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

Pablo Iriso Soret

Contents

22 de febrero, 2021

Fundamento teórico. Proteasa VIH-1

La proteasa VIH-1 es una enzima pertenciente al virus de la inmunodeficiencia humana (VIH). Se trata de una enzima de la familia de las aspartil proteasas, también conocidas como ácido proteasas, caracterizadas por tener muy conservadas las secuencias Asp-Gly-Thr. Las proteasas son enzimas que rompen enlaces peptídicos entre los aminoácidos de las proteínas, utilizando una molécula de agua. Este proceso es conocido como corte proteolítico. Las proteínas formadas por el Virus de la Inmunodeficiencia Humana (VIH) se sintetizan como precursores de largas cadenas proteicas que deben ser cortadas para dar lugar a los componentes proteicos activos del virus maduro, en esta fase intervienenn la proteasas VIH 1. Wlodawer and Vondrasek (1998)

Algoritmo k-NN

El algoritmo K-nearest neighbors (kNN - k vecinos más cercanos) se encuentra entre las diez técnicas más empleadas en el *data mining*. Este método utiliza el principio de Ciceron *"pares cum paribus facillime congregantur"* (pájaros del mismo plumaje vuelan juntos o, literalmente, iguales con iguales se asocian fácilmente), y pertenece al conjunto de algoritmos de clasificación.

Un algoritmo de clasificación permite la identificación de la categoría a la que pertenece un objeto concreto. De tal forma que, a partir de un conjunto de observaciones $(x_1, y_1), ...(x_n, y_n)$, donde y es la categoría a la que pertenece la muestra, y x, un vector de características correspondientes a dicha muestra, los algoritmos de clasificación nos van a permitir determinar dicha categoría para un objeto del que poseamos únicamente sus caracterítsicas. Kramer (2013)

Para este método concreto, dicha asignación se basará en la cercanía de nuestro nuevo objeto, al resto de observaciones más cercanas, y le otorga una clase basado en la mayoría de los datos que le rodean (y las clases que presentan dichos datos). Será por lo tanto necesario precisar, en primer lugar, que valor de k vamos a escoger, es decir, en base a que número de vecinos próximos estableceremos la clasificación, y en segundo lugar, como vamos a representar computacionalmente dichas distancias.

El algoritmo k-NN se denomina habitualmente como clasificador "vago," dado que técnicamente no genera un clasificador a partir de los datos de entrenamiento, si no que cada vez que quiere asignar una clase a un nuevo objeto, calcula las distancias para dicho objeto a partir de los datos de entrenamiento. Esto provoca que sea costoso computacionalmente. Mucherino, Papajorgji, and Pardalos (2009)

Algoritmo kNN basico:

- **Input** : Presenta los siguientes componentes, D, el set de entrenamiento, formado por un conjunto de objetos; z, que es el objeto al que queremos asignar una clase y viene definido por un vector de valores; y L, el conjunto de clases para los objetos.
- **Output**: $c_z \in L$, la clase de z.
  Mucherino, Papajorgji, and Pardalos (2009)

Tabla de fortalezas y desventajas del modelo

Fortalezas	Debilidades
No tiene periodo de aprendizaje	No produce un modelo. Dificulta comprensión de la relación entre los datos. Costoso computacionalmente
Se pueden agregar nuevos datos sin problemas	Sensible a la información ruidosa, valores perdidos y "outliers"
Es fácil de implementar	El rendimiento diseminuye conforme aumentan las dimensiones
Se pueden agregar nuevos datos sin problemas	Es necesaria una normalización de los datos

(Lantz 2013)

Carga de datos y preparación para la codificación Ortogonal

```
file1 <- "ort_enc.csv"
file2 <- "schillingData.txt"
file3 <- "impensData.txt"


ort <- read.csv(file1)
res1 <- read.csv(file2, header = F)
res2 <- read.csv(file3, header = F)
```

A la hora de implementar un algoritmo es necesario encontrar una representación de los datos que nos permita extraer conclusiones. Para los octámeros se va a utilizar una representación ortogonal (otro tipo de representaciones son posibles, como OETMAP y GP1). En este caso, cada aminoácido será representado como una combinación de bits, donde habrá 19 bits igualados a 0 y unn bit igualado a 1. Cada uno de los 20 aminoácidos naturales se corresponde con una combinación de bits única.

En primer lugar, asignaremos a cada aminoácido una combinación de bits. Además, diseñaremos una función que asignará a cada octámero un vector de 160 valores, que se corresponderá con los 160 bits para dicho octámero.

```
#Para crear esta función, definiremos una combinación de bits para cada octámero, y haremos una asociac

A <- c(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
R <- c(0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
N <- c(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
D <- c(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
C <- c(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
Q <- c(0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
E <- c(0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0)
G <- c(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0)
H <- c(0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0)
I <- c(0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0)
L <- c(0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0)
K <- c(0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0)
```

```
M <- c(0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0)
F <- c(0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0)
P <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0)
S <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0)
T <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0)
W <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0)
Y <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0)
V <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)
sigla <- c("A","R","N","D","C","Q","E","G","H","I","L","K","M","F","P","S","T","W","Y","V")

#Creamos una matriz y la convertimos en un dataframe
aminoacidos <- rbind(A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V)
aminoacidos <- cbind(sigla, aminoacidos)
aminoacidos <- as.data.frame(aminoacidos)


#Creamos una función que nos proporciona un vector con los valores en bits del octámero.
cod_o <- function(sequence){
  unlist(asplit(aminoacidos[na.omit(match(strsplit(sequence, "")[[1]], aminoacidos$sigla)), -1], 1), us
}

#Unimos esta función a una nueva función que es capaz de iterar dicha función,
#creando un data frame con el octámero y su valor.
generador <- function(datos){
  bits <- t(as.data.frame(lapply(datos$V1, cod_o)))
  row.names(bits) <- datos$V1
  return(bits)
}
```

