347052e-09
[33,] 1 2.161493e-20
[34,] 1 2.387286e-21
[35,] 1 1.309736e-15
```

```r
# Calculamos probabilidad clase positiva y construimos curva ROC
pred1 <- prediction(predictions = test_pred1[,2], labels = data_test_labels)
perf1 <- performance(pred1, measure = "tpr", x.measure = "fpr")

plot(perf1, main = "ROC curve Laplace = 1", col = "red", lwd=3, colorize = TRUE)
abline(a=0, b=1, lwd=2, lty=2)
```




```r
# Área bajo la curva
perf.auc1 <- performance(pred1, measure = "auc")
perf.auc1@y.values
```

```
[[1]]
[1] 0.9379085
```

El area bajo la curva es **0.9379085**.

2.2.3 Conclusión algoritmo

Tal y como se puede observar, tanto con Laplace activado (Laplace = 1) como desactivado (Laplace = 0), se obtienen los mismos resultados. Los resultados son los siguientes:

Laplace	AUC	Accuracy	Error rate	Sensivity	Specificity	Kappa
0	88.57	93.79	11.43	82.35	94.44	77.05
1	88.57	93.79	11.43	82.35	94.44	77.05

Esto puede ser debido a que los datos contengan ya de por sí combinaciones de todas las variables posibles. Por tanto, no tiene sentido aplicar Laplace, ya que se encarga de que cada combinación de factores aparezca al menos una vez. En cuanto a los resultados, se obtienen buenos valores de los parámetros que miden el rendimiento del algoritmo, por lo que se podría decir que el algoritmo funciona bien para la clasificación de estos datos.

2.3 Artificial Neural Network

En este caso, los datos ya están normalizados, puesto que el mínimo es 0 y el máximo 1.

```
summary(data_train)[,2:7]
```

```
       V1                V2                V3             V4                V5
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0   Min.   :0.0000   Min.   :0.0
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0   1st Qu.:0.0000   1st Qu.:0.0
 Median :0.0000   Median :0.0000   Median :0.0   Median :0.0000   Median :0.0
 Mean   :0.3571   Mean   :0.2286   Mean   :0.1   Mean   :0.3143   Mean   :0.3
 3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.0   3rd Qu.:1.0000   3rd Qu.:1.0
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0   Max.   :1.0000   Max.   :1.0
       V6
 Min.   :0.0
 1st Qu.:0.0
 Median :0.0
 Mean   :0.2
 3rd Qu.:0.0
 Max.   :1.0
```

```
summary(data_test)[,2:7]
```

```
       V1                V2                V3                V4
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
 Median :0.0000   Median :0.0000   Median :0.0000   Median :0.0000
 Mean   :0.3429   Mean   :0.3143   Mean   :0.2286   Mean   :0.1143
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
       V5                V6
 Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.0000   1st Qu.:0.0000
 Median :0.0000   Median :0.0000
 Mean   :0.1429   Mean   :0.2286
 3rd Qu.:0.0000   3rd Qu.:0.0000
 Max.   :1.0000   Max.   :1.0000
```

2.3.1 Entrenamiento del modelo

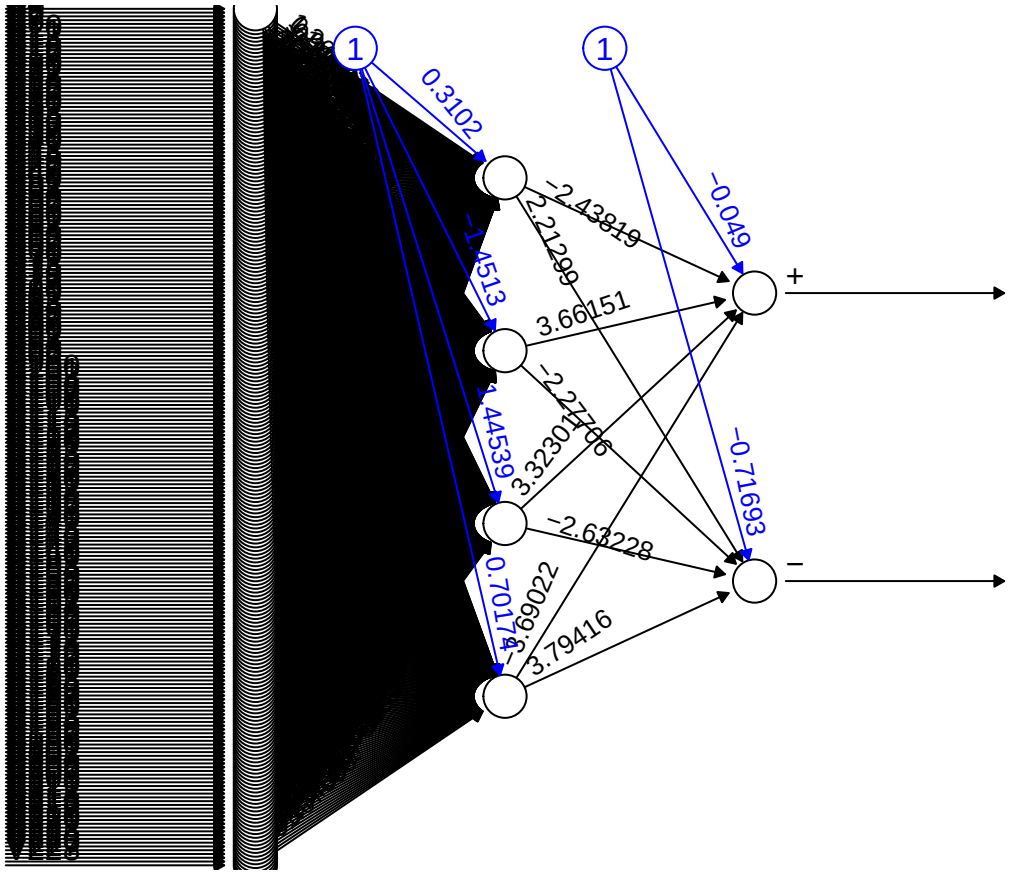
En primer lugar, fijamos la semilla generadora:

