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

Escribir vuestro nombre y apellidos

14 de octubre, 2020

## Índice



```
# CRAN repository

libraries <- c("class", "gmodels",  "knitr","pROC", "ROCR","devtools")
check.libraries <- is.element(libraries, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries[check.libraries]
if (length(libraries.to.install!=0)) {
  install.packages(libraries.to.install)
}

for (i in libraries) {
  library(i, character.only = TRUE)
}


# Bioconductor repository
libraries.bct <- c("Biostrings")
check.libraries <- is.element(libraries.bct, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries.bct[check.libraries]
```

```r
if (length(libraries.to.install!=0)) {
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(libraries.to.install, version = BiocManager::version())
}

for (i in libraries.bct) {
  library(i, character.only = TRUE)
}

# Github repository

libraries <- c("ggseqlogo")
check.libraries <- is.element(libraries, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries[check.libraries]
if (length(libraries.to.install!=0)) {
  devtools::install_github("omarwagih/ggseqlogo")
}

for (i in libraries) {
  library(i, character.only = TRUE)
}
```

## El Algoritmo kNN

El algoritmo de kNN aplica el enfoque de los vecinos más próximos para clasificar un nuevo individuo. Las fortalezas y debilidades de este algoritmo son:

| Fortalezas | Debilidades |
|---|---|
| * Simple y efectivo | * No produce un modelo, lo que limita la capacidad de encontrar nuevos conocimientos en las relaciones entre las características |
| * No hace suposiciones sobre la distribución de datos subyacente | * Fase de clasificación lenta |
| * Fase de entrenamiento rápida | * Requiere una gran cantidad de memoria |
| | * Variables nominales y los datos faltantes requieren un procesamiento adicional |

El algoritmo kNN comienza con un conjunto de datos de entrenamiento donde se conoce sus categorias, es decir, sus etiquetas. Por otra parte, tenemos un conjunto de datos de prueba que contiene ejemplos no etiquetados del mismo tipo que los datos de entrenamiento. Para cada registro en el conjunto de datos de prueba, kNN identifica `k` registros en los datos de entrenamiento que son los "más cercanos" en similitud, donde `k` es un valor positivo especificado de antemano. Al registro no etiquetado se le asigna la clase de la mayoría de los `k` vecinos más cercanos.

En resumen, en este algoritmo de clasificación se tiene que elegir la **distancia** que servirá para comparar los ejemplos y el valor de la `k` para comparar los vecinos más cercanos. Estos parámetros son cruciales para tener un buen rendimiento del clasificador.

## Predicción in-silico de sitios de escisión reconocidos por la proteasa del HIV-1

En esta PEC vamos a realizar un informe que analiza un caso basado en los datos del artículo:

**State of the art prediction of HIV-1 protease cleavage sites. Rögnvaldsson et al.Bioinformatics, 2015, 1-7. doi: 10.1093/bioinformatics/btu810**

Los datos se pueden obtener directamente de la revista Bioinformatics o del dataset "HIV-1 protease cleavage data" desde la *UCI Machine Learning Repository*, disponible en http://archive.ics.uci.edu/ml.

En dicho artículo se investiga la predicción in-silico de los sitios en las proteinas que son reconocidos para la escisión (cleavage) por la proteasa del HIV-1.

*There are two different approaches to predicting cleavage by HIV-1 protease: molecular modeling and sequence analysis. It has been argued that the HIV-1 protease recognizes shape rather than a specific amino acid sequence (Prabu-Jeybalan et al., 2002), which supports aiming for the molecular modeling approach. However, the method is cumbersome and no large scale study has been done on the accuracy of molecular modeling approaches so it is very unclear if the approach is, or will be, competitive with the sequence based approach. This article demonstrates the current state-of-theart prediction, which uses the sequence-based approach.*

En particular, en articulo de Rögnvaldsson et al.se centra en la tarea de predecir si dado un octamero (secuencia de 8 aminoacidos), éste será o no reconocido por la proteasa.