Script en R para llevar a cabo k-NN

Step 1: Obtención y presentación de los datos

Leer los datos *impensData.txt* y *schillingData.txt* Crear un nuevo conjunto de datos quesea la unión de ambos y hacer una breve descripción de los datos. Incluir en esta descripción el patrón de cada clase de octamero mediante la representación de su secuencia logo

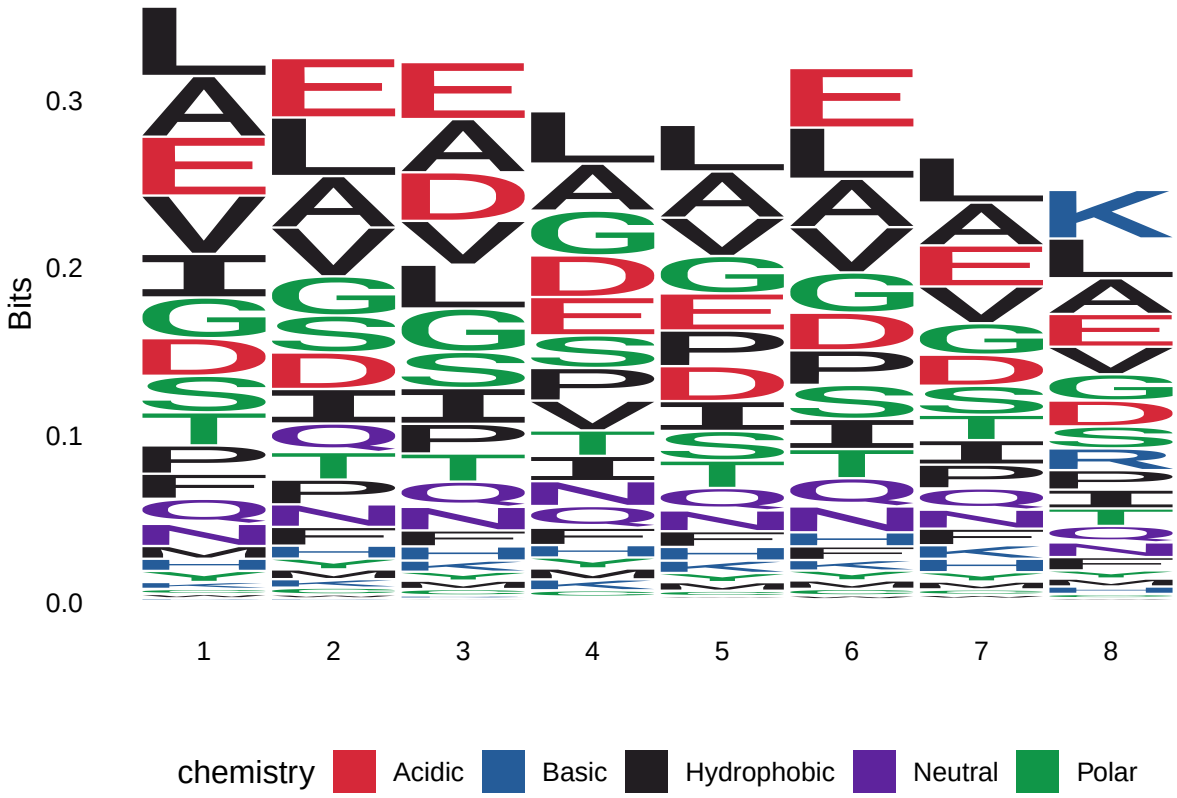
```
secuencias <- rbind(res1, res2)

colnames(secuencias) <- c("Octamero","Valor")
nrowsecuencias <- nrow(secuencias)
ncolsecuencias <- ncol(secuencias)
head(secuencias)
```

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

El *dataframe* esta compuesto por 4219 muestras, con 2 columnas, la primera de ellas correpondiendo con la secuencia octamérica, y la segunda con el posible resultado: 1, 0; octámero reconocible por la proteasa y no reconocile respectivamente.

La siguiente función nos permite visualizar el *seqlogo* para los diferentes octámeros. A continuación hacemos una representación del conjunto de octámeros contenido en *schillingData* a modo de demostración. En el último apartado del informe se emplea esta función para visualizar los resultados.

```
ggseqlogo(res1$V1)
```



Datos originales

Los siguientes apartados se van a realizar dos veces. Vamos a reproducir el algoritmo una primera vez, en la que mostraremos el procedimiento y los resultados obtenidos para los datos originales (proporcionados por la PEC), y posteriormente con las codificaciones obtenidas a partir de nuestras funciones. Esto nos permitirá comparar los resultados para ambos *dataframes* y observar si se asemejan o no.