```
set.seed(params$seed.clsfier)
```

DE 4 NODOS

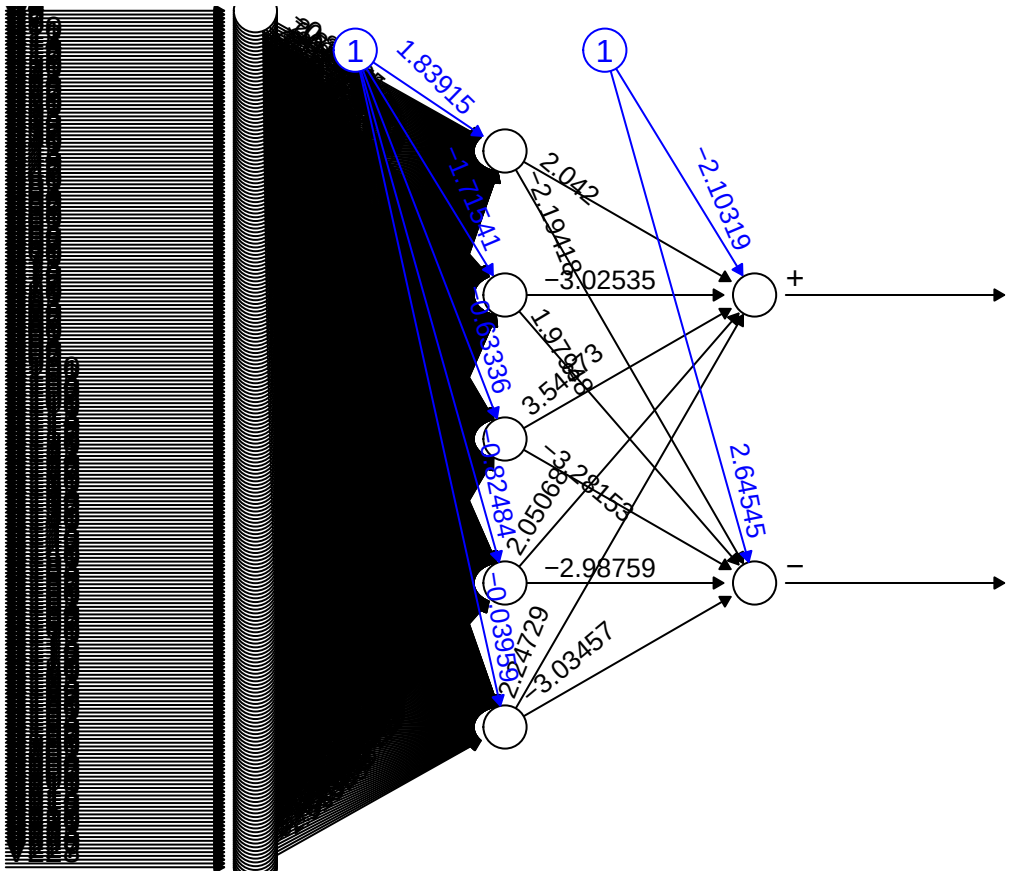
```
# Construcción del modelo
model_ann4 <- neuralnet(class ~ .,
                        data = data_train,
                        hidden = 4,
                        linear.output = FALSE)
```

```r
# Visualización del modelo
plot(model_ann4, rep = "best")
```



DE 5 NODOS

```r
# Construcción del modelo
model_ann5 <- neuralnet(class ~ .,
                        data = data_train,
                        hidden = 5,
                        linear.output = FALSE)

# Visualización del modelo
plot(model_ann5, rep = "best")
```



2.3.2 Predicción y evaluación

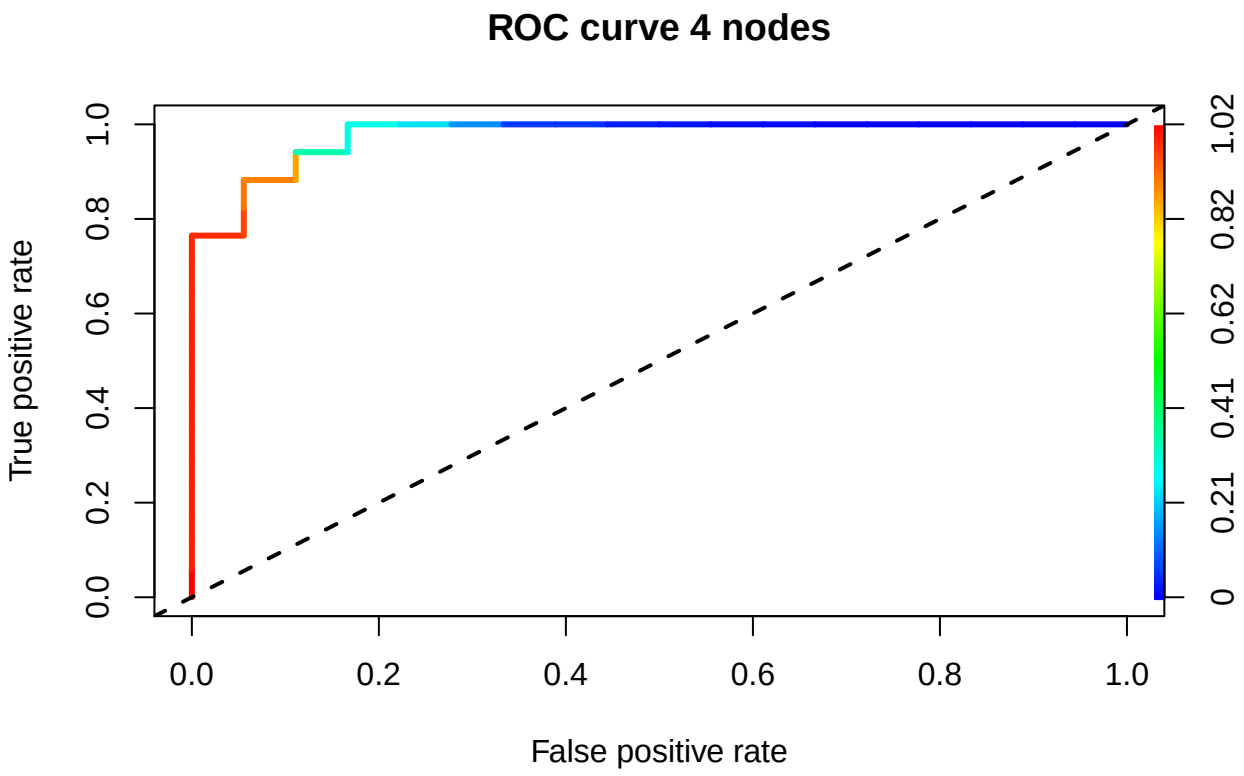
DE 4 NODOS