*The HIV-1 cleavage problem is described in detail in (Rögnvaldsson et al., 2007) together with discussions on different encoding schemes. Only a concise description is given here. The classification task is to tell whether a given octamer (sequence of eight amino acids) will be cleaved or not between the fourth and the fifth position.*

La manera elegida para representar los datos es un paso crucial en los algoritmos de clasificación. En el caso que nos ocupa, análisis basados en secuencias, usaremos el mismo tipo de representación que los autores emplearon en el su estudio.

*The octamer is represented using an orthogonal encoding where each amino acid is represented by a 20-bit vector with 19 bits set to zero and one bit set to one (other encodings have been suggested, see later). This maps each octamer to an 8 by 20 binary matrix that is transformed into a 160-dimensional vector.*

En la PEC se implementará un algoritmo `knn` para predecir aquellos octameros que son sustrato para de la proteasa del HIV-1.

# Resolución apartado 3

## Función para implementar la codificación ortogonal

Este seria una de las maneras de programar la codificación binaria de los octameros. Seguro que hay mejores.

```
################## data transformation
# There are 20 allowed letters in the 8 character string
# (the allowed alphabet): 'ARNDCQEGHILKMFPSTWYV'

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

cod.ort <-function(x, alphabet = alphabet){
y <- unlist(strsplit(x,""))
sapply(y,function(x){match(alphabet,x,nomatch=0)})
}
```

En primer lugar se crea la variable `alphabet` que contiene todos los posibles aminoácidos que pueden formar los octameros. Con la función `cod.ort` se consigue pasar un octamero a un codigo binario de 8 filas y 20 columnas.

Para probar su uso, se presenta el siguiente ejemplo:

```
oct <- "AASAAAVD"
t(cod.ort(oct,alphabet))
```

```
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0
A    1    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
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0
V    0    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
  [,17] [,18] [,19] [,20]
A     0     0     0     0
A     0     0     0     0
S     0     0     0     0
A     0     0     0     0
A     0     0     0     0
A     0     0     0     0
V     0     0     0     1
D     0     0     0     0
```

# Step 1 - Recoger los datos

Se pueden bajar la información directamente del articulo indicado en el apartado anterior, además de como dataset "HIV-1 protease cleavage data" desde la *UCI Machine Learning Repository*, disponible en

http://archive.ics.uci.edu/ml.

# Resolución apartado 4

## Step 2 - Exploración y preparación de los datos

```
mydata_1 <- read.csv(file.path(params$folder.data,params$file1), header=FALSE)
mydata_2 <- read.csv(file.path(params$folder.data,params$file2), header=FALSE)
```

El primer conjunto de datos denominado *impensData.txt* esta formado por 947 octameros. El segundo conjunto de datos denominado *schillingData.txt* esta formado por 3272 octameros. En la primera columna está la secuencia y en la segunda columna está la clasificación codificada como -1(no escinde) y 1(escinde).

Lo primero que hay que hacer es fusionar ambos conjuntos de datos. El nuevo `dataframe` se denomina `mydata`.

```
mydata <- rbind(mydata_1,mydata_2)
head(mydata)
```

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

Como no tienen nombre las columnas del dataset original, se usa el siguiente código para añadir su nombre y cambiar la primera columna al tipo `character` y la segunda a tipo `factor`.

```
str(mydata)
```

```
'data.frame':   4219 obs. of  2 variables:
 $ V1: chr  "AAAGKSGG" "AAAVDAGM" "AAGKSGGG" "AALALEYG" ...
 $ V2: int  -1 -1 -1 1 -1 -1 -1 -1 -1 -1 ...
```

```
colnames(mydata)<-c("octamer","cleaved")
mydata$octamer<-as.character(mydata$octamer)
mydata$cleaved<-as.factor(mydata$cleaved)
str(mydata)
```

```
'data.frame':   4219 obs. of  2 variables:
 $ octamer: chr  "AAAGKSGG" "AAAVDAGM" "AAGKSGGG" "AALALEYG" ...
 $ cleaved: Factor w/ 2 levels "-1","1": 1 1 1 2 1 1 1 1 1 1 ...
```