Step 2: Preparación de los datos (Codificación ortogonal)

Como hemos comentado, para este primer apartado no vamos a llevar a cabo la codificación (lo haremos para los datos generados por nuestra función), si no que reproduciremos nuestro algoritmo sobre los datos proporcionados por la PEC. Lo primero que hacemos es llevar a cabo un tratamiento de los datos:

```r
#Creamos un dataframe que contiene, el nombre del octámero, el valor observado, y su codificación en bi
datos <- cbind(secuencias, ort)
datos <- select(datos, -Octamero)
```

Step 3: Separación de datos

A continuación, mediante la fijación de la aleatoriedad (*semilla aleatoria 123*), vamos a dividir nuestros datos en dos partes, una parte para *training* (67%) y una parte para *test* (33%)

```r
set.seed(123)

#Para ordenarlo de forma aleatoria creamos una distribución aleatoria de todas las columnas, y posterio
rows <- sample(nrow(datos))
datos_n <- datos[rows, ]


#La expresión round(0.67*nrow(ort)), nos proporciona el 67% de los valores por orden.
datos_train <- datos_n[1:round(0.67*nrow(datos_n)), ]
datos_test <- datos_n[round(0.67*nrow(datos_n)):length(datos_n$Valor),]


datos_train_labels <- datos_n[1:round(0.67*nrow(datos_n)), 2]
datos_test_labels <- datos_n[round(0.67*nrow(datos_n)):length(datos_n$Valor), 2]
```

Step 4: Implementación de kNN y curvas ROC

(d) Utilizar un knn (k = 3, 5, 7, 11) basado en el training para predecir que octameros del test tienen o no cleavage site. Además, realizar una curva ROC para cada k y calcular su área bajo la curva (AUC).

```r
num_k3 <- 3

datos_test_pred3 <- knn(train=datos_train, test=datos_test, cl=datos_train_labels, k=num_k3)
prop3 <- table(datos_test_pred3)
prop_nc3 <- prop3[1]
prop_c3 <- prop3[2]
```

```
#Evaluación del modelo
pander(CrossTable(x = datos_test_labels, y = datos_test_pred3, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

| datos_test_pred3

datos_test_labels	0	1	Row Total
0	1276	7	1283
0.995	0.005	0.921	
0.976	0.082		
0.916	0.005		
1	32	78	110
0.291	0.709	0.079	
0.024	0.918		
0.023	0.056		
Column Total	1308	85	1393
0.939	0.061		

- **t**:

	0	1
0	1276	7
1	32	78

- **prop.row**:

	0	1
0	0.9945	0.005456
1	0.2909	0.7091

- **prop.col**:

	0	1
0	0.9755	0.08235
1	0.02446	0.9176

- **prop.tbl**:

	0	1
0	0.916	0.005025
1	0.02297	0.05599

La función *CrossTable* nos otorga las frecuencias de positivos, negativos, falsos positivos y falsos negativos; así como los porcentajes para estos. En este primer caso, con un valor de K=3, el algoritmo detecta 1273 negativos, 78 positivos, 32 falsos negativos y 7 falsos positivos.

```
num_k5 <- 5

datos_test_pred5 <- knn(train=datos_train, test=datos_test, cl=datos_train_labels, k=num_k5)
prop5 <- table(datos_test_pred5)

prop_nc5 <- prop5[1]
prop_c5 <- prop5[2]


#Evaluación del modelo
pander(CrossTable(x = datos_test_labels, y = datos_test_pred5, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

| datos_test_pred5

datos_test_labels	0	1	Row Total
0	1281	2	1283
0.998	0.002	0.921	
0.974	0.026		
0.920	0.001		
1	34	76	110
0.309	0.691	0.079	
0.026	0.974		
0.024	0.055		
Column Total	1315	78	1393
0.944	0.056		

- **t**:

	0	1
0	1281	2
1	34	76

- **prop.row**:

	0	1
0	0.9984	0.001559
1	0.3091	0.6909

- **prop.col**:

	0	1
0	0.9741	0.02564
1	0.02586	0.9744

- **prop.tbl**:

	0	1
0	0.9196	0.001436
1	0.02441	0.05456

Para K=5, nuestro algoritmo ha detectado 1281 negativos, 76 positivos, 34 falsos negativos y 2 falsos positivos.

```
num_k7 <- 7

datos_test_pred7 <- knn(train=datos_train, test=datos_test, cl=datos_train_labels, k=num_k7)
prop7 <- table(datos_test_pred7)

prop_nc7 <- prop7[1]
prop_c7 <- prop7[2]


#Evaluación del modelo
pander(CrossTable(x = datos_test_labels, y = datos_test_pred7, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

| datos_test_pred7

datos_test_labels	0	1	Row Total
0	1282	1	1283
0.999	0.001	0.921	
0.972	0.014		
0.920	0.001		
———————	———	———	———
1	37	73	110
0.336	0.664	0.079	
0.028	0.986		
0.027	0.052		

datos_test_labels	0	1	Row Total
Column Total	1319	74	1393
0.947	0.053		

- **t**:

	0	1
0	1282	1
1	37	73

- **prop.row**:

	0	1
0	0.9992	0.0007794
1	0.3364	0.6636

- **prop.col**:

	0	1
0	0.9719	0.01351
1	0.02805	0.9865

- **prop.tbl**:

	0	1
0	0.9203	0.0007179
1	0.02656	0.0524

Para K=7, nuestro algoritmo ha detectado 1282 negativos, 73 positivos, 37 falsos negativos y 0 falsos positivos.

