PEC 1: Predicción 'in-silico' de sitios de escisión reconocidos por la proteasa del HIV-1

Maria Ajenjo Bauza

10/25/2020

Contents

Algoritmo k-NN **2**
Tabla de fortalezas y debilidades algoritmo k-NN 2

Función para codificación ortogonal de octámeros **3**

Step 1: Obtención de los datos **5**

Step 2: Exploración y preparación de los datos **5**
Transformación de los datos: Codificación de los octámeros 8
Preparación de los datos: Separación de los datos en train y test 8

Step 3: Entrenar al modelo con los datos y Step 4: Evaluar el modelo **9**
Discusión resultados 20

Step 5: Mejora del modelo **21**

Referencias **26**

```
# Install packages
# Load packages
# ...
library(yaml)
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 3.6.2
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
```

```
##     intersect, setdiff, setequal, union
```

```
library(formatR)
library(rlist) # listas
library(hash) # diccionarios
```

```
## hash-2.2.6.1 provided by Decision Patterns
```

Algoritmo k-NN

El algoritmo k-NN es uno de los más conocidos entre los métodos de clustering o clasificación. Un algoritmo de clasificación es aquel que utiliza datos etiquetados para poder aprender a clasificar datos sin etiquetar. También es un tipo de algoritmo no supervisado que presenta dos propiedades:

- Es un algoritmo de aprendizaje vago o *lazy learning algorithm*, ya que no cuenta con una fase específica de entrenamiento, sino que usa todos los datos de entrenamiento a la vez que clasifica.
- Es un algoritmo de aprendizaje no paramétrico (*non-parametric learning algorithm*), ya que no asume nada sobre los datos subyacentes.

En concreto, el algoritmo k-NN asume como similar los datos que están cercanos una determinada distancia (k). Para utilizarlo, hay que seguir los pasos que se detallan a continuación:

1. Cargar los datos de entrenamiento (*training dataset*).
2. Elegir el valor de k, que puede ser cualquier número entero.
3. Para cada ejemplo de los datos para test (*test dataset*):
   - Se calcula la distancia entre el dato que estamos evaluando procedente del *test dataset* y los datos del *training dataset*. Se puede calcular la distancia Euclídea, de Manhattan o de Hamming, aunque normalmente se calcula la Euclídea.
   - Se ordenan los datos en orden ascendente según la distancia.
   - Se eligen las filas que tienen distancias que más pequeñas que k.
   - Se asigna al dato testado una clase en función de la clase más frecuente de las filas elegidas en el apartado anterior.

Como todos los algoritmos, el algoritmo k-NN tiene fortalezas y debilidades. En la siguiente tabla podemos observarlas:

Tabla de fortalezas y debilidades algoritmo k-NN

```
sw_knn <- data.frame("Strengths" = c(
                        "Simple and effective",
                        "Makes no assumptions about the underlying data distribution",
                        "Fast training phase",
                        "",
                        ""),
                     "Weaknesses" = c(
                        "Does not produce a model, limiting the ability to understand how the features a
                        "Requires selection of an appropriate k",
                        "Slow classification phase",
                        "Nominal features and missing data require additional processing",
                        "Very sensitive to the noise and dimensionality"))

knitr::kable(sw_knn, "pipe", caption = "Strengths and Weaknesses of the k-NN algorithm")
```

Table 1: Strengths and Weaknesses of the k-NN algorithm

Strengths	Weaknesses
Simple and effective	Does not produce a model, limiting the ability to understand how the features are related to the class
Makes no assumptions about the underlying data distribution	Requires selection of an appropriate k
Fast training phase	Slow classification phase
	Nominal features and missing data require additional processing
	Very sensitive to the noise and dimensionality

Función para codificación ortogonal de octámeros

Vamos a crear una función que tome como argumento una lista con los aminoácidos a codificar y devuelva un diccionario, donde las claves serán cada uno de los aa y los valores su codificación ortogonal.

```
# Creamos una funcion que codifique ortogonalmente los octámeros a la que
# hay que pasarle una lista de los aa que queremos codificar ortogonalmente
ort_cod_aa <- function(lista_aa){
  # Creamos un diccionario para guardar el ouput
  dict_aa <- hash()
  # lista de 0 de la que se parte para cambiar 0 por 1 según el aa
  aa_base <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
  # para cada aa en la lista de aa esenciales
  for (num in 1:length(lista_aa)){
    # se cambia el 0 por el 1 en esa posición
    aa_base[num] = 1
    # se asigna esa lista de 0 y 1 como valor al aa en el diccionario
    dict_aa[[list_aa[num]]] <- aa_base
    # se reestablece la lista de 0
    aa_base[num] = 0
  }
  return(dict_aa)
}
```

```
# Creamos una lista con los 20 aa esenciales con su código de 1 letra
# correspondiente, ordenando los códifos por orden alfabético de nombre

list_aa <- c("A","R","N","D","C","E","Q","G","H","I",
             "L","K","M","F","P","S","T","W","Y","V")
```

Probamos la función con nuestra lista de aa esenciales.

```
ort_cod_aa(list_aa)
```

```
## <hash> containing 20 key-value pair(s).
##   A : 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   C : 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   D : 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   E : 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   F : 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
##   G : 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
##   H : 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
##   I : 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

```
##   K : 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
##   L : 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
##   M : 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
##   N : 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   P : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
##   Q : 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
##   R : 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   S : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
##   T : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
##   V : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
##   W : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
##   Y : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Ahora crearemos una función que tome como argumento una tabla con diferentes octámeros en cada una de las líneas de una misma columna de esa tabla y que devuelva una tabla con todos los octámeros codificados ortogonalmente según nuestra función anterior ort_cod_aa(). En cada una de las filas de la tabla devuelta por la función code_octamer() tendremos la codificación ortogonal del octámero que estaba en esa misma fila en la tabla dada como argumento.

La función que vamos a crear también tomará como argumento una lista de aa a codificar ortogonalmente y llamará a la función creada anteriormente para crear un diccionario.

```
# Argumentos: tabla de octámeros y lista aa a codificar ortogonalmente
code_octamer <- function(octamer_table, lista_aa){
  # Llamamos a la función anterior
  dict_aa <- ort_cod_aa(lista_aa)
  # Creamos una tabla vacía con 160 columnas
  oct_table <- matrix(ncol = 160)
  # Para cada elemento de la tabla de octameros dada
  for (oct in 1:length(octamer_table[,1])){
    # pasamos ese octámero a cadena de caracteres
    oct_char <- as.character(octamer_table[oct,1])
    # creamos una lista vacía donde añadir el resultado de la codificacion
    # de cada aa del octamero
    lista <- c()
    # partimos el octamero en sus diferentes aa
    oct_split <- strsplit(oct_char,"")[[1]]
    # para cada aa de los aa del octamero
    for (i in oct_split){
      # añadimos a la lista el resultado de la codificacion
      lista = append(lista, dict_aa[[i]])
    }
    # añadimos a la tabla el resultado de la codificacion del octamero entero
    oct_table <- rbind(oct_table, lista)
  }
  # Eliminamos la primera fila para que no nos aparezcan los NA que
  # aparecian al crear la matriz vacia
  oct_table <- oct_table[-1,]
  return(oct_table)
}
```

Cargamos una tabla con octámeros.

```
oct_data <- read.table("/Users/Maria/Desktop/Machine_learning/PEC_1/schillingData.txt", sep = ",")
head(oct_data)
```

```
##         V1 V2
```

```
## 1 AAAAAPAK -1
## 2 AAAAPAKV -1
## 3 AAAELGAR -1
## 4 AAAPAKVE -1
## 5 AAAPVAAA -1
## 6 AAAPVVPQ -1
```