Una información interesante es la cantidad de octameros de cada clase. En nuestro caso es:

```
table(mydata[,2])
```

```
  -1    1
3636  583
```

### Secuencia logo de cada clase

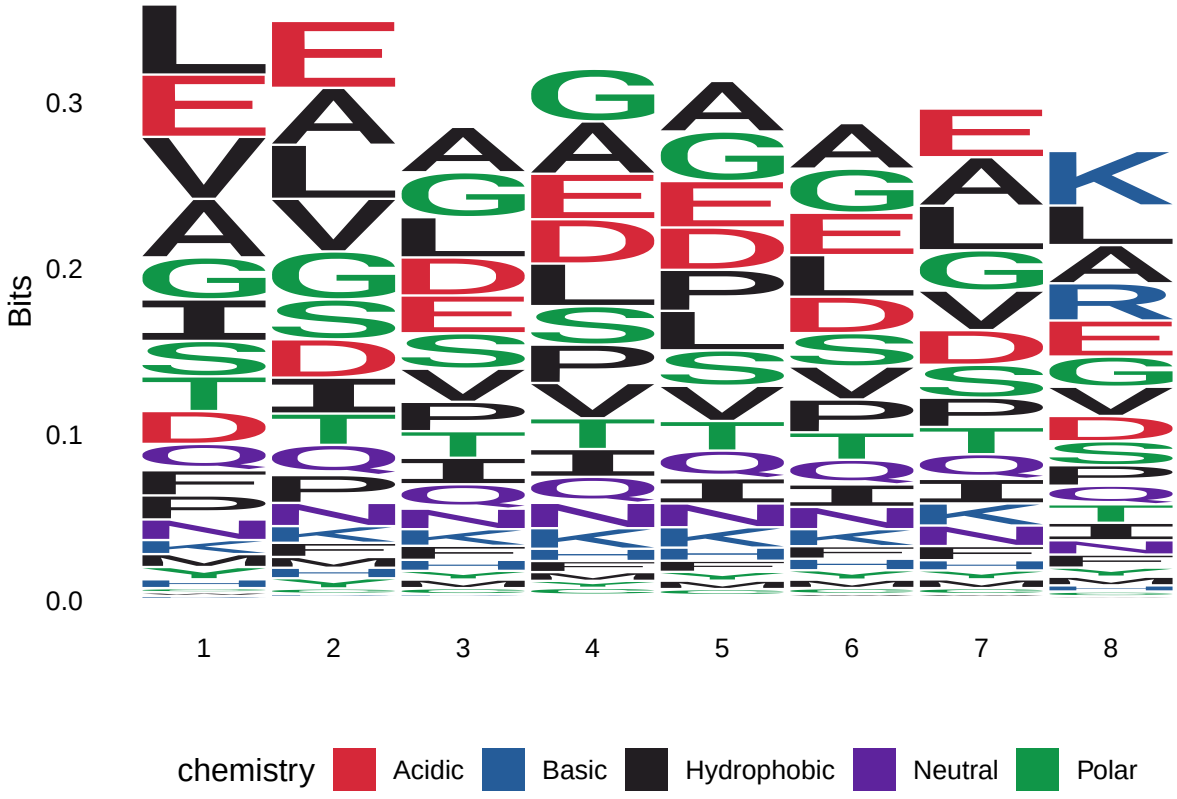
Una forma de presentar el patrón de la secuencia de un manera gráfica es realizar una secuencia logo (ver https://en.wikipedia.org/wiki/Sequence_logo)).