```
p4 <- neuralnet::compute(model_ann4, data_test)$net.result

# Ahora pasamos el output de binario a categórico
maxidx <- function(arr) {
    return(which(arr == max(arr)))
}

idx <- apply(p4, 1, maxidx)
prediction <- c("-", "+")[idx]
res <- table(prediction, data_test$class)

# Matriz de confusión
cmatrix4 <- confusionMatrix(res, positive = "+")
cmatrix4
```

```
Confusion Matrix and Statistics


prediction  -  +
         - 16  2
         +  2 15
```

```
               Accuracy : 0.8857
                 95% CI : (0.7326, 0.968)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 3.724e-06

                  Kappa : 0.7712

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8824
            Specificity : 0.8889
         Pos Pred Value : 0.8824
         Neg Pred Value : 0.8889
             Prevalence : 0.4857
         Detection Rate : 0.4286
   Detection Prevalence : 0.4857
      Balanced Accuracy : 0.8856

       'Positive' Class : +
```

```r
# Curva ROC
test_pred4 <- predict(model_ann4, data_test_nolab, type = "raw")

# Calculamos probabilidad clase positiva y construimos curva ROC
pred4 <- prediction(predictions = test_pred4[,2], labels = data_test_labels)
perf4 <- performance(pred4, measure = "tpr", x.measure = "fpr")

plot(perf4, main = "ROC curve 4 nodes", col = "red", lwd=3, colorize = TRUE)
abline(a=0, b=1, lwd=2, lty=2)
```




```
# Área bajo la curva
perf.auc4 <- performance(pred4, measure = "auc")
perf.auc4@y.values
```

```
[[1]]
[1] 0.9771242
```

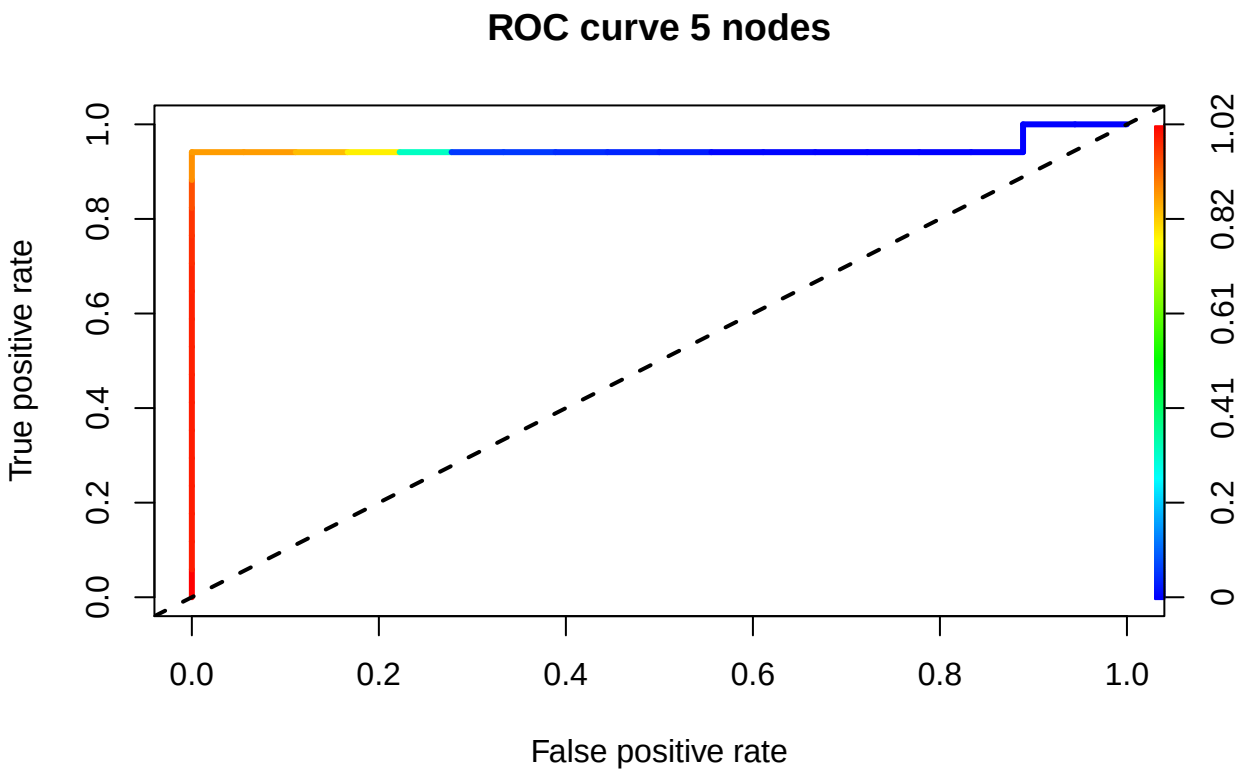
DE 5 NODOS

```
p5 <- neuralnet::compute(model_ann5, data_test)$net.result

# Ahora pasamos el output de binario a categórico
maxidx <- function(arr) {
    return(which(arr == max(arr)))
}

idx <- apply(p5, 1, maxidx)
prediction <- c("-", "+")[idx]
res <- table(prediction, data_test$class)

# Matriz de confusión
cmatrix5 <- confusionMatrix(res, positive = "+")
cmatrix5
```

```
Confusion Matrix and Statistics
```

```
prediction  -  +
         - 15  1
         +  3 16

               Accuracy : 0.8857
                 95% CI : (0.7326, 0.968)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 3.724e-06

                  Kappa : 0.772

 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.9412
            Specificity : 0.8333
         Pos Pred Value : 0.8421
         Neg Pred Value : 0.9375
             Prevalence : 0.4857
         Detection Rate : 0.4571
   Detection Prevalence : 0.5429
      Balanced Accuracy : 0.8873