Probamos la función con los datos cargados anteriormente.

```
output <- code_octamer(oct_data, list_aa)
head(output[,1:12])
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## lista    1    0    0    0    0    0    0    0    0     0     0     0
## lista    1    0    0    0    0    0    0    0    0     0     0     0
## lista    1    0    0    0    0    0    0    0    0     0     0     0
## lista    1    0    0    0    0    0    0    0    0     0     0     0
## lista    1    0    0    0    0    0    0    0    0     0     0     0
## lista    1    0    0    0    0    0    0    0    0     0     0     0
```

Step 1: Obtención de los datos

Lectura de los datos

En primer lugar, leemos los conjuntos de datos impensData.txt y schillingData.txt.

```
data_imp <- read.table("/Users/Maria/Desktop/Machine_learning/PEC_1/impensData.txt", sep = ",")
data_schi <- read.table("/Users/Maria/Desktop/Machine_learning/PEC_1/schillingData.txt", sep = ",")
```

Step 2: Exploración y preparación de los datos

```
# Observamos los primeros datos de ambos ficheros
head(data_imp)
```

```
##         V1 V2
## 1 AAAGKSGG -1
## 2 AAAVDAGM -1
## 3 AAGKSGGG -1
## 4 AALALEYG  1
## 5 AANDGPMP -1
## 6 AASAAAVD -1
```

```
head(data_schi)
```

```
##         V1 V2
## 1 AAAAAPAK -1
## 2 AAAAPAKV -1
## 3 AAAELGAR -1
## 4 AAAPAKVE -1
## 5 AAAPVAAA -1
## 6 AAAPVVPQ -1
```

```
# Observamos cuántos octámeros contiene cada fichero
length(data_imp[,1])
```

```
## [1] 947
```

```r
length(data_schi[,1])
```

```
## [1] 3272
```

Creación de un nuevo conjunto de datos que sea la unión de ambos

```r
common_data <- rbind(data_schi, data_imp)
head(common_data)
```

```
##         V1 V2
## 1 AAAAAPAK -1
## 2 AAAAPAKV -1
## 3 AAAELGAR -1
## 4 AAAPAKVE -1
## 5 AAAPVAAA -1
## 6 AAAPVVPQ -1
```

```r
length(common_data[,2])
```

```
## [1] 4219
```

Breve descripción de los datos y representación secuencia logo del patrón de cada clase de octámero

```r
head(common_data)
```

```
##         V1 V2
## 1 AAAAAPAK -1
## 2 AAAAPAKV -1
## 3 AAAELGAR -1
## 4 AAAPAKVE -1
## 5 AAAPVAAA -1
## 6 AAAPVVPQ -1
```

En esta nueva tabla obtenemos los octámeros en la primera columna y en la segunda un 1 o un -1, según si se trata de octámeros divididos (1) o sin dividir (-1) por la proteasa HIV-1.

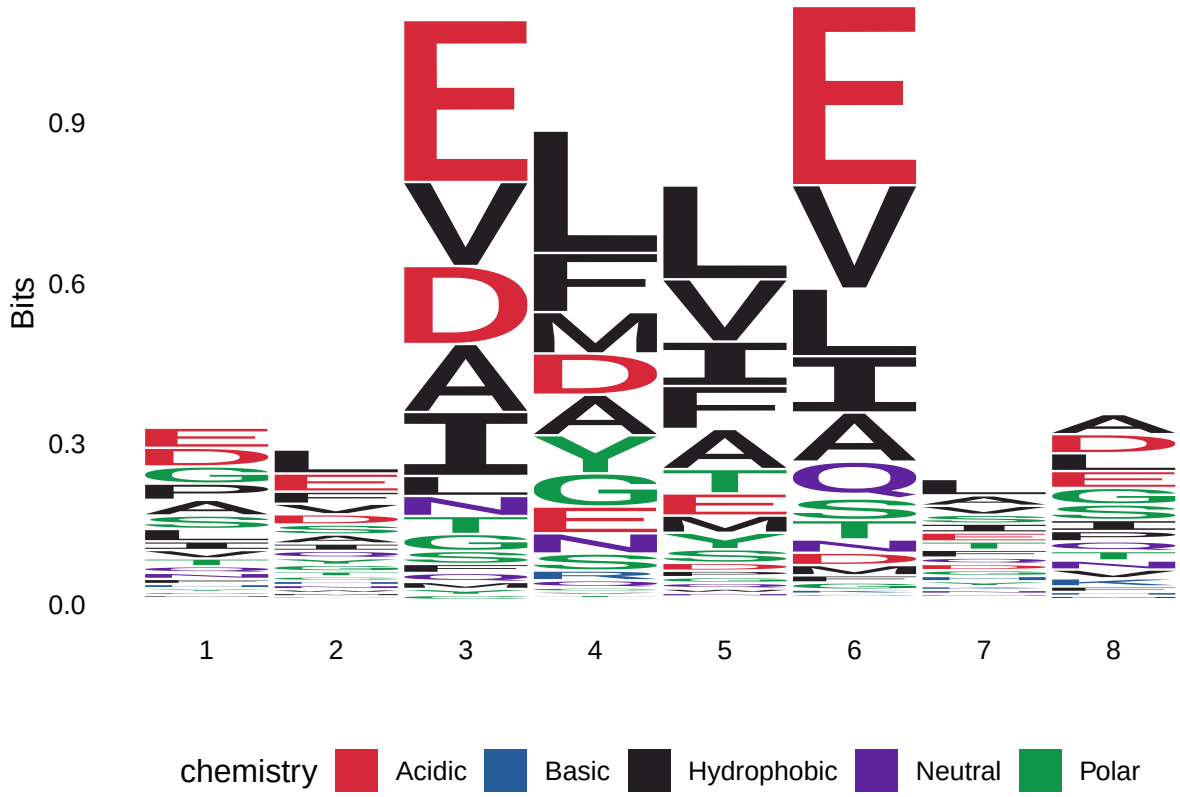
Para la representación del patrón de cada clase de octámero mediante la representación de su secuencia logo instalamos el paquete ggseqlogo.

```r
#library(devtools)
#devtools::install_github("omarwagih/ggseqlogo")
library(ggseqlogo)
```