```
num_k11 <- 11

datos_test_pred11 <- knn(train=datos_train, test=datos_test, cl=datos_train_labels, k=num_k11)
prop11 <- table(datos_test_pred11)

prop_nc11 <- prop11[1]
prop_c11 <- prop11[2]


#Evaluación del modelo
pander(CrossTable(x = datos_test_labels, y = datos_test_pred11, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

```
| datos_test_pred11
```

datos_test_labels	0	1	Row Total
0	1283	0	1283
1.000	0.000	0.921	
0.968	0.000		
0.921	0.000		
1	43	67	110
0.391	0.609	0.079	
0.032	1.000		
0.031	0.048		
Column Total	1326	67	1393
0.952	0.048		

- **t**:

	0	1
0	1283	0
1	43	67

- **prop.row**:

	0	1
0	1	0
1	0.3909	0.6091

- **prop.col**:

	0	1
0	0.9676	0
1	0.03243	1

- **prop.tbl**:

	0	1
0	0.921	0
1	0.03087	0.0481

Para K=11, nuestro algoritmo ha detectado 1281 negativos, 76 positivos, 34 falsos negativos y 2 falsos positivos.

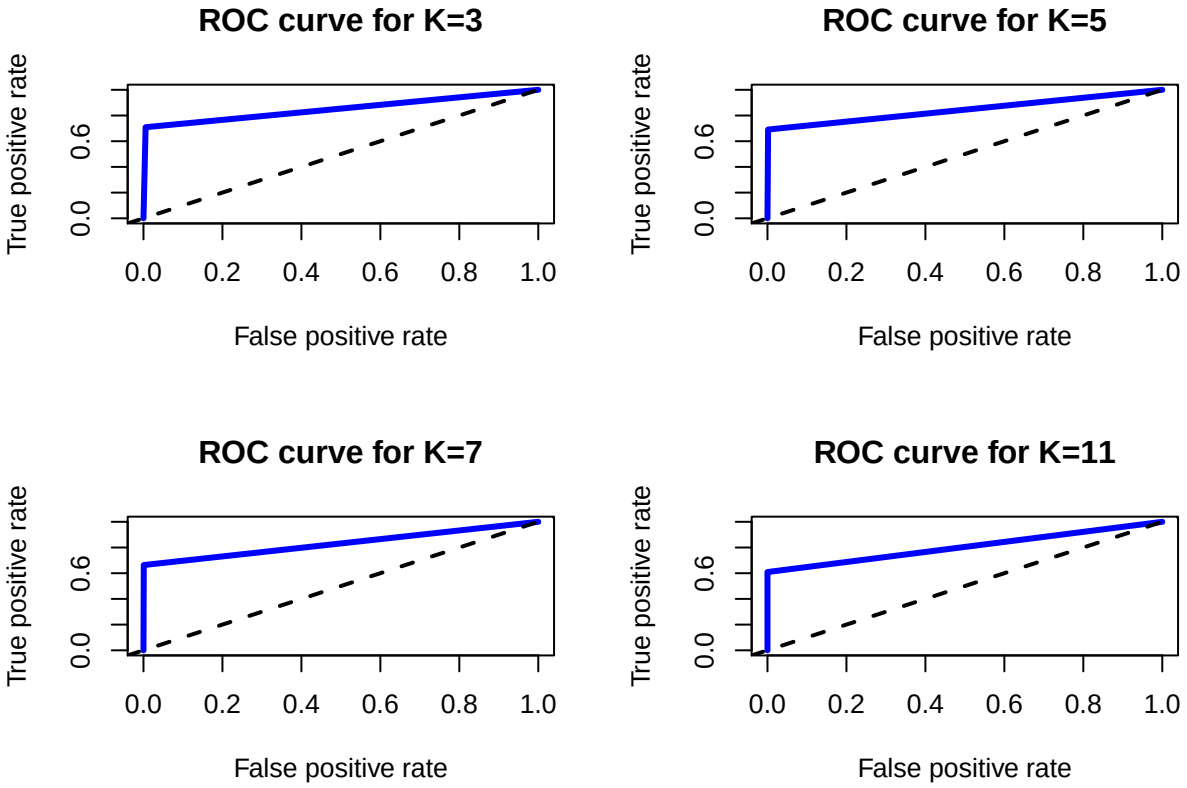
```
kappa3 <- Kappa(table(datos_test_labels, datos_test_pred3))
kappa5 <- Kappa(table(datos_test_labels, datos_test_pred5))
kappa7 <- Kappa(table(datos_test_labels, datos_test_pred7))
kappa11 <- Kappa(table(datos_test_labels, datos_test_pred11))
```

K escogida	Coeficiente Kappa de Cohen
K=3	Value = 0.7852, ASE=0.03323
K=5	Value = 0.7951, ASE=0.03303
K=7	Value = 0.7795, ASE=0.03443
K=11	Value = 0.7416, ASE=0.03747

El que mejor valor de coeficiente Kappa de Cohen presenta es para el modelo con **K=5**. Visualizamos ahora mediante curvas ROC.