       'Positive' Class : +
```

```r
# Curva ROC
test_pred5 <- predict(model_ann5, data_test_nolab, type = "raw")

# Calculamos probabilidad clase positiva y construimos curva ROC
pred5 <- prediction(predictions = test_pred5[,2], labels = data_test_labels)
perf5 <- performance(pred5, measure = "tpr", x.measure = "fpr")

plot(perf5, main = "ROC curve 5 nodes", col = "red", lwd=3, colorize = TRUE)
abline(a=0, b=1, lwd=2, lty=2)
```




```
# Área bajo la curva
perf.auc5 <- performance(pred5, measure = "auc")
perf.auc5@y.values
```

```
[[1]]
[1] 0.9477124
```

2.3.3 Conclusión algoritmo

Nodes	AUC	Accuracy	Error rate	Sensivity	Specificity	Kappa
4	97.71	88.57	11.43	88.24	88.89	77.12
5	94.77	88.57	11.43	94.12	83.33	77.20

Tras observar los parámetros obtenidos de las dos opciones diferentes, podríamos decir que tanto la elección de 4 nodos como de 5 obtienen resultados muy similares. Sin embargo, podríamos decir que con 5 nodos se obtiene un clasificador bastante bueno (observando el valor de AUC) a la par que bueno en cuanto a la concordancia entre las predicciones y los valores verdaderos (Kappa). Además, se obtiene una mayor sensibilidad que con 4 nodos. Aún así, es posible que se obtuvieran mejores resultados añadiendo más capas.

2.4 Support Vector Machine

2.4.1 Entrenamiento modelo

KERNEL LINEAL

```
# Kernel lineal
clasific_lineal <- ksvm(class ~ ., data = data_train,
                        kernel = "vanilladot")
```

```
 Setting default kernel parameters
```

```
clasific_lineal
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 61

Objective Function Value : -0.0997
Training error : 0
```

KERNEL RBF (RADIAL BASIS)

```
# Kernel lineal
clasific_rbf <- ksvm(class ~ ., data = data_train,
                        kernel = "rbfdot")
clasific_rbf
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.00215902590705669

Number of Support Vectors : 70

Objective Function Value : -34.3173
Training error : 0
```

2.4.2 Predicción y evaluación

```
predictions_lineal <- predict(clasific_lineal, data_test)
predictions_rbf <- predict(clasific_rbf, data_test)
```

Veremos ahora las matrices de confusión.

```
# Lineal
cmatrix_lineal <- confusionMatrix(predictions_lineal, data_test$class,
                                  positive = "+")
cmatrix_lineal
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 15  1
         +  3 16

               Accuracy : 0.8857
                 95% CI : (0.7326, 0.968)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 3.724e-06

                  Kappa : 0.772

 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.9412
            Specificity : 0.8333
         Pos Pred Value : 0.8421
         Neg Pred Value : 0.9375
             Prevalence : 0.4857
         Detection Rate : 0.4571
   Detection Prevalence : 0.5429
      Balanced Accuracy : 0.8873

       'Positive' Class : +
```

```
# rbf
cmatrix_rbf <- confusionMatrix(predictions_rbf, data_test$class,
                               positive = "+")
cmatrix_rbf
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 17  1
         +  1 16

               Accuracy : 0.9429
                 95% CI : (0.8084, 0.993)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 4.406e-08

                  Kappa : 0.8856
```

```
Mcnemar's Test P-Value : 1

            Sensitivity : 0.9412
            Specificity : 0.9444
         Pos Pred Value : 0.9412
         Neg Pred Value : 0.9444
             Prevalence : 0.4857
         Detection Rate : 0.4571
   Detection Prevalence : 0.4857
      Balanced Accuracy : 0.9428

       'Positive' Class : +
```

2.4.3 Conclusión algoritmo

Kernel	Accuracy	Error rate	Sensivity	Specificity	Kappa
Lineal	88.57	11.43	94.12	83.33	77.20
rbf	94.29	5.71	94.12	94.44	88.56

En el caso del algoritmo Support Vector Machine, podemos observar que el kernel rbf funciona muy bien para clasificar estos datos, ya que se consigue una accuracy del 94.29% (y por tanto una tasa de error del 5.71%), una sensibilidad y especificidad altas y un muy buen valor del estadístico kappa.

2.5 Árbol de Decisión

2.5.1 Entrenamiento del modelo

BOOSTING DESACTIVADO

```
model_b.desact <- C5.0(formula = class ~.,
                       data = data_train,
                       trials = 1,
                       rules = FALSE,
                       control = C5.0Control(seed = 123))
model_b.desact
```

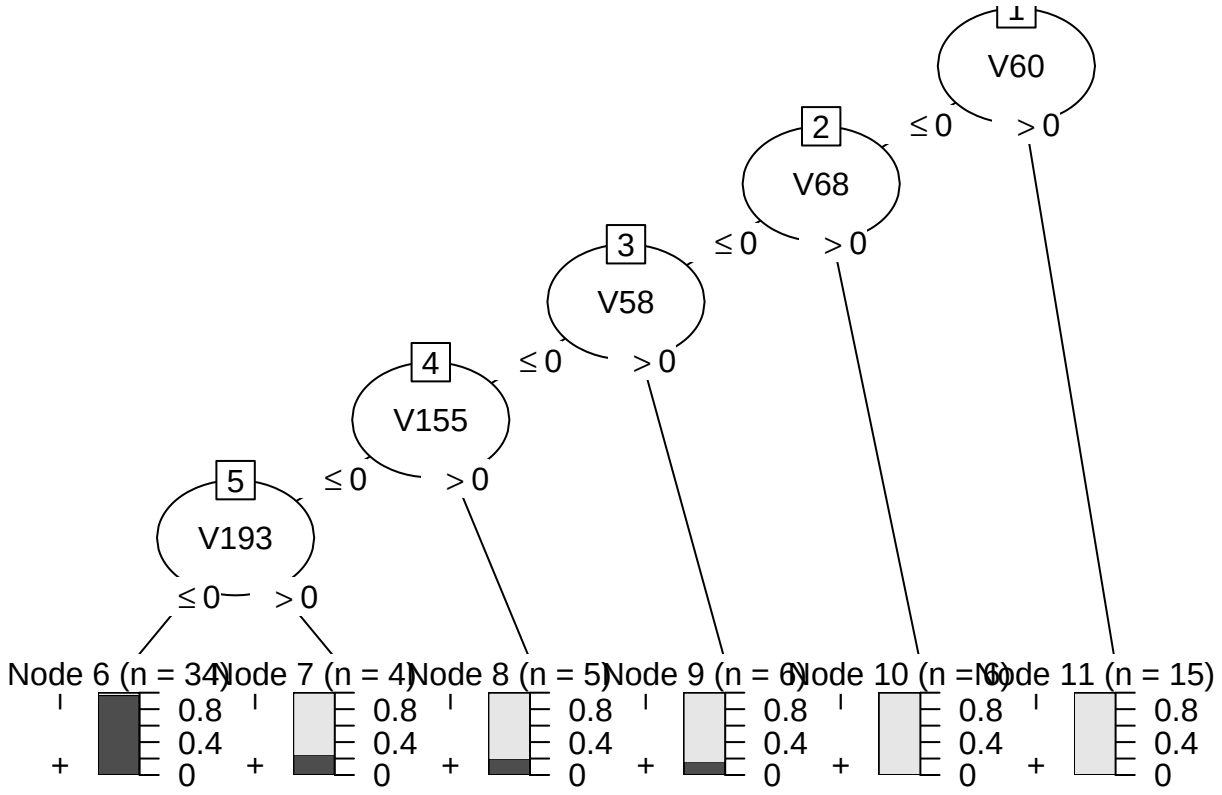
```
Call:
C5.0.formula(formula = class ~ ., data = data_train, trials = 1, rules =
 FALSE, control = C5.0Control(seed = 123))