"Para crear logos de secuencias, las secuencias relacionadas de ADN, ARN o proteínas, o bien secuencias de ADN que comparten lugares de unión conservados, son alineadas hasta que las partes más conservadas crean buenos alineamientos. Se puede crear entonces un logo de secuencias a partir del alineamiento múltiple de secuencias conservadas. El logo de secuencias pondrá de manifiesto el grado de conservación de los residuos en cada posición: un menor número de residuos diferentes provocará mayor tamaño en las letras, ya que la conservación es mejor en esa posición. Los residuos diferentes en la misma posición se escalarán de acuerdo a su frecuencia. Los logos de secuencias pueden usarse para representar sitios conservados de unión al ADN, donde quedan unidos los factores de transcripción" (extraido de https://es.wikipedia.org/wiki/Logo_de_secuencias)

Para realizar esta representación se usa el paquete `ggseqlogo` descargable desde github.

**Secuencia logo para la clase -1**

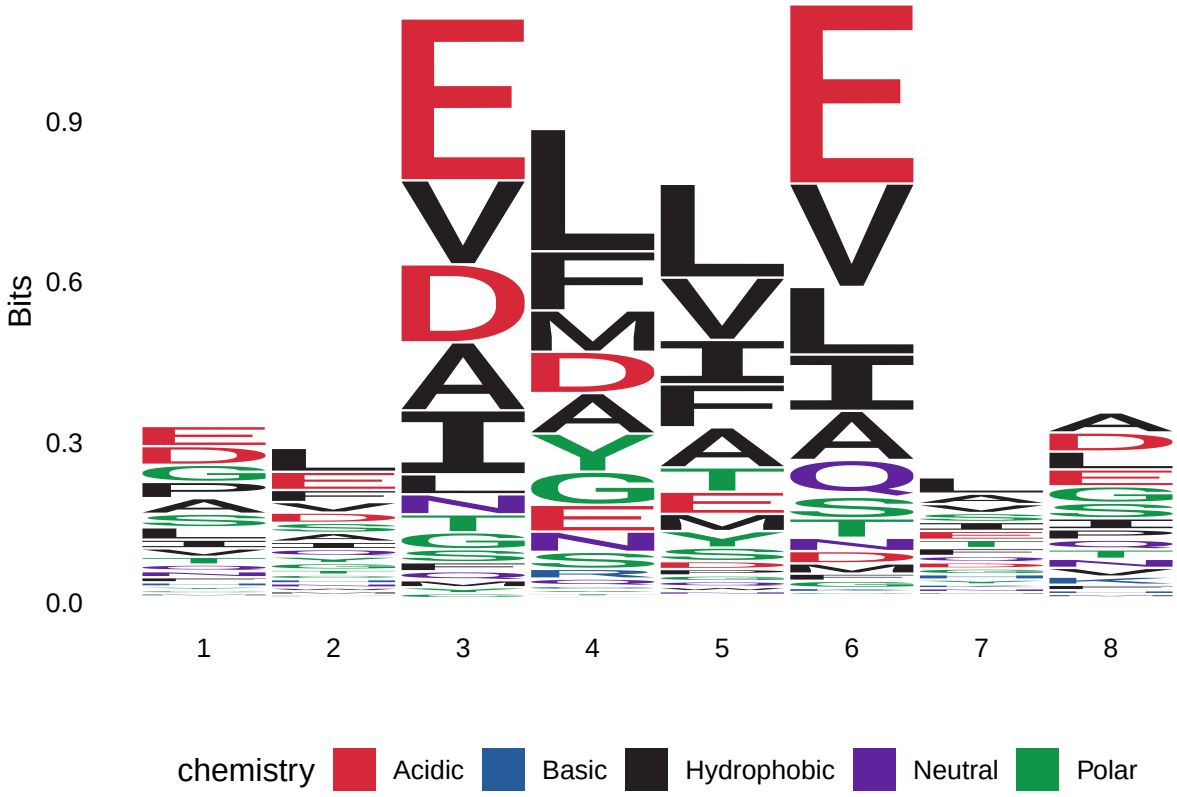
```
seq_select <- mydata[mydata$cleaved==-1,1]
ggseqlogo(seq_select)
```




El patrón mostrado es muy variable, no hay ninguna posición con pocos residuos. Además, los valores obtenidos de bits son bajos.