Curvas ROC

```
datos_test_pred3 <- as.vector(datos_test_pred3)
datos_test_pred3 <- as.integer(datos_test_pred3)
datos_test_labels <- as.vector(datos_test_labels)
pred3 <- prediction(predictions = datos_test_pred3, labels = datos_test_labels)
perf3 <- performance(pred3, measure = "tpr", x.measure = "fpr")

datos_test_pred5 <- as.vector(datos_test_pred5)
datos_test_pred5 <- as.integer(datos_test_pred5)
pred5 <- prediction(predictions = datos_test_pred5, labels = datos_test_labels)
perf5 <- performance(pred5, measure = "tpr", x.measure = "fpr")

datos_test_pred7 <- as.vector(datos_test_pred7)
datos_test_pred7 <- as.integer(datos_test_pred7)
pred7 <- prediction(predictions = datos_test_pred7, labels = datos_test_labels)
perf7 <- performance(pred7, measure = "tpr", x.measure = "fpr")

datos_test_pred11 <- as.vector(datos_test_pred11)
datos_test_pred11 <- as.integer(datos_test_pred11)
pred11 <- prediction(predictions = datos_test_pred11, labels = datos_test_labels)
perf11 <- performance(pred11, measure = "tpr", x.measure = "fpr")

#Curva ROC
par(mfrow=c(2,2))
plot(perf3, main = "ROC curve for K=3",col = "blue", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
plot(perf5, main = "ROC curve for K=5",col = "blue", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
plot(perf7, main = "ROC curve for K=7",col = "blue", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
plot(perf11, main = "ROC curve for K=11",col = "blue", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
```



Visualmente no presentan grandes diferentes. Calculamos el área bajo la curva (AUC) para comprobar cual de las diferentes K maximiza dicha área.

Step 5: AUC

(e) Comentar los resultados de la clasificación en función del AUC, número de falsos positivos, falsos negativos y error de clasificación obtenidos para los diferentes valores de k. La clase que será asignada como positiva es la 1.

```
perf.auc3 <- performance(pred3, measure = "auc")
perf.auc5 <- performance(pred5, measure = "auc")
perf.auc7 <- performance(pred7, measure = "auc")
perf.auc11 <- performance(pred11, measure = "auc")

auc3 <- unlist(perf.auc3@y.values)
auc5 <- unlist(perf.auc5@y.values)
auc7 <- unlist(perf.auc7@y.values)
auc11 <- unlist(perf.auc11@y.values)
```

AUC según la k escogida	Valor del Área bajo la curva (AUC)
AUC para K=3	0.852
AUC paraK=5	0.845
AUC paraK=7	0.831
AUC paraK=11	0.804

Curiosamente, el valor de k que maximiza AUC es 3, con un valor de **0.852**. Esto implica que, para un nuevo octámero dado, nuestro algoritmo tiene un 85.2% de posibilidades de determinar correctamente si dicho octámero será reconocido y escindido por la proteasa de VIH-1.

Datos generados

A continuación vamos a generar un dataframe para las secuencias octaméricas predefinidas. La función *generador* diseñada previamente asignará a cada octámero una secuencia de 160 bits, específica para cada combinación de aminoácidos.

Step 2: Codificación ortogonal

```
secuencias_gen <- rbind(res1, res2)
octameros <- secuencias_gen[1]


valores <- generador(octameros)
valores <- as.data.frame(valores)


datos_g <- cbind(secuencias_gen$V2, valores)
```

Step 3: Separación de datos

```
set.seed(123)

#Para ordenarlo de forma aleatoria creamos una distribución aleatoria de todas las columnas, y posterio
rows_g <- sample(nrow(datos_g))
datos_n_g <- datos_g[rows, ]
#La expresión round(0.67*nrow(ort)), nos proporciona el 67% de los valores por orden.


datos_train_g <- datos_n_g[1:round(0.67*nrow(datos_n_g)), ]
datos_test_g <- datos_n_g[round(0.67*nrow(datos_n_g)):length(datos_n_g$V1),]

datos_train_labels_g <- datos_n_g[1:round(0.67*nrow(datos_n_g)), 2]
datos_test_labels_g <- datos_n_g[round(0.67*nrow(datos_n_g)):length(datos_n_g$V1), 2]
```

Step 4: Algoritmo kNN y curvas ROC

(d) Utilizar un knn (k = 3, 5, 7, 11) basado en el training para predecir que octameros del test tienen o no cleavage site. Además, realizar una curva ROC para cada k y calcular su área bajo la curva (AUC).

```
#num_k3 <- 3

datos_test_pred3_g <- knn(train=datos_train_g, test=datos_test_g, cl=datos_train_labels_g, k=num_k3)
```

```
prop3_g <- table(datos_test_pred3_g)
prop_nc3_g <- prop3_g[1]
prop_c3_g <- prop3_g[2]


#Evaluación del modelo
pander(CrossTable(x = datos_test_labels_g, y = datos_test_pred3_g, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

| datos_test_pred3_g

datos_test_labels_g	0	1	Row Total
0	1256	5	1261
0.996	0.004	0.905	
0.960	0.059		
0.902	0.004		
1	52	80	132
0.394	0.606	0.095	
0.040	0.941		
0.037	0.057		
Column Total	1308	85	1393
0.939	0.061		

- **t**:

	0	1
0	1256	5
1	52	80

- **prop.row**:

	0	1
0	0.996	0.003965
1	0.3939	0.6061

- **prop.col**:

	0	1
0	0.9602	0.05882
1	0.03976	0.9412

- **prop.tbl**:

	0	1
0	0.9017	0.003589
1	0.03733	0.05743

Para K=3, nuestro algoritmo ha detectado 1265 negativos, 80 positivos, 52 falsos negativos y 5 falsos positivos.