Classification Tree
Number of samples: 70
Number of predictors: 228

Tree size: 6

Non-standard options: attempt to group attributes
```

```
summary(model_b.desact)
```

```
Call:
C5.0.formula(formula = class ~ ., data = data_train, trials = 1, rules =
 FALSE, control = C5.0Control(seed = 123))


C5.0 [Release 2.07 GPL Edition]     Wed Jan 13 13:20:22 2021
-------------------------------

Class specified by attribute `outcome'

Read 70 cases (229 attributes) from undefined.data

Decision tree:

V60 > 0: - (15)
V60 <= 0:
:...V68 > 0: - (6)
    V68 <= 0:
    :...V58 > 0: - (6/1)
        V58 <= 0:
        :...V155 > 0: - (5/1)
            V155 <= 0:
            :...V193 <= 0: + (34/1)
                V193 > 0: - (4/1)
```

```
Evaluation on training data (70 cases):

        Decision Tree
      ----------------
      Size      Errors

         6    4( 5.7%)   <<


       (a)   (b)    <-classified as
      ----  ----
        33     1    (a): class -
         3    33    (b): class +


    Attribute usage:

    100.00% V60
     78.57% V68
     70.00% V58
     61.43% V155
     54.29% V193


Time: 0.0 secs
```

```r
# Visualización del modelo
plot(model_b.desact)
```



BOOSTING ACTIVADO

En este caso elejimos el valor de trials como 10, porque es el valor estándar, ya que según investigaciones reduce las tasas de error en los datos de prueba un 25% aproximadamente.

```
model_b.act <- C5.0(formula = class ~.,
                        data = data_train,
                        trials = 10,
                        rules = FALSE,
                        control = C5.0Control(seed = 123))
model_b.act
```

```
Call:
C5.0.formula(formula = class ~ ., data = data_train, trials = 10, rules
 = FALSE, control = C5.0Control(seed = 123))

Classification Tree
Number of samples: 70
Number of predictors: 228

Number of boosting iterations: 10
Average tree size: 4.5

Non-standard options: attempt to group attributes
```

```
summary(model_b.act)
```

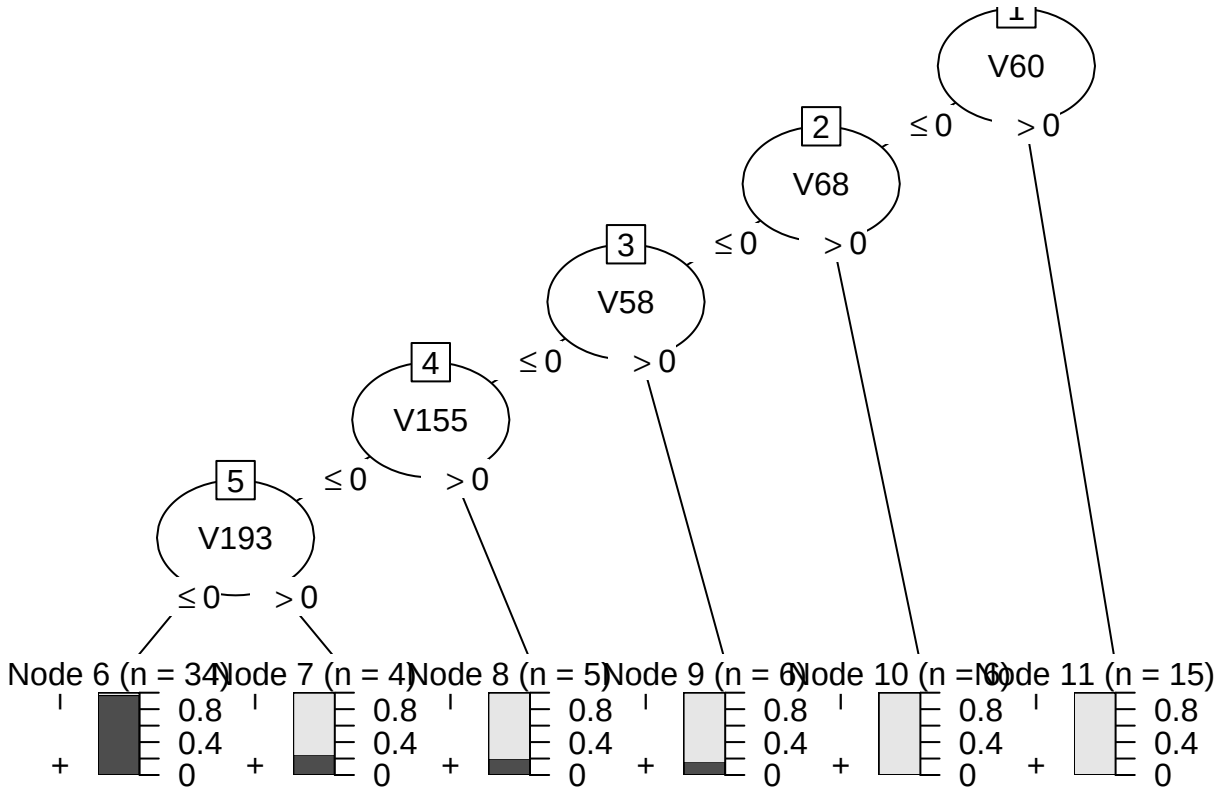
```
Call:
C5.0.formula(formula = class ~ ., data = data_train, trials = 10, rules
 = FALSE, control = C5.0Control(seed = 123))