**Secuencia logo para la clase 1**

```
seq_select <- mydata[mydata$cleaved==1,1]
ggseqlogo(seq_select)
```




En cambio, para esta clase se observa claramente un patrón menos variable en las posiciones de 3 a 6 sobre el resto de posiciones. Además, estas posiciones tienen un valor de bits muy alto respecto al resto de posiciones.

**Transformación - Cambiar los octameros a una codificación binaria**

En primer lugar, se muestra el resultado de la función `cod.ort` para la primera secuencia de la base de datos.

```
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0
G    0    0    0    0    0    0    0    1    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
S    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0     1
G    0    0    0    0    0    0    0    1    0     0     0     0     0     0     0     0
G    0    0    0    0    0    0    0    1    0     0     0     0     0     0     0     0
  [,17] [,18] [,19] [,20]
A     0     0     0     0
A     0     0     0     0
A     0     0     0     0
G     0     0     0     0
K     0     0     0     0
S     0     0     0     0
G     0     0     0     0
G     0     0     0     0
```

La función `apply` en combinación con la función creada `cod.ort` codifica todos los octameros del fichero a la

vez.

```
out<-apply(as.data.frame(mydata[,1]),1,cod.ort,alphabet)
out<-t(out)
```

Ahora se ha creado un nuevo fichero out con el mismo número de filas que octameros, 4219 y con tantas columnas como `8 * 20` ya que para cada aminoácido se codifica en un sistema binario de tamaño 20, y tenemos 8 aminoácidos.

```
dim(out)
```

```
[1] 4219  160
```

```
#head(out)
```

Se muestra las primeras 20 variables de los 6 primeros registros.

```
out[1:6,1:20]
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15]
[1,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
[2,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
[3,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
[4,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
[5,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
[6,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
     [,16] [,17] [,18] [,19] [,20]
[1,]     0     0     0     0     0
[2,]     0     0     0     0     0
[3,]     0     0     0     0     0
[4,]     0     0     0     0     0
[5,]     0     0     0     0     0
[6,]     0     0     0     0     0
```

Por ultimo, solo **centraremos** los datos transformados aplicando la función `scale`.

```
# center data
out<-scale(out,center=T,scale=F)
#head(out)
out[1:6,1:20]
```

```
          [,1]         [,2]        [,3]        [,4]         [,5]        [,6]       [,7]
[1,] 0.9028206 -0.005688552 -0.03460536 -0.06233705 -0.008769851 -0.04337521 -0.1061863
[2,] 0.9028206 -0.005688552 -0.03460536 -0.06233705 -0.008769851 -0.04337521 -0.1061863
[3,] 0.9028206 -0.005688552 -0.03460536 -0.06233705 -0.008769851 -0.04337521 -0.1061863
[4,] 0.9028206 -0.005688552 -0.03460536 -0.06233705 -0.008769851 -0.04337521 -0.1061863
[5,] 0.9028206 -0.005688552 -0.03460536 -0.06233705 -0.008769851 -0.04337521 -0.1061863
[6,] 0.9028206 -0.005688552 -0.03460536 -0.06233705 -0.008769851 -0.04337521 -0.1061863
           [,8]        [,9]       [,10]      [,11]       [,12]       [,13]       [,14]
[1,] -0.07442522 -0.01611756 -0.06707751 -0.1114008 -0.02204314 -0.02109505 -0.04053093
[2,] -0.07442522 -0.01611756 -0.06707751 -0.1114008 -0.02204314 -0.02109505 -0.04053093
[3,] -0.07442522 -0.01611756 -0.06707751 -0.1114008 -0.02204314 -0.02109505 -0.04053093
[4,] -0.07442522 -0.01611756 -0.06707751 -0.1114008 -0.02204314 -0.02109505 -0.04053093
[5,] -0.07442522 -0.01611756 -0.06707751 -0.1114008 -0.02204314 -0.02109505 -0.04053093
[6,] -0.07442522 -0.01611756 -0.06707751 -0.1114008 -0.02204314 -0.02109505 -0.04053093
           [,15]       [,16]       [,17]        [,18]     [,19]       [,20]
[1,] -0.04764162 -0.06210002 -0.05641147 -0.005925575 -0.020621 -0.09646836
[2,] -0.04764162 -0.06210002 -0.05641147 -0.005925575 -0.020621 -0.09646836
[3,] -0.04764162 -0.06210002 -0.05641147 -0.005925575 -0.020621 -0.09646836
```

```
[4,] -0.04764162 -0.06210002 -0.05641147 -0.005925575 -0.020621 -0.09646836
[5,] -0.04764162 -0.06210002 -0.05641147 -0.005925575 -0.020621 -0.09646836
[6,] -0.04764162 -0.06210002 -0.05641147 -0.005925575 -0.020621 -0.09646836
```