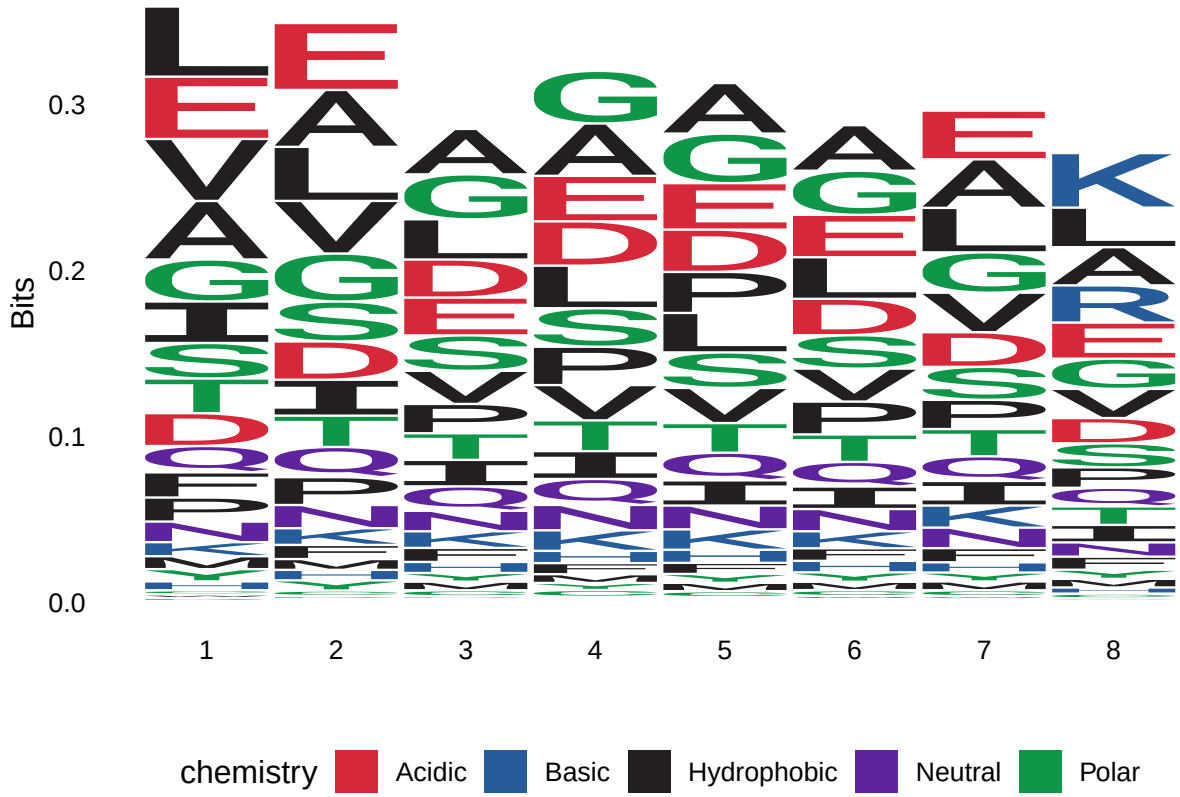
Contaremos como clases los octámeros escindidos por la proteasa HIV-1 (que son los que presentan en la segunda columna un 1) y los no escindidos (-1).

```r
# Separamos los datos por clases
cleav <- common_data[common_data$V2 == 1, ]
uncleav <- common_data[common_data$V2 == -1, ]

# Representación de la secuencia logo de los octámeros escindidos
# por la proteasa HIV-1
ggseqlogo(data = as.character(cleav$V1), seq_type = 'aa')
```




```
# Representación de la secuencia logo de los octámeros no escindidos
# por la proteasa HIV-1
ggseqlogo(data = as.character(uncleav$V1), seq_type = 'aa')
```



Se puede observar que los que sí que se han escindido por la proteasa muestran patrones comunes entre los aa 3 y 6. Esto puede ser debido a que sea esos patrones los que reconozca la proteasa.

Transformación de los datos: Codificación de los octámeros

```
# Utilizamos la función creada anteriormente
oct_code_table <- code_octamer(common_data, list_aa)
# Mostramos algunos resultados
oct_code_table[1250:1270,1:12]
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## lista    0    0    0    0    0    0    0    1    0     0     0     0
## lista    0    0    0    0    0    0    0    1    0     0     0     0
## lista    0    0    0    0    0    0    0    1    0     0     0     0
## lista    0    0    0    0    0    0    0    1    0     0     0     0
## lista    0    0    0    0    0    0    0    1    0     0     0     0
## lista    0    0    0    0    0    0    0    1    0     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
## lista    0    0    0    0    0    0    0    0    1     0     0     0
```

Preparación de los datos: Separación de los datos en train y test

Utilizaremos la tabla creada en el apartado anterior en la codificación de los octámeros con la función code_octamer().

```
# Observamos las dimensiones de la tabla creada
# El primer valor son las filas y el segundo las columnas
dim(oct_code_table)
```

```
## [1] 4219  160
```

A esta tabla vamos a añadir la columna correspondiente a si hay (1) o no (-1) escisión por la proteasa HIV-1, pasándola previamente a factor.

```
common_data$V2 <- factor(common_data$V2, levels = c(-1,1),
                         labels = c("uncleaved", "cleaved"))
head(common_data)
```

```
##         V1        V2
## 1 AAAAAPAK uncleaved
## 2 AAAAPAKV uncleaved
## 3 AAAELGAR uncleaved
## 4 AAAPAKVE uncleaved
```

```
## 5 AAAPVAAA uncleaved
## 6 AAAPVVPQ uncleaved
```

```
# Transformamos en dataframe la matriz de 1 y 0
oct_code_table<- as.data.frame(oct_code_table)
head(oct_code_table[,1:12])
```

```
##         V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12
## lista    1  0  0  0  0  0  0  0  0   0   0   0
## lista.1  1  0  0  0  0  0  0  0  0   0   0   0
## lista.2  1  0  0  0  0  0  0  0  0   0   0   0
## lista.3  1  0  0  0  0  0  0  0  0   0   0   0
## lista.4  1  0  0  0  0  0  0  0  0   0   0   0
## lista.5  1  0  0  0  0  0  0  0  0   0   0   0
```

Añadimos a la matriz la columna cleavage.

```
oct_code_table <- cbind(oct_code_table, common_data$V2)
names(oct_code_table)[161] <- "cleavage"
head(oct_code_table[,155:161])
```

```
##         V155 V156 V157 V158 V159 V160  cleavage
## lista      0    0    0    0    0    0 uncleaved
## lista.1    0    0    0    0    0    1 uncleaved
## lista.2    0    0    0    0    0    0 uncleaved
## lista.3    0    0    0    0    0    0 uncleaved
## lista.4    0    0    0    0    0    0 uncleaved
## lista.5    0    0    0    0    0    0 uncleaved
```

```
# Creamos la semilla aleatoria
set.seed(123)
```

Separamos los datos en training (67%) y test (33%).

```
data_train <- sample_frac(oct_code_table, size = 0.67)
data_test <- sample_frac(oct_code_table, size = 0.33)
```

Step 3: Entrenar al modelo con los datos y Step 4: Evaluar el modelo

Estos dos pasos se realizarán juntos. Para cada valor de k se calcularán todos los parámetros oportunos.

```
# Cargamos el paquete class, que incluye la funcion knn
library(class)
```

```
## Warning: package 'class' was built under R version 3.6.2
```

```
# Cargamos el paquete gmodels que nos permite hacer las CrossTable
library(gmodels)
# Cargamos el paquete ROCR para realizar las curvas ROC
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following object is masked from 'package:gmodels':
##
```

```
##     ci
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# Cargamos el paquete caret para poder extraer estadísticas de la
# matriz de confusión
library(caret)
```

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

```
## Warning: package 'lattice' was built under R version 3.6.2
```

Una vez creadas las clases guardamos las etiquetas de clase en vectores:

```
# Asignamos las etiquetas correspondientes a unos nuevos vectores
data_train_labels <- data_train$cleavage
data_test_labels <- data_test$cleavage
# Eliminamos la fila de cleavage de los datos train y test
data_train <- data_train[-161]
data_test <- data_test[-161]
```

k = 3

```
k = 3
pred_test_k <- knn(train = data_train, test = data_test,
                cl = data_train_labels, k = k, prob = TRUE)

# Almacenamos los valores de probabilidad de cada predicción obtenida
knn_scores <- attr(pred_test_k, "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 cleaved o 1.

```
# Calculamos la probabilidad de ser clase ganadora
# Cambiamos la p(no ser clase ganadora) por
# 1 - p(ser clase ganadora)
knn_scores[pred_test_k == "uncleaved"] <- 1 - knn_scores[pred_test_k == "uncleaved"]
```

Realizamos ahora la matriz de confusión.