C5.0 [Release 2.07 GPL Edition]     Wed Jan 13 13:20:22 2021
-------------------------------

Class specified by attribute `outcome'

Read 70 cases (229 attributes) from undefined.data

-----  Trial 0:  -----

Decision tree:

V60 > 0: - (15)
V60 <= 0:
:...V68 > 0: - (6)
    V68 <= 0:
    :...V58 > 0: - (6/1)
        V58 <= 0:
        :...V155 > 0: - (5/1)
            V155 <= 0:
            :...V193 <= 0: + (34/1)
                V193 > 0: - (4/1)

-----  Trial 1:  -----

Decision tree:

V60 > 0: - (11.5)
V60 <= 0:
:...V67 > 0: + (35.3/2.3)
    V67 <= 0:
    :...V57 <= 0: - (12.5)
        V57 > 0: + (10.7/3.8)

-----  Trial 2:  -----

Decision tree:

V61 <= 0: - (35.2/4.7)
V61 > 0:
:...V78 <= 0: + (27.3/2.4)
    V78 > 0: - (7.5/1.2)

-----  Trial 3:  -----

Decision tree:
```

```
V60 > 0: - (8)
V60 <= 0:
:...V68 > 0: - (5.9)
    V68 <= 0:
    :...V158 > 0: - (9.6/0.5)
        V158 <= 0:
        :...V71 > 0: - (2.5)
            V71 <= 0:
            :...V51 <= 0: + (28.8/0.5)
                V51 > 0: - (15.2/6.2)

-----  Trial 4:  -----

Decision tree:

V60 > 0: - (6.3)
V60 <= 0:
:...V68 > 0: - (4.6)
    V68 <= 0:
    :...V24 > 0: + (22.2)
        V24 <= 0:
        :...V193 > 0: - (6.6)
            V193 <= 0:
            :...V155 > 0: - (5.9)
                V155 <= 0:
                :...V78 <= 0: + (22.1/2.8)
                    V78 > 0: - (2.3)

-----  Trial 5:  -----

Decision tree:

V67 <= 0: - (41.4/8)
V67 > 0:
:...V64 <= 0: + (24.5/1.7)
    V64 > 0: - (4.1/0.9)

-----  Trial 6:  -----

Decision tree:

V57 <= 0: - (31/5.2)
V57 > 0:
:...V177 <= 0: + (29/0.9)
    V177 > 0: - (10/3.1)

-----  Trial 7:  -----

Decision tree:

V60 > 0: - (4.5)
V60 <= 0:
:...V58 > 0: - (15.8/2.7)
```

```
    V58 <= 0:
    :...V155 <= 0: + (43.2/5)
        V155 > 0: - (6.5/1.1)

-----  Trial 8:  -----

Decision tree:

V67 > 0: + (29.7/4.4)
V67 <= 0:
:...V57 <= 0: - (18.3)
    V57 > 0:
    :...V149 <= 0: - (14.3/3.8)
        V149 > 0: + (7.6)

-----  Trial 9:  -----

Decision tree:

V71 > 0: - (9.7)
V71 <= 0:
:...V24 > 0: + (15.7/0.3)
    V24 <= 0:
    :...V186 > 0: + (11.5/0.4)
        V186 <= 0:
        :...V129 <= 0: - (25.5/1)
            V129 > 0: + (7.6/1.9)


Evaluation on training data (70 cases):

Trial       Decision Tree
-----     ----------------
      Size      Errors

   0         6    4( 5.7%)
   1         4    8(11.4%)
   2         3   14(20.0%)
   3         6   13(18.6%)
   4         7    3( 4.3%)
   5         3   13(18.6%)
   6         3   10(14.3%)
   7         4    9(12.9%)
   8         4   11(15.7%)
   9         5    9(12.9%)
boost             0( 0.0%)   <<


       (a)   (b)    <-classified as
      ----  ----
        34          (a): class -
              36    (b): class +
```

```
Attribute usage:

100.00% V57
100.00% V60
100.00% V61
100.00% V67
100.00% V71
 85.71% V24
 78.57% V58
 78.57% V68
 72.86% V155
 70.00% V158
 68.57% V193
 64.29% V78
 58.57% V51
 54.29% V177
 50.00% V64
 48.57% V186
 38.57% V129
 20.00% V149


Time: 0.0 secs
```

```r
# Visualización del modelo
plot(model_b.act)
```



2.5.2 Predicción y evaluación

```
# Boosting desactivado

predict_b.desact <- predict(model_b.desact, data_test)
cmatrix_b.desact <- confusionMatrix(predict_b.desact, data_test$class,
                                    positive = "+")
cmatrix_b.desact
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 17  3
         +  1 14

               Accuracy : 0.8857
                 95% CI : (0.7326, 0.968)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 3.724e-06

                  Kappa : 0.7705

 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.8235
            Specificity : 0.9444
         Pos Pred Value : 0.9333
         Neg Pred Value : 0.8500
             Prevalence : 0.4857
         Detection Rate : 0.4000
   Detection Prevalence : 0.4286
      Balanced Accuracy : 0.8840

       'Positive' Class : +
```

```
# Boosting activado

predict_b.act <- predict(model_b.act, data_test)
cmatrix_b.act <- confusionMatrix(predict_b.act, data_test$class,
                                    positive = "+")
cmatrix_b.act
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 18  1
         +  0 16