**Preparación de los datos - crear el dataset de entrenamiento y de test**

Primero se divide el dataset en una parte de entrenamiento y en otra de test.

```
################## data spliting
set.seed(123) #fijar la semilla para el generador pseudoaleatorio
train<-sample(1:nrow(out),round(2*nrow(out)/3,0))
# create training and test data
out_training<-out[train,]
out_test<-out[-train,]
# create labels for training and test data
class_training<-mydata[train,2]
class_test<-mydata[-train,2]
```

## Step 3 - Entrenar el modelo con los datos

Con la función `knn` del package class se obtiene la predicción para un `k` fijado.

```
############### libraries loading
#library(class) # knn
############## data prediction
test_pred <- knn(train =out_training, test = out_test, cl = class_training, k=3)
```

## Step 4 - Evaluación del rendimiento del algoritmo

Se compara las predicciones del algoritmo kNN con las categorias reales. Se usa la función `CrossTable` del package `gmodels`

```
# load the "gmodels" library
#library(gmodels)
# Create the cross tabulation of predicted vs. actual
############ evaluating model performance
CrossTable(x = class_test, y = test_pred , prop.chisq=FALSE)
```

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


Total Observations in Table:  1406


             | test_pred
  class_test |        -1 |         1 | Row Total |
-------------|-----------|-----------|-----------|
          -1 |      1190 |        20 |      1210 |
```

```
             |     0.983 |     0.017 |     0.861 |
             |     0.892 |     0.278 |           |
             |     0.846 |     0.014 |           |
-------------|-----------|-----------|-----------|
           1 |       144 |        52 |       196 |
             |     0.735 |     0.265 |     0.139 |
             |     0.108 |     0.722 |           |
             |     0.102 |     0.037 |           |
-------------|-----------|-----------|-----------|
Column Total |      1334 |        72 |      1406 |
             |     0.949 |     0.051 |           |
-------------|-----------|-----------|-----------|
```