```
#num_k5 <- 5

datos_test_pred5_g <- knn(train=datos_train_g, test=datos_test_g, cl=datos_train_labels_g, k=num_k5)
prop5_g <- table(datos_test_pred5_g)

prop_nc5_g <- prop5_g[1]
prop_c5_g <- prop5_g[2]


#Evaluación del modelo
pander(CrossTable(x = datos_test_labels_g, y = datos_test_pred5_g, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

| datos_test_pred5_g

datos_test_labels_g	0	1	Row Total
0	1260	1	1261
0.999	0.001	0.905	
0.952	0.014		
0.905	0.001		
1	63	69	132
0.477	0.523	0.095	
0.048	0.986		
0.045	0.050		
Column Total	1323	70	1393
0.950	0.050		

- **t**:

	0	1
0	1260	1
1	63	69

- **prop.row**:

	0	1
0	0.9992	0.000793
1	0.4773	0.5227

- **prop.col**:

	0	1
0	0.9524	0.01429
1	0.04762	0.9857

- **prop.tbl**:

	0	1
0	0.9045	0.0007179
1	0.04523	0.04953

Para K=5, nuestro algoritmo ha detectado 1260 negativos, 69 positivos, 63 falsos negativos y 1 falsos positivos.

```
#num_k7 <- 7

datos_test_pred7_g <- knn(train=datos_train_g, test=datos_test_g, cl=datos_train_labels_g, k=num_k7)
prop7_g <- table(datos_test_pred7_g)

prop_nc7_g <- prop7_g[1]
prop_c7_g <- prop7_g[2]


#Evaluación del modelo
pander(CrossTable(x = datos_test_labels_g, y = datos_test_pred7_g, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

	datos_test_pred7_g		
datos_test_labels_g	0	1	Row Total
0	1261	0	1261
1.000	0.000	0.905	
0.950	0.000		
0.905	0.000		
———————	———	———	———
1	67	65	132
0.508	0.492	0.095	

datos_test_labels_g	0	1	Row Total
0.050	1.000		
0.048	0.047		
Column Total	1328	65	1393
0.953	0.047		

- **t**:

	0	1
0	1261	0
1	67	65

- **prop.row**:

	0	1
0	1	0
1	0.5076	0.4924

- **prop.col**:

	0	1
0	0.9495	0
1	0.05045	1

- **prop.tbl**:

	0	1
0	0.9052	0
1	0.0481	0.04666

Para K=7, nuestro algoritmo ha detectado 1261 negativos, 65 positivos, 67 falsos negativos y 0 falsos positivos.

```
#num_k11 <- 11

datos_test_pred11_g <- knn(train=datos_train_g, test=datos_test_g, cl=datos_train_labels_g, k=num_k11)
prop11_g <- table(datos_test_pred11_g)

prop_nc11_g <- prop11_g[1]
prop_c11_g <- prop11_g[2]