               Accuracy : 0.9714
```

```
               95% CI : (0.8508, 0.9993)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 2.657e-09

                  Kappa : 0.9427

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9412
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9474
             Prevalence : 0.4857
         Detection Rate : 0.4571
   Detection Prevalence : 0.4571
      Balanced Accuracy : 0.9706

       'Positive' Class : +
```

2.5.3 Conclusión del algoritmo

Boosting	Accuracy	Error rate	Sensivity	Specificity	Kappa
Desactivado	88.57	11.43	82.35	94.44	77.05
Activado	97.14	2.86	94.12	100	94.27

En este caso, se obtiene un mejor y muy buen rendimiento con el boosting activado. Se obtienen valores de los parámetros como una accuracy del 97.14% o una especificidad del 100%, así como un valor de kappa del 94.27% que demuestran que este algoritmo clasifica muy bien estos datos.

2.6 Random Forest

2.6.1 Entrenamiento modelo

```
# Semilla de aleatoriedad para la clasificación
set.seed(params$seed.clsfier)
```

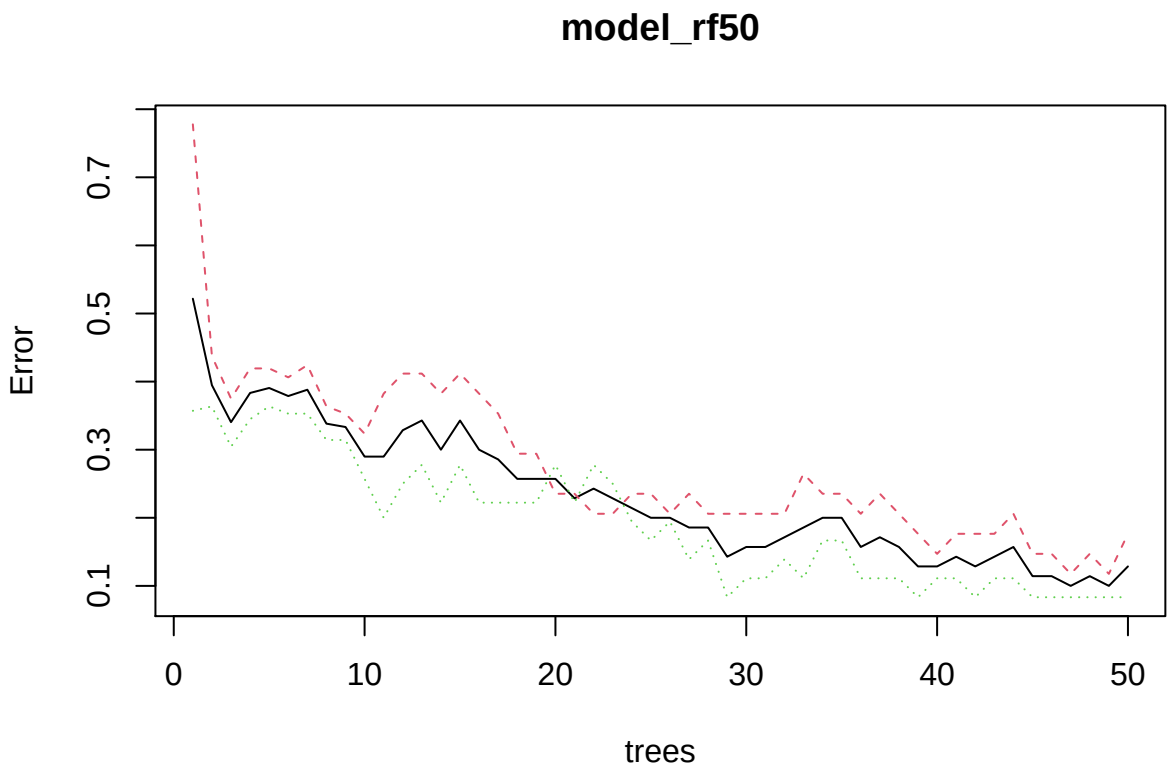
n = 50

```
model_rf50 <- randomForest(class ~.,
                           data = data_train,
                           ntree = 50)
print(model_rf50)
```

```
Call:
 randomForest(formula = class ~ ., data = data_train, ntree = 50)
               Type of random forest: classification
                     Number of trees: 50
No. of variables tried at each split: 15

        OOB estimate of  error rate: 12.86%
Confusion matrix:
   -  + class.error
- 28  6  0.17647059
+  3 33  0.08333333
```

En el bosque hay 50 árboles y se probó 15 variables en cada división.

```
# Visualización modelo
plot(model_rf50)
```



La línea negra representa el OOB, la línea roja el error al intentar predecir la clase y la línea roja el error al intentar predecir la clase -.

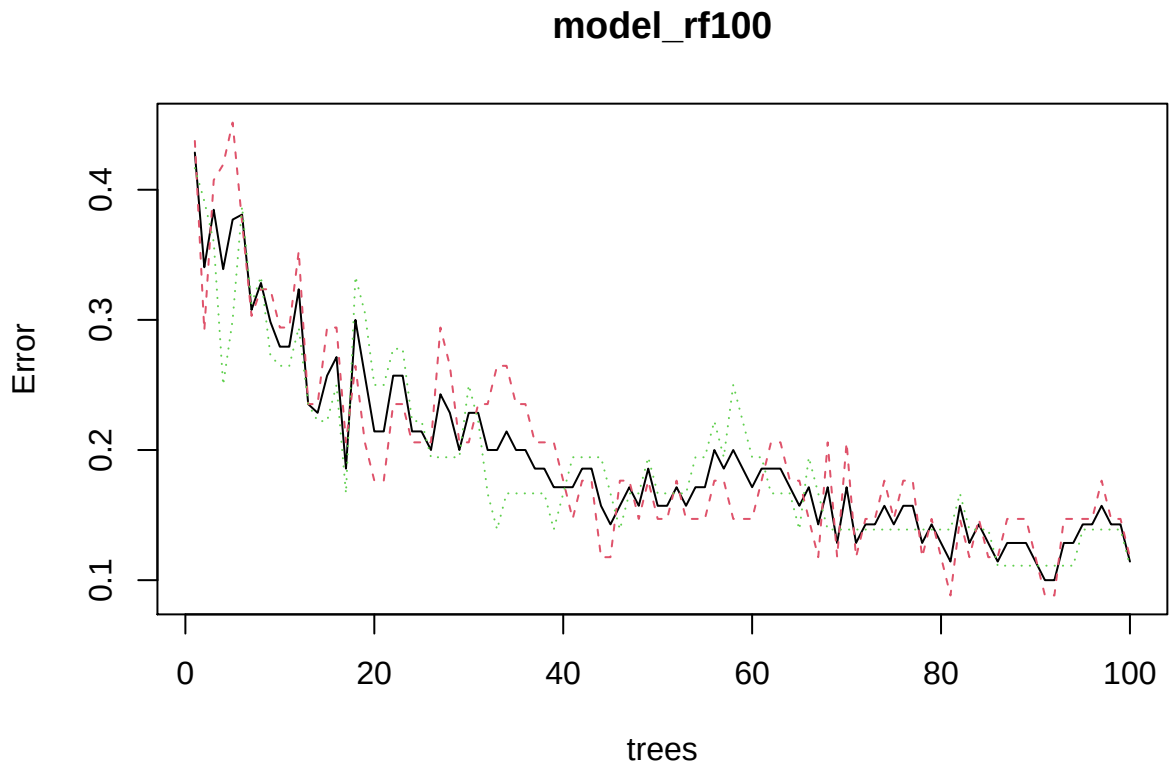
n = 100