```
table(class_test)
```

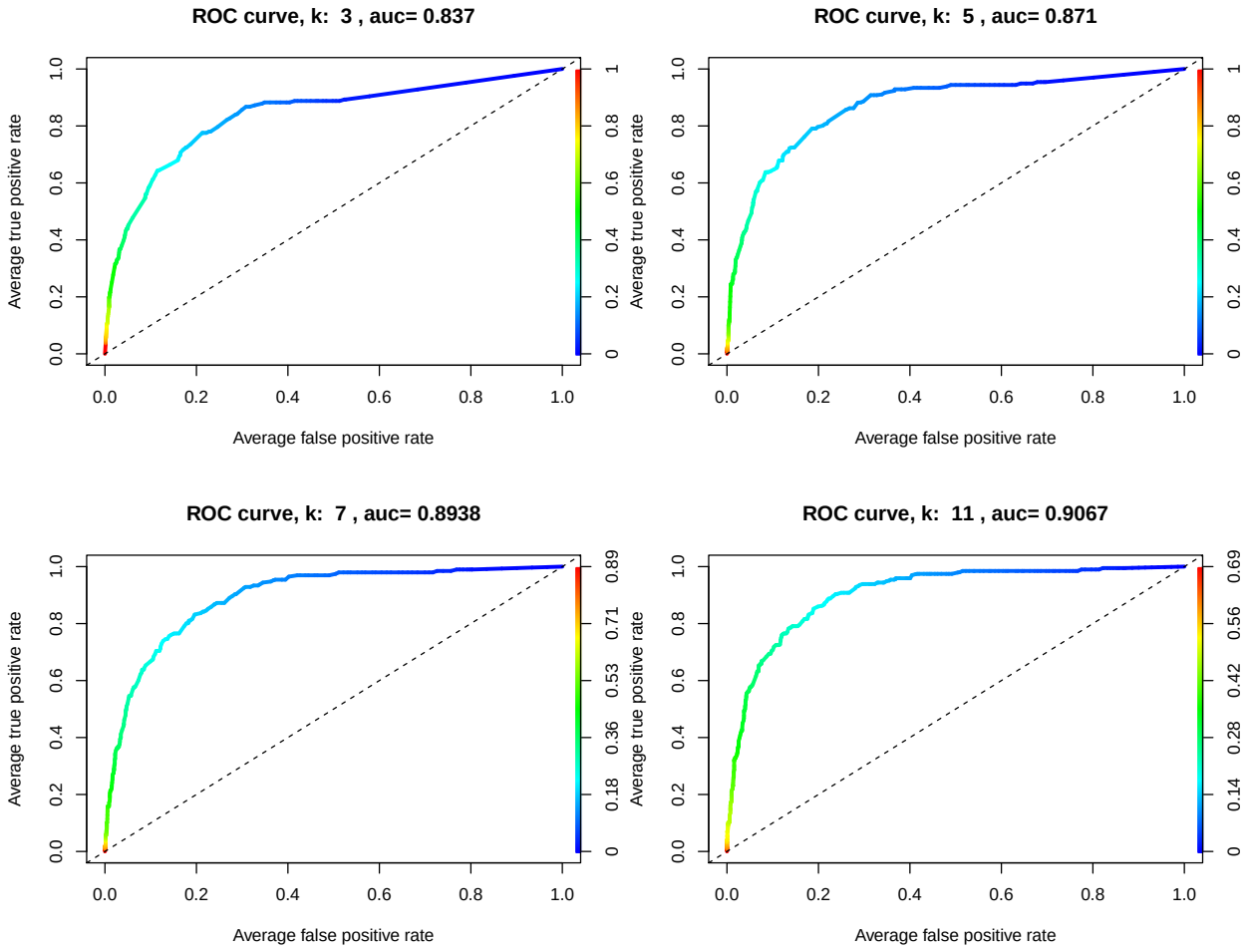
```
class_test
  -1    1
1210  196
```

### Curva ROC del clasificador kNN para diversos valores de k

Para cada valor de `k` se necesita obtener las probabilidades de la clase positiva, en este caso es la clase `1`, de la predicción con los datos de test. Se usa el argumento `prob` de la función `knn()` que devuelve la probabilidad de la clase ganadora para su calculo. El siguiente paso es representar la curva ROC donde también se ha añadido el valor de AUC.