```
# Matriz confusión
conf_matrix <- CrossTable(x = data_test_labels, y = pred_test_k, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
```

```
## Total Observations in Table:  1392
##
##
##                   | pred_test_k
## data_test_labels | uncleaved |   cleaved | Row Total |
## -----------------|-----------|-----------|-----------|
##        uncleaved |      1185 |        24 |      1209 |
##                  |     0.980 |     0.020 |     0.869 |
##                  |     0.924 |     0.218 |           |
##                  |     0.851 |     0.017 |           |
## -----------------|-----------|-----------|-----------|
##          cleaved |        97 |        86 |       183 |
##                  |     0.530 |     0.470 |     0.131 |
##                  |     0.076 |     0.782 |           |
##                  |     0.070 |     0.062 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |      1282 |       110 |      1392 |
##                  |     0.921 |     0.079 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

Calcularemos algunos parámetros gracias a los datos extraídos de la matriz de confusión, para observar cómo de buena es la clasificación que realiza nuestro algoritmo.

```
# Almacenamos los valores de la matriz de confusión
tp <- conf_matrix$t[4]
tn <- conf_matrix$t[1]
fp <- conf_matrix$t[3]
fn <- conf_matrix$t[2]

# Accuracy
ac = (tp + tn)/(tp+tn+fp+fn)
cat("Accuracy:", ac, "\n")
```

```
## Accuracy: 0.9130747
```

```
# Error rate
er = 1 - ac
cat("Error Rate:", er, "\n")
```

```
## Error Rate: 0.08692529
```

```
# Sensivity
sen = tp/(tp+fn)
cat("Sensivity:", sen, "\n")
```

```
## Sensivity: 0.4699454
```

```
# Specificity
spec = tn/(tn+fp)
cat("Specificity:", spec, "\n")
```

```
## Specificity: 0.9801489
```

```
# Precision
prec = tp/(tp + fp)
cat("Precision:", prec, "\n")
```

```
## Precision: 0.7818182
```

```
# Recall
rec = tp/(tp + fn)
cat("Recall:", rec, "\n")
```

```
## Recall: 0.4699454
```

```
# F-measure
fmes = (2*tp)/(2*tp + fp + fn)
cat("F-measure:", fmes, "\n")
```

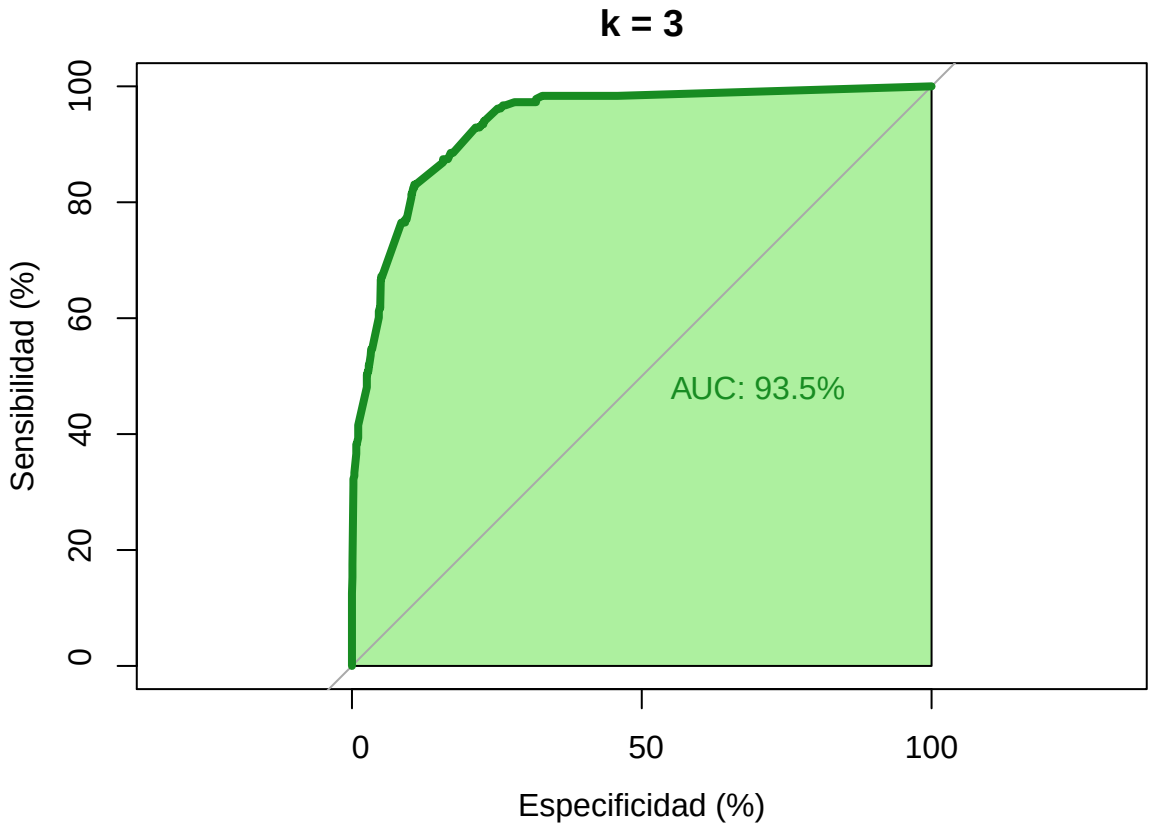
```
## F-measure: 0.5870307
```

Por último, realizaremos ahora la curva ROC y para ello utilizaremos la función roc() del paquete pROC.

```
roc_curve <- roc(data_test_labels, knn_scores, 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 = 3")
```

```
## Setting levels: control = uncleaved, case = cleaved
```

```
## Setting direction: controls < cases
```




```
# AUC
roc_curve$auc
```

```
## Area under the curve: 93.5%
```

k = 5

```
k = 5
pred_test_k <- knn(train = data_train, test = data_test,
                cl = data_train_labels, k = k, prob = TRUE)

# Almacenamos los valores de probabilidad de cada predicción obtenida
knn_scores <- attr(pred_test_k, "prob")
```

```
# Calculamos la probabilidad de ser clase ganadora
# Cambiamos la p(no ser clase ganadora) por
# 1 - p(ser clase ganadora)
knn_scores[pred_test_k == "uncleaved"] <- 1 - knn_scores[pred_test_k == "uncleaved"]
```

```
# Matriz confusión
conf_matrix <- CrossTable(x = data_test_labels, y = pred_test_k, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1392
##
##
##                  | pred_test_k
## data_test_labels | uncleaved |   cleaved | Row Total |
## -----------------|-----------|-----------|-----------|
##        uncleaved |      1195 |        14 |      1209 |
##                  |     0.988 |     0.012 |     0.869 |
##                  |     0.907 |     0.189 |           |
##                  |     0.858 |     0.010 |           |
## -----------------|-----------|-----------|-----------|
##          cleaved |       123 |        60 |       183 |
##                  |     0.672 |     0.328 |     0.131 |
##                  |     0.093 |     0.811 |           |
##                  |     0.088 |     0.043 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |      1318 |        74 |      1392 |
##                  |     0.947 |     0.053 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

```
# Almacenamos los valores de la matriz de confusión
tp <- conf_matrix$t[4]
tn <- conf_matrix$t[1]
fp <- conf_matrix$t[3]
fn <- conf_matrix$t[2]
```