#Evaluación del modelo
pander(CrossTable(x = datos_test_labels_g, y = datos_test_pred11_g, prop.chisq=FALSE))
```

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

Total Observations in Table: 1393

| datos_test_pred11_g

datos_test_labels_g	0	1	Row Total
0	1261	0	1261
1.000	0.000	0.905	
0.943	0.000		
0.905	0.000		
1	76	56	132
0.576	0.424	0.095	
0.057	1.000		
0.055	0.040		
Column Total	1337	56	1393
0.960	0.040		

- **t**:

	0	1
0	1261	0
1	76	56

- **prop.row**:

	0	1
0	1	0
1	0.5758	0.4242

- **prop.col**:

	0	1
0	0.9432	0
1	0.05684	1

- **prop.tbl**:

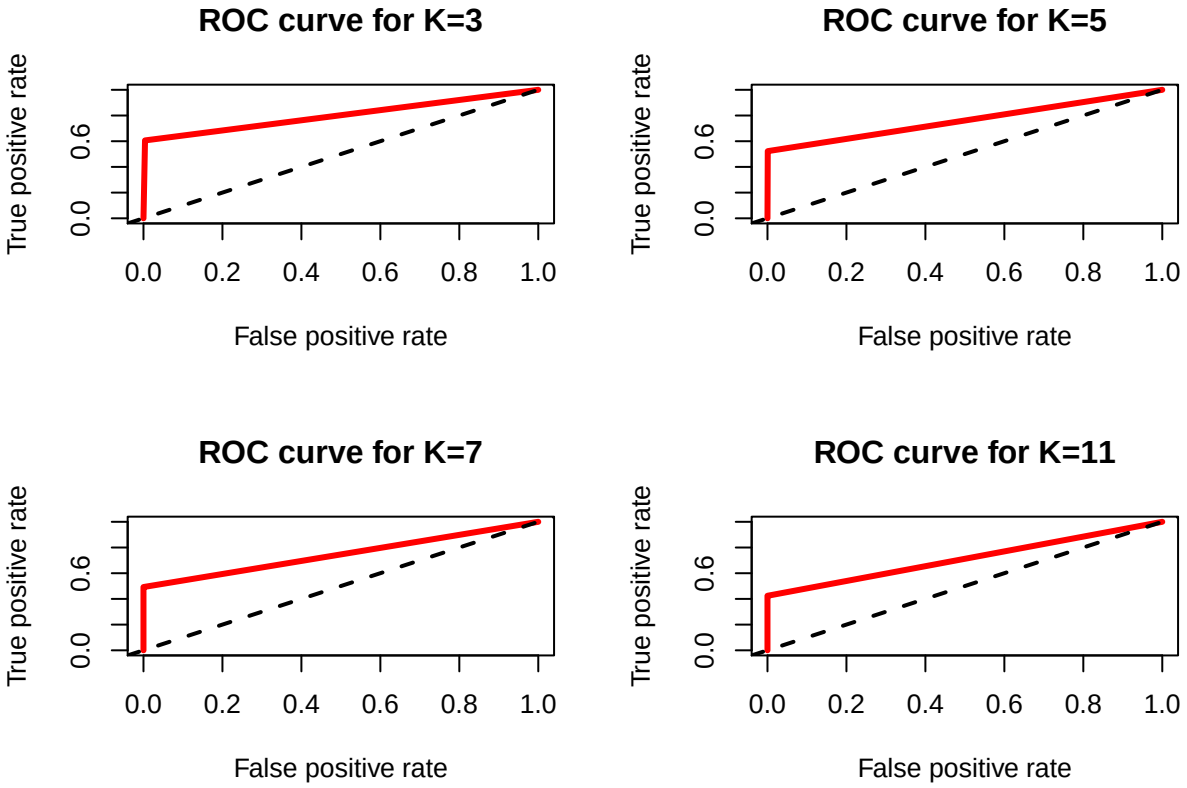
	0	1
0	0.9052	0
1	0.05456	0.0402

Para K=11, nuestro algoritmo ha detectado 1261 negativos, 56 positivos, 76 falsos negativos y 0 falsos

positivos.

```
kappa3_g <- Kappa(table(datos_test_labels_g, datos_test_pred3_g))
kappa5_g <- Kappa(table(datos_test_labels_g, datos_test_pred5_g))
kappa7_g <- Kappa(table(datos_test_labels_g, datos_test_pred7_g))
kappa11_g <- Kappa(table(datos_test_labels_g, datos_test_pred11_g))
```

```
datos_test_pred3_g <- as.vector(datos_test_pred3_g)
datos_test_pred3_g <- as.integer(datos_test_pred3_g)
datos_test_labels_g <- as.vector(datos_test_labels_g)
pred3_g <- prediction(predictions = datos_test_pred3_g, labels = datos_test_labels_g)
perf3_g <- performance(pred3_g, measure = "tpr", x.measure = "fpr")

datos_test_pred5_g <- as.vector(datos_test_pred5_g)
datos_test_pred5_g <- as.integer(datos_test_pred5_g)
pred5_g <- prediction(predictions = datos_test_pred5_g, labels = datos_test_labels_g)
perf5_g <- performance(pred5_g, measure = "tpr", x.measure = "fpr")

datos_test_pred7_g <- as.vector(datos_test_pred7_g)
datos_test_pred7_g <- as.integer(datos_test_pred7_g)
pred7_g <- prediction(predictions = datos_test_pred7_g, labels = datos_test_labels_g)
perf7_g <- performance(pred7_g, measure = "tpr", x.measure = "fpr")

datos_test_pred11_g <- as.vector(datos_test_pred11_g)
datos_test_pred11_g <- as.integer(datos_test_pred11_g)
pred11_g <- prediction(predictions = datos_test_pred11_g, labels = datos_test_labels_g)
perf11_g <- performance(pred11_g, measure = "tpr", x.measure = "fpr")

#Curva ROC
par(mfrow=c(2,2))
plot(perf3_g, main = "ROC curve for K=3",col = "red", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
plot(perf5_g, main = "ROC curve for K=5",col = "red", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
plot(perf7_g, main = "ROC curve for K=7",col = "red", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
plot(perf11_g, main = "ROC curve for K=11",col = "red", lwd = 3)
abline(a = 0, b = 1, lwd = 2, lty = 2)
```



Visualmente no presentan grandes diferentes. Calculamos el área bajo la curva (AUC) para comprobar cual de las diferentes K maximiza dicha área.

Step 5: AUC