```
# try several different values of k

ks <- c(3,5,7,11)
vec.auc <- NULL
par(mfrow=c(2,2))
for (i in ks){
  test_pred <- knn(train =out_training, test = out_test, cl = class_training, k=i, prob= TRUE)

  prob <- attr(test_pred, "prob")
  # convert the proportion of the votes for the winning class to p(test_pred == 1)
  prob1 <- ifelse(test_pred == 1, prob, 1-prob)

  # res <- auc(class_test,prob1)
  # Setting levels: control = 1, case = -1
  # Setting direction: controls < cases

  pred_knn <- ROCR::prediction(predictions= prob1,
                               labels= class_test,
                               label.ordering = c("-1","1"))
  perf <- performance(pred_knn, "tpr", "fpr")
  perf.auc <- performance(pred_knn, measure="auc")
  perf.auc <- unlist(perf.auc@y.values)
  vec.auc <- c(vec.auc,perf.auc)

  plot(perf, avg= "threshold", colorize=T, lwd=3,
       main=paste("ROC curve, k: ", i,
                  ", auc=", round(perf.auc,4)))
  abline(a = 0, b = 1, lwd = 1, lty = 2)
```

```
}
```




```
par(mfrow=c(1,1))
```

Los resultados de la curvas ROC estan acorde con los valores de rendimiento presentados en la tabla anterior.

## Evaluación del clasificador kNN para varios valores de k

Como hay que evaluar el algoritmo kNN para diferentes valores de `k`, en particular seran 3, 5, 7, 11 con los mismos datos de entrenamiento y test se crea un bucle. Como criterios para evaluar el rendimiento en cada valor de `k` se obtiene el número de falsos positivos, falsos negativos, porcentaje de mal clasificados y AUC

```
#require(knitr, quietly = TRUE)
kable(resum, col.names=c("valor k", "# falsos negativos",
"# falsos positivos", "% mal clasificados", "AUC"),
align= c("l","c","c","c", "c"), caption= paste("Algoritmo kNN: ",
params$file1, " + ", params$file2 ,sep=""))
```

Cuadro 2: Algoritmo kNN: impensData.txt + schillingData.txt

| valor k | # falsos negativos | # falsos positivos | % mal clasificados | AUC |
|---|---|---|---|---|
| 3 | 142 | 20 | 11.52205 | 0.8369814 |

| valor k | # falsos negativos | # falsos positivos | % mal clasificados | AUC |
|---|---|---|---|---|
| 5 | 162 | 8 | 12.09104 | 0.8709711 |
| 7 | 170 | 6 | 12.51778 | 0.8937827 |
| 11 | 179 | 3 | 12.94452 | 0.9066685 |

Entre los 4 valores de `k` evaluados el que tiene un menor porcentaje de clasificación erronea es el valor **3** con un valor de **11.52205** % mal clasificados. El rango de % mal clasificados está entre *11.52205* y *12.94452*. Así que la diferencia entre el valor mínimo y máximo es *1.42248* y por tanto, es pequeña la diferencia.

Observar que el promedio de falsos negativos es *163.25* y su desviacion estandard *15.78*. Para el caso de falsos positivos es *9.25* y su desviacion estandard *7.46*. En promedio hay más *falsos negativos* con una relación de *23* veces respecto al menor tipo de error.

Respecto al AUC, el valor de `k` con mayor AUC es **11** con un AUC de **0.9067**. El rango de valores de AUC está entre *0.837* y *0.9067*. Así que, la diferencia entre el valor mínimo y máximo es *0.0697* y por tanto, es pequeña la diferencia.

# Referencias

Lantz, Brett. 2015. *Machine learning with R*. Packt Publishing Ltd. https://www.packtpub.com/books/content/machine-learning-r.

Rögnvaldsson, Thorsteinn, Liwen You, y Daniel Garwicz. 2015. «State of the art prediction of HIV-1 protease cleavage sites». *Bioinformatics* 31 (8): 1204-10. https://doi.org/10.1093/bioinformatics/btu810.