```r
# Accuracy
ac = (tp + tn)/(tp+tn+fp+fn)
cat("Accuracy:", ac, "\n")
```

```
## Accuracy: 0.9015805
```

```r
# Error rate
er = 1 - ac
cat("Error Rate:", er, "\n")
```

```
## Error Rate: 0.09841954
```

```r
# Sensivity
sen = tp/(tp+fn)
cat("Sensivity:", sen, "\n")
```

```
## Sensivity: 0.3278689
```

```r
# Specificity
spec = tn/(tn+fp)
cat("Specificity:", spec, "\n")
```

```
## Specificity: 0.9884202
```

```r
# Precision
prec = tp/(tp + fp)
cat("Precision:", prec, "\n")
```

```
## Precision: 0.8108108
```

```r
# Recall
rec = tp/(tp + fn)
cat("Recall:", rec, "\n")
```

```
## Recall: 0.3278689
```

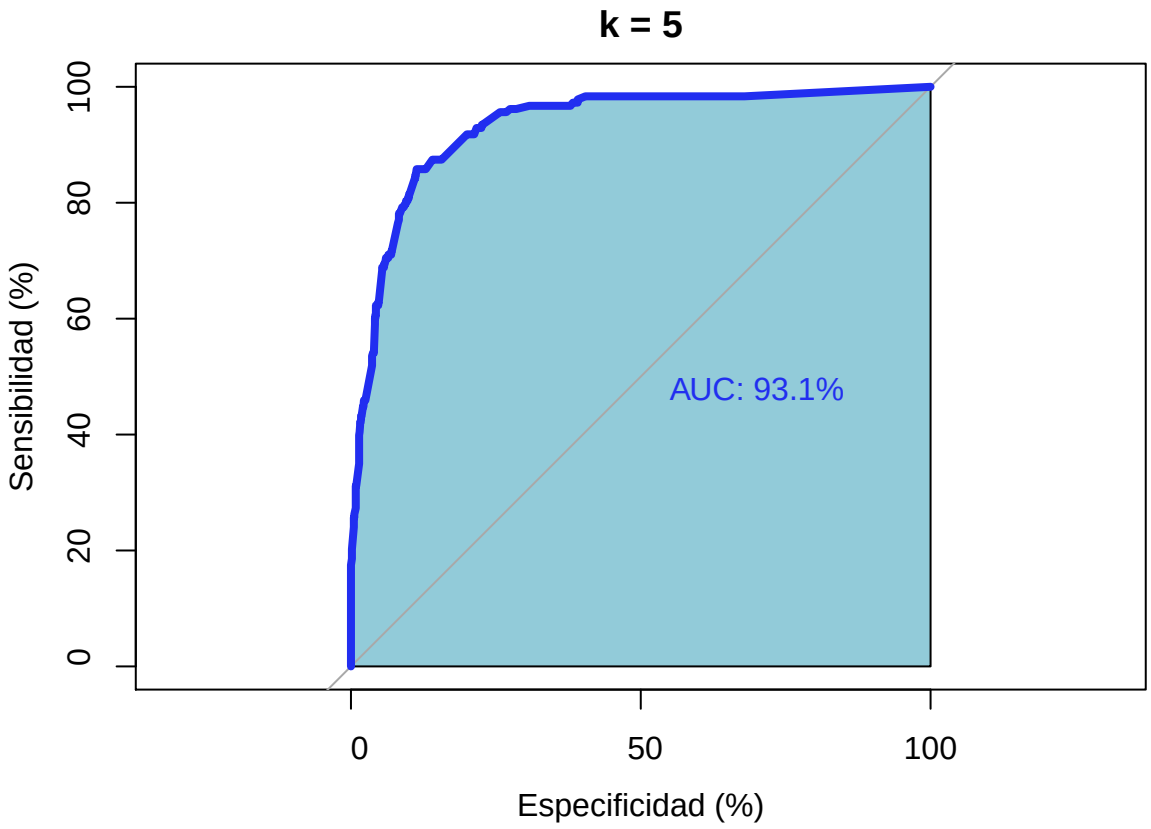
```r
# F-measure
fmes = (2*tp)/(2*tp + fp + fn)
cat("F-measure:", fmes, "\n")
```

```
## F-measure: 0.4669261
```

```r
roc_curve <- roc(data_test_labels, knn_scores, 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 = 5")
```

```
## Setting levels: control = uncleaved, case = cleaved
```

```
## Setting direction: controls < cases
```




```
# AUC
roc_curve$auc
```

```
## Area under the curve: 93.11%
```

k = 7

```
k = 7
pred_test_k <- knn(train = data_train, test = data_test,
                   cl = data_train_labels, k = k, prob = TRUE)

# Almacenamos los valores de probabilidad de cada predicción obtenida
knn_scores <- attr(pred_test_k, "prob")
```

```
# Calculamos la probabilidad de ser clase ganadora
# Cambiamos la p(no ser clase ganadora) por
# 1 - p(ser clase ganadora)
knn_scores[pred_test_k == "uncleaved"] <- 1 - knn_scores[pred_test_k == "uncleaved"]
```

```
# Matriz confusión
conf_matrix <- CrossTable(x = data_test_labels, y = pred_test_k, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
```

```
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1392
##
##
##                  | pred_test_k
## data_test_labels | uncleaved |   cleaved | Row Total |
## -----------------|-----------|-----------|-----------|
##        uncleaved |      1199 |        10 |      1209 |
##                  |     0.992 |     0.008 |     0.869 |
##                  |     0.896 |     0.185 |           |
##                  |     0.861 |     0.007 |           |
## -----------------|-----------|-----------|-----------|
##          cleaved |       139 |        44 |       183 |
##                  |     0.760 |     0.240 |     0.131 |
##                  |     0.104 |     0.815 |           |
##                  |     0.100 |     0.032 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |      1338 |        54 |      1392 |
##                  |     0.961 |     0.039 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

```
# Almacenamos los valores de la matriz de confusión
tp <- conf_matrix$t[4]
tn <- conf_matrix$t[1]
fp <- conf_matrix$t[3]
fn <- conf_matrix$t[2]

# Accuracy
ac = (tp + tn)/(tp+tn+fp+fn)
cat("Accuracy:", ac, "\n")
```

```
## Accuracy: 0.8929598
```

```
# Error rate
er = 1 - ac
cat("Error Rate:", er, "\n")
```

```
## Error Rate: 0.1070402
```

```
# Sensivity
sen = tp/(tp+fn)
cat("Sensivity:", sen, "\n")
```

```
## Sensivity: 0.2404372
```

```
# Specificity
spec = tn/(tn+fp)
cat("Specificity:", spec, "\n")
```

```
## Specificity: 0.9917287
```

```r
# Precision
prec = tp/(tp + fp)
cat("Precision:", prec, "\n")
```

```
## Precision: 0.8148148
```

```r
# Recall
rec = tp/(tp + fn)
cat("Recall:", rec, "\n")
```

```
## Recall: 0.2404372
```

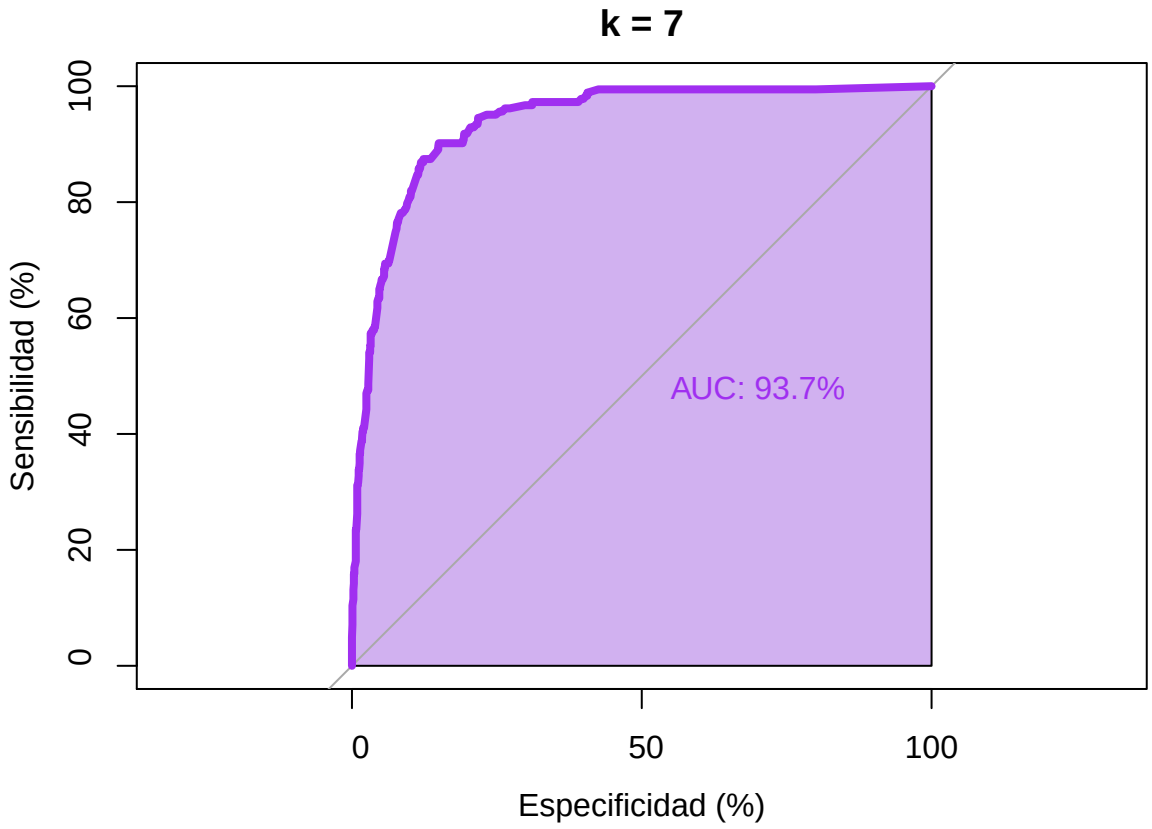
```r
# F-measure
fmes = (2*tp)/(2*tp + fp + fn)
cat("F-measure:", fmes, "\n")
```

```
## F-measure: 0.371308
```

```r
roc_curve <- roc(data_test_labels, knn_scores, 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 = 7")
```

```
## Setting levels: control = uncleaved, case = cleaved
```

```
## Setting direction: controls < cases
```




```r
# AUC
roc_curve$auc
```

```
## Area under the curve: 93.7%
```