```
model_rf100 <- randomForest(class ~.,
                            data = data_train,
                            ntree = 100)
print(model_rf100)
```

```
Call:
 randomForest(formula = class ~ ., data = data_train, ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 15

        OOB estimate of  error rate: 11.43%
Confusion matrix:
   -  + class.error
- 30  4   0.1176471
+  4 32   0.1111111
```

En el bosque hay 100 árboles y se probó 15 variables en cada división.

```r
# Visualización modelo
plot(model_rf100)
```



2.6.2 Predicción y evaluación

n = 50

```r
predict_rf50 <- predict(model_rf50, data_test)
confusionMatrix(data_test$class, predict_rf50)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 17  1
         +  1 16

               Accuracy : 0.9429
                 95% CI : (0.8084, 0.993)
    No Information Rate : 0.5143
    P-Value [Acc > NIR] : 4.406e-08

                  Kappa : 0.8856
```

```
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9444
            Specificity : 0.9412
         Pos Pred Value : 0.9444
         Neg Pred Value : 0.9412
             Prevalence : 0.5143
         Detection Rate : 0.4857
   Detection Prevalence : 0.5143
      Balanced Accuracy : 0.9428

       'Positive' Class : -
```

n = 100

```
predict_rf100 <- predict(model_rf100, data_test)
confusionMatrix(data_test$class, predict_rf100)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 16  2
         +  3 14

               Accuracy : 0.8571
                 95% CI : (0.6974, 0.9519)
    No Information Rate : 0.5429
    P-Value [Acc > NIR] : 8.704e-05

                  Kappa : 0.7136

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8421
            Specificity : 0.8750
         Pos Pred Value : 0.8889
         Neg Pred Value : 0.8235
             Prevalence : 0.5429
         Detection Rate : 0.4571
   Detection Prevalence : 0.5143
      Balanced Accuracy : 0.8586

       'Positive' Class : -
```

2.6.3 Conclusión del algoritmo

n	Accuracy	Error rate	Sensivity	Specificity	Kappa
50	94.29	5.71	94.44	94.12	88.56
100	85.71	14.29	84.21	87.50	71.36

Al igual que en el algorimo anterior, la diferencia de rendimiento entre elegir 50 o 100 árboles en el bosque es bastante grande. En este caso, se obtienen muy buenos rendimientos con 50 árboles, con una accuracy del 94.29% y valores altos de especificidad, sensibilidad y el estadístico Kappa.

3 Discusión y conclusión

Tras haber explorado dentro de cada algoritmo cuáles son los parámetros con los que se obtienen mejores resultados, realizamos una tabla comparativa entre los diferentes algoritmos:

Algorithm	value	AUC	Accuracy	Error rate	Sensivity	Specificity	Kappa
k-NN	k = 7	89.2	82.86	17.14	94.12	72.22	65.91
Naive Bayes	Laplace = 0	88.57	93.79	11.43	82.35	94.44	77.05
ANN	5 nodes	94.77	88.57	11.43	94.12	83.33	77.20
SVM	rbf	-	94.29	5.71	94.12	94.44	88.56
Classification Tree	Boost. act.	-	97.14	2.86	94.12	100	94.27
Random Forest	n = 50	-	94.29	5.71	94.44	94.12	88.56

En general, no se han obtenido malas clasificaciones con ninguno de los algoritmos. Sin embargo, con los últimos tres algoritmos implementados (Support Vector Machine, Classification Tree y Random Forest) se obtiene un mejor rendimiento.

En concreto y según el estudio llevado a cabo, el mejor clasificador para estos datos es el árbol de clasificación con el boosting activado, ya que se obtiene una accuracy del 97.14% y una sensibilidad del 94.12%. Además, se obtiene una especificidad del 100%, por lo que todas las secuencias clasificadas como no promotoras, no lo eran. También se obtiene un valor del estadístico Kappa de 94.27. Este estadístico indica la concordancia entre las predicciones y los valores verdaderos, por lo que en este caso, esta concordancia es muy buena.

Si se observa el gráfico del modelo en el apartado correspondiente, parece que el algoritmo ha encontrado un patrón de 0 y 1 en unas determinadas variables que le ayuda a discernir entre las secuencias que son promotoras y las que no.

Referencias

Hassoun, Mohamad H, and others. 1995. *Fundamentals of Artificial Neural Networks*. MIT press.

Lantz, Brett. 2019. *Machine Learning with R: Expert Techniques for Predictive Modeling*. Packt Publishing Ltd.

Noble, William S. 2006. "What Is a Support Vector Machine?" *Nature Biotechnology* 24 (12): 1565–7.

Xie, Yihui, Joseph J Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. CRC Press.