```
perf.auc3_g <- performance(pred3_g, measure = "auc")
perf.auc5_g <- performance(pred5_g, measure = "auc")
perf.auc7_g <- performance(pred7_g, measure = "auc")
perf.auc11_g <- performance(pred11_g, measure = "auc")

auc3_g <- unlist(perf.auc3_g@y.values)
auc5_g <- unlist(perf.auc5_g@y.values)
auc7_g <- unlist(perf.auc7_g@y.values)
auc11_g <- unlist(perf.auc11_g@y.values)
```

AUC según la k escogida	Valor del Área bajo la curva (AUC)
AUC para K=3	0.801
AUC para K=5	0.761
AUC para K=7	0.746
AUC para K=11	0.712

Nuestro mayor valor de AUC se corresponde con el valor de K=3. Este tiene un valor de 0.801, lo que sugiere

que hay un 80.1% de posibilidades de que para un octámero dado nuestro algoritmo determine fidedignamente si va a ser reconocido y escindido por la proteasa.

Resultados

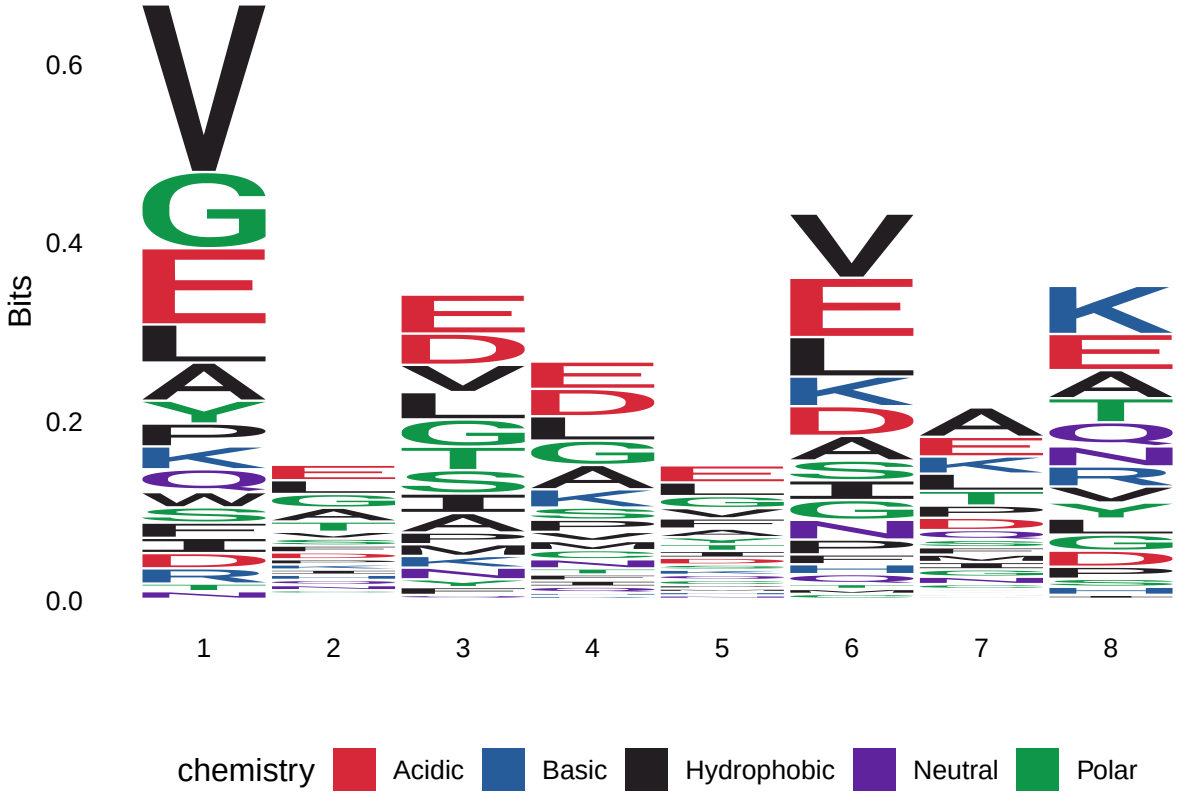
Las técnicas de diagnóstico para determinar la fiabilidad de nuestro algoritmo han resultado ligeramente diferentes para ambos modelos. El primero, obtenido a partir del dataframe disponible, tenía un 85% de posibilidades de predecir correctamente la escisión de un nuevo octámero; mientras que para los datos obtenidos a partir de nuestra función generadora de codificación, dicho porcentaje de predicción se reducía hasta un 80%. En ambos casos, K = 3es la mejor opción.

Observabamos además, como según ibamos aumentando el vlaor de K, los valores de ROC descendían, detectabamos menos falsos positivos y más negativos, pero también menos positivos y más falsos negativos. Esto es debido a que los posibles resultados para nuestra clase (positivo=escindidido por la proteasa, y negativo=no escindido por la proteasa), se encuentran muy desproporcionados, siendo los octámeros con un valor negativo, mucho más abundantes. Esto produce que el equilibrio entre los dos tipos de errores se encuentre a niveles bajos de K; conforme la vayamos aumentando nuestro algoritmo pierde precisión, se vuelve demasiado laxo en la detección de negativos y pierde fiabilidad.

Adjuntamos finalmente una pequeña implementación del paquete *ggseplogo*. Para cada uno de los modelos empleados, vamos a obtener la representación del octámero, siempre que acierte en su predicción, para el valor de K que maximiza el área bajo la curva. Para ello, crearemos un vector con las secuencias aminoacídicas reconocidas correctamente como dianas de escisión.