k = 11

```
k = 11
pred_test_k <- knn(train = data_train, test = data_test,
                cl = data_train_labels, k = k, prob = TRUE)

# Almacenamos los valores de probabilidad de cada predicción obtenida
knn_scores <- attr(pred_test_k, "prob")
```

```
# Calculamos la probabilidad de ser clase ganadora
# Cambiamos la p(no ser clase ganadora) por
# 1 - p(ser clase ganadora)
knn_scores[pred_test_k == "uncleaved"] <- 1 - knn_scores[pred_test_k == "uncleaved"]
```

```
# Matriz confusión
conf_matrix <- CrossTable(x = data_test_labels, y = pred_test_k, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1392
##
##
##                  | pred_test_k
## data_test_labels | uncleaved |   cleaved | Row Total |
## -----------------|-----------|-----------|-----------|
##        uncleaved |      1206 |         3 |      1209 |
##                  |     0.998 |     0.002 |     0.869 |
##                  |     0.890 |     0.081 |           |
##                  |     0.866 |     0.002 |           |
## -----------------|-----------|-----------|-----------|
##          cleaved |       149 |        34 |       183 |
##                  |     0.814 |     0.186 |     0.131 |
##                  |     0.110 |     0.919 |           |
##                  |     0.107 |     0.024 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |      1355 |        37 |      1392 |
##                  |     0.973 |     0.027 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

```
# Almacenamos los valores de la matriz de confusión
tp <- conf_matrix$t[4]
tn <- conf_matrix$t[1]
```

```r
fp <- conf_matrix$t[3]
fn <- conf_matrix$t[2]

# Accuracy
ac = (tp + tn)/(tp+tn+fp+fn)
cat("Accuracy:", ac, "\n")
```

```
## Accuracy: 0.8908046
```

```r
# Error rate
er = 1 - ac
cat("Error Rate:", er, "\n")
```

```
## Error Rate: 0.1091954
```

```r
# Sensivity
sen = tp/(tp+fn)
cat("Sensivity:", sen, "\n")
```

```
## Sensivity: 0.1857923
```

```r
# Specificity
spec = tn/(tn+fp)
cat("Specificity:", spec, "\n")
```

```
## Specificity: 0.9975186
```

```r
# Precision
prec = tp/(tp + fp)
cat("Precision:", prec, "\n")
```

```
## Precision: 0.9189189
```

```r
# Recall
rec = tp/(tp + fn)
cat("Recall:", rec, "\n")
```

```
## Recall: 0.1857923
```

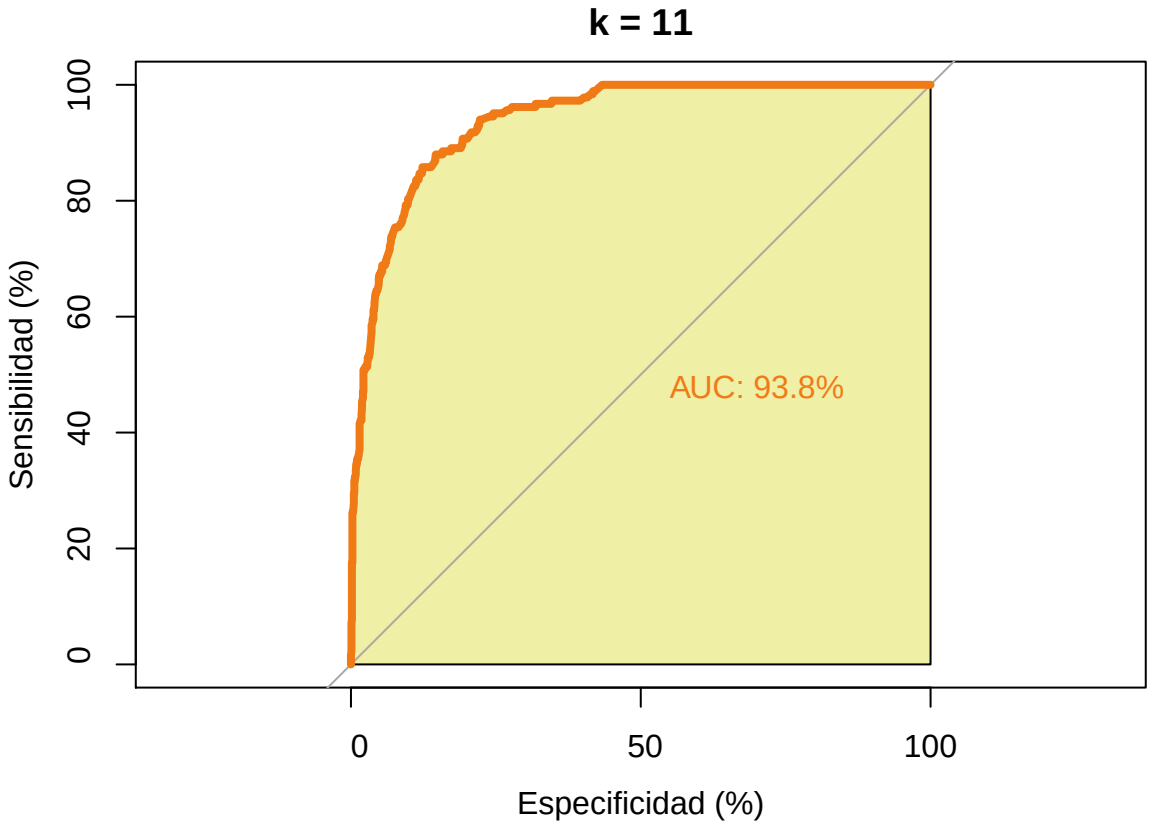
```r
# F-measure
fmes = (2*tp)/(2*tp + fp + fn)
cat("F-measure:", fmes, "\n")
```

```
## F-measure: 0.3090909
```

```r
roc_curve <- roc(data_test_labels, knn_scores, plot = TRUE, legacy.axes = TRUE,
    percent = TRUE, xlab = "Especificidad (%)", ylab = "Sensibilidad (%)",
    col = "#F07A16", lwd = 4, print.auc = TRUE, print.auc.x = 45,
    auc.polygon = TRUE, auc.polygon.col = "#F0EFA6", auc = TRUE,
    main = "k = 11")
```

```
## Setting levels: control = uncleaved, case = cleaved
```

```
## Setting direction: controls < cases
```




```
# AUC
roc_curve$auc
```

```
## Area under the curve: 93.85%
```

Discusión resultados

A continuación se muestra una tabla con algunos de los resultados más significativos para cada k, mostrados en porcentaje (%).

k	AUC	Accuracy	Error rate	Sensivity	Specificity	FP	FN
3	93.5	91.3	8.7	46.9	98	24	97
5	93.11	90.1	9.8	32.8	98.8	14	123
7	93.7	89.3	10.7	24.04	99.2	10	139
11	93.85	89.1	10.9	18.6	99.8	3	149

Nos interesa poder predecir con la mayor precisión si un octámero va a ser o no escindido por la proteasa HIV-1. Para ello necesitamos un valor de k para el cual el porcentaje de predicción de positivos correctamente sea máximo, no haya un gran número de falsos positivos, presente una buena tasa de error, etc.

Para las cuatro k posibles que hemos probado, se podría decir que el algoritmo k-nn no es óptimo en nuestra predicción con ninguna de las k. Con k = 3, por ejemplo, es con la que se consigue mejor tasa de éxito y mejor sensibilidad (clasificación correcta de positivos), así como también se consigue la menor tasa de error. Aún así, la clasificación no es muy precisa, ya que no se consigue clasificar con éxito ni el 50% de los datos. Además, es el valor de k con el que hay un mayor número de falsos positivos.

En cambio, con el mayor valor de k probado, k = 11, obtenemos el menor número de falsos positivos y una especificidad de casi el 100%, pero a cambio obtenemos una sensibilidad muy baja, del 18.6%, y el mayor

número de falsos negativos.

El Área bajo la curva o AUC es similar en todos los casos, varía entre 93.11 (k = 5) y 93.85 (k = 11), por lo que todos los valores se encuentran en el grado B, es decir, excelente o bueno.

Por tanto, tras observar los datos podríamos decir que parece que este algoritmo no es un buen algoritmo para predecir con precisión si un octámero va a ser o no escindido por la proteasa HIV-1.

Step 5: Mejora del modelo

Para mejorar el modelo podemos probar con otros valores de k. Si fuera otro tipo de datos numéricos podría realizarse también una estandarización z-score, pero no en este caso. Probaremos con k = 1 y k = 9.

k = 1

```
k = 1
pred_test_k <- knn(train = data_train, test = data_test,
                cl = data_train_labels, k = k, prob = TRUE)

# Almacenamos los valores de probabilidad de cada predicción obtenida
knn_scores <- attr(pred_test_k, "prob")
```

```
# Calculamos la probabilidad de ser clase ganadora
# Cambiamos la p(no ser clase ganadora) por
# 1 - p(ser clase ganadora)
knn_scores[pred_test_k == "uncleaved"] <- 1 - knn_scores[pred_test_k == "uncleaved"]
```

```
# Matriz confusión
conf_matrix <- CrossTable(x = data_test_labels, y = pred_test_k, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1392
##
##
##                  | pred_test_k
## data_test_labels | uncleaved |   cleaved | Row Total |
## -----------------|-----------|-----------|-----------|
##        uncleaved |      1184 |        25 |      1209 |
##                  |     0.979 |     0.021 |     0.869 |
##                  |     0.974 |     0.142 |           |
##                  |     0.851 |     0.018 |           |
## -----------------|-----------|-----------|-----------|
##          cleaved |        32 |       151 |       183 |
##                  |     0.175 |     0.825 |     0.131 |
##                  |     0.026 |     0.858 |           |
```

```
##                  |     0.023 |     0.108 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |      1216 |       176 |      1392 |
##                  |     0.874 |     0.126 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

```r
# Almacenamos los valores de la matriz de confusión
tp <- conf_matrix$t[4]
tn <- conf_matrix$t[1]
fp <- conf_matrix$t[3]
fn <- conf_matrix$t[2]

# Accuracy
ac = (tp + tn)/(tp+tn+fp+fn)
cat("Accuracy:", ac, "\n")
```

```
## Accuracy: 0.9590517
```

```r
# Error rate
er = 1 - ac
cat("Error Rate:", er, "\n")
```

```
## Error Rate: 0.04094828
```

```r
# Sensivity
sen = tp/(tp+fn)
cat("Sensivity:", sen, "\n")
```

```
## Sensivity: 0.8251366
```

```r
# Specificity
spec = tn/(tn+fp)
cat("Specificity:", spec, "\n")
```

```
## Specificity: 0.9793218
```

```r
# Precision
prec = tp/(tp + fp)
cat("Precision:", prec, "\n")
```

```
## Precision: 0.8579545
```

```r
# Recall
rec = tp/(tp + fn)
cat("Recall:", rec, "\n")
```

```
## Recall: 0.8251366
```

```r
# F-measure
fmes = (2*tp)/(2*tp + fp + fn)
cat("F-measure:", fmes, "\n")
```

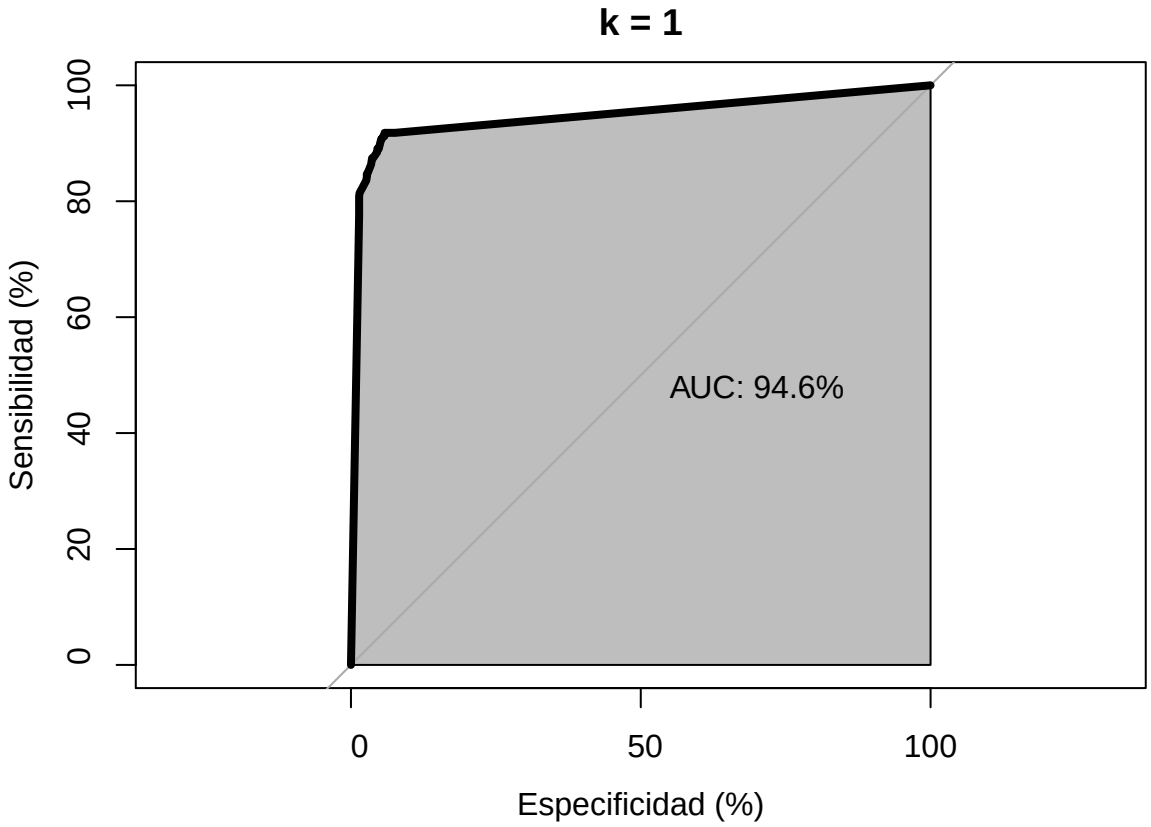
```
## F-measure: 0.8412256
```

```r
roc_curve <- roc(data_test_labels, knn_scores, plot = TRUE, legacy.axes = TRUE,
    percent = TRUE, xlab = "Especificidad (%)", ylab = "Sensibilidad (%)",
    col = "black", lwd = 4, print.auc = TRUE, print.auc.x = 45,
    auc.polygon = TRUE, auc.polygon.col = "grey", auc = TRUE,
```

```
    main = "k = 1")
```

```
## Setting levels: control = uncleaved, case = cleaved
```

```
## Setting direction: controls < cases
```




```
# AUC
roc_curve$auc
```

```
## Area under the curve: 94.61%
```

Con k = 1, aunque mejore algún parámetro, obtenemos un mayor número de falsos positivos, con lo que no es el valor de k que buscamos. Con este valor de k el solape entre las clases o ruido tiene mucha influencia.

k = 9

```
k = 9
pred_test_k <- knn(train = data_train, test = data_test,
                   cl = data_train_labels, k = k, prob = TRUE)

# Almacenamos los valores de probabilidad de cada predicción obtenida
knn_scores <- attr(pred_test_k, "prob")
```

```
# Calculamos la probabilidad de ser clase ganadora
# Cambiamos la p(no ser clase ganadora) por
# 1 - p(ser clase ganadora)
knn_scores[pred_test_k == "uncleaved"] <- 1 - knn_scores[pred_test_k == "uncleaved"]
```

```r
# Matriz confusión
conf_matrix <- CrossTable(x = data_test_labels, y = pred_test_k, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1392
##
##
##                  | pred_test_k
## data_test_labels | uncleaved |   cleaved | Row Total |
## -----------------|-----------|-----------|-----------|
##        uncleaved |      1203 |         6 |      1209 |
##                  |     0.995 |     0.005 |     0.869 |
##                  |     0.892 |     0.140 |           |
##                  |     0.864 |     0.004 |           |
## -----------------|-----------|-----------|-----------|
##          cleaved |       146 |        37 |       183 |
##                  |     0.798 |     0.202 |     0.131 |
##                  |     0.108 |     0.860 |           |
##                  |     0.105 |     0.027 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |      1349 |        43 |      1392 |
##                  |     0.969 |     0.031 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

```r
# Almacenamos los valores de la matriz de confusión
tp <- conf_matrix$t[4]
tn <- conf_matrix$t[1]
fp <- conf_matrix$t[3]
fn <- conf_matrix$t[2]

# Accuracy
ac = (tp + tn)/(tp+tn+fp+fn)
cat("Accuracy:", ac, "\n")
```

```
## Accuracy: 0.8908046
```

```r
# Error rate
er = 1 - ac
cat("Error Rate:", er, "\n")
```

```
## Error Rate: 0.1091954
```

```r
# Sensivity
sen = tp/(tp+fn)
```

```r
cat("Sensivity:", sen, "\n")
```

```
## Sensivity: 0.2021858
```

```r
# Specificity
spec = tn/(tn+fp)
cat("Specificity:", spec, "\n")
```

```
## Specificity: 0.9950372
```

```r
# Precision
prec = tp/(tp + fp)
cat("Precision:", prec, "\n")
```

```
## Precision: 0.8604651
```

```r
# Recall
rec = tp/(tp + fn)
cat("Recall:", rec, "\n")
```

```
## Recall: 0.2021858
```

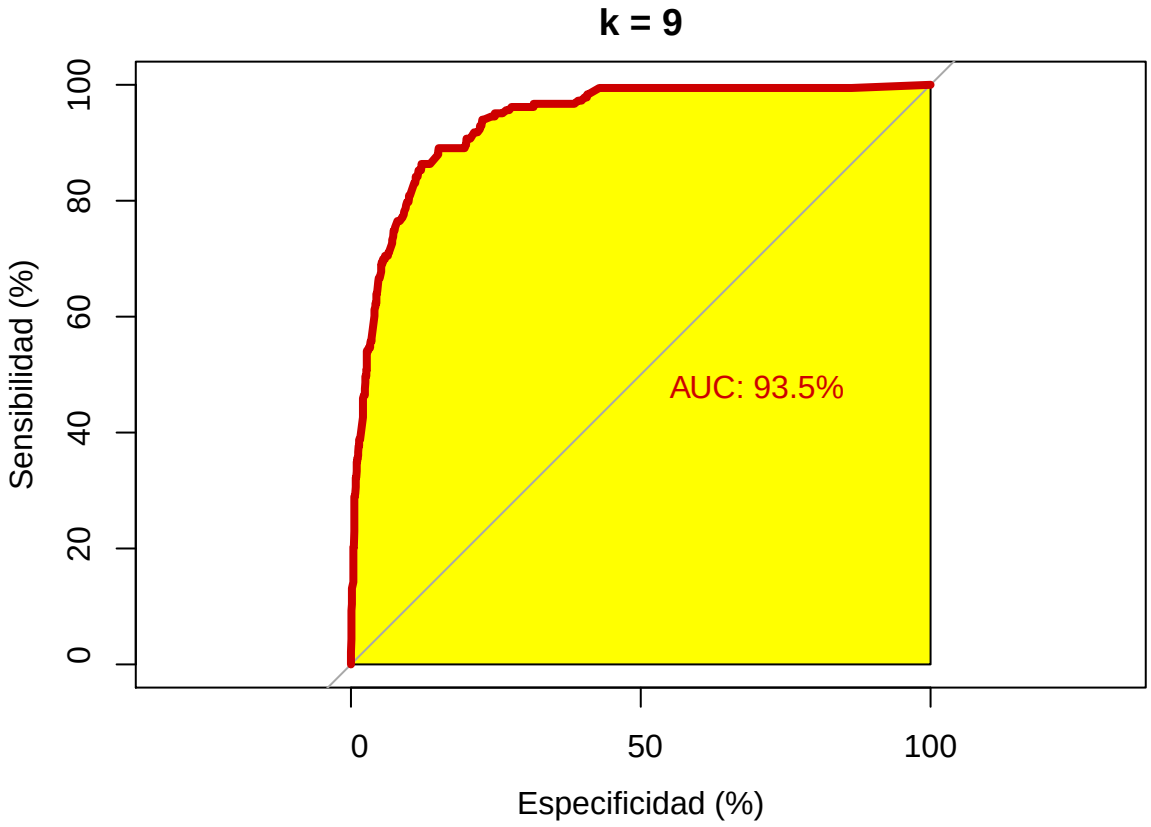
```r
# F-measure
fmes = (2*tp)/(2*tp + fp + fn)
cat("F-measure:", fmes, "\n")
```

```
## F-measure: 0.3274336
```

```r
roc_curve <- roc(data_test_labels, knn_scores, plot = TRUE, legacy.axes = TRUE,
    percent = TRUE, xlab = "Especificidad (%)", ylab = "Sensibilidad (%)",
    col = "red3", lwd = 4, print.auc = TRUE, print.auc.x = 45,
    auc.polygon = TRUE, auc.polygon.col = "yellow", auc = TRUE,
    main = "k = 9")
```

```
## Setting levels: control = uncleaved, case = cleaved
```

```
## Setting direction: controls < cases
```




```
# AUC
roc_curve$auc
```

```
## Area under the curve: 93.5%
```

Con k = 9, aunque mejore algún parámetro, obtenemos un mayor número de falsos negativos, con lo que tampoco es el valor de k que buscamos. Conforme aumenta el valor de k, puede perderse la idea de localidad.

En conclusión, como se ha comentado en la discusión de los resultados del Step anterior, no creo que este sea un buen algoritmo para la clasificación de estos datos en concreto, puesto que no podemos tomar los datos predichos como buenos con total seguridad.

Referencias

Asuncion, Arthur, and David Newman. 2007. "UCI Machine Learning Repository."

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

Soucy, Pascal, and Guy W Mineau. 2001. "A Simple Knn Algorithm for Text Categorization." In *Proceedings 2001 Ieee International Conference on Data Mining*, 647–48. IEEE.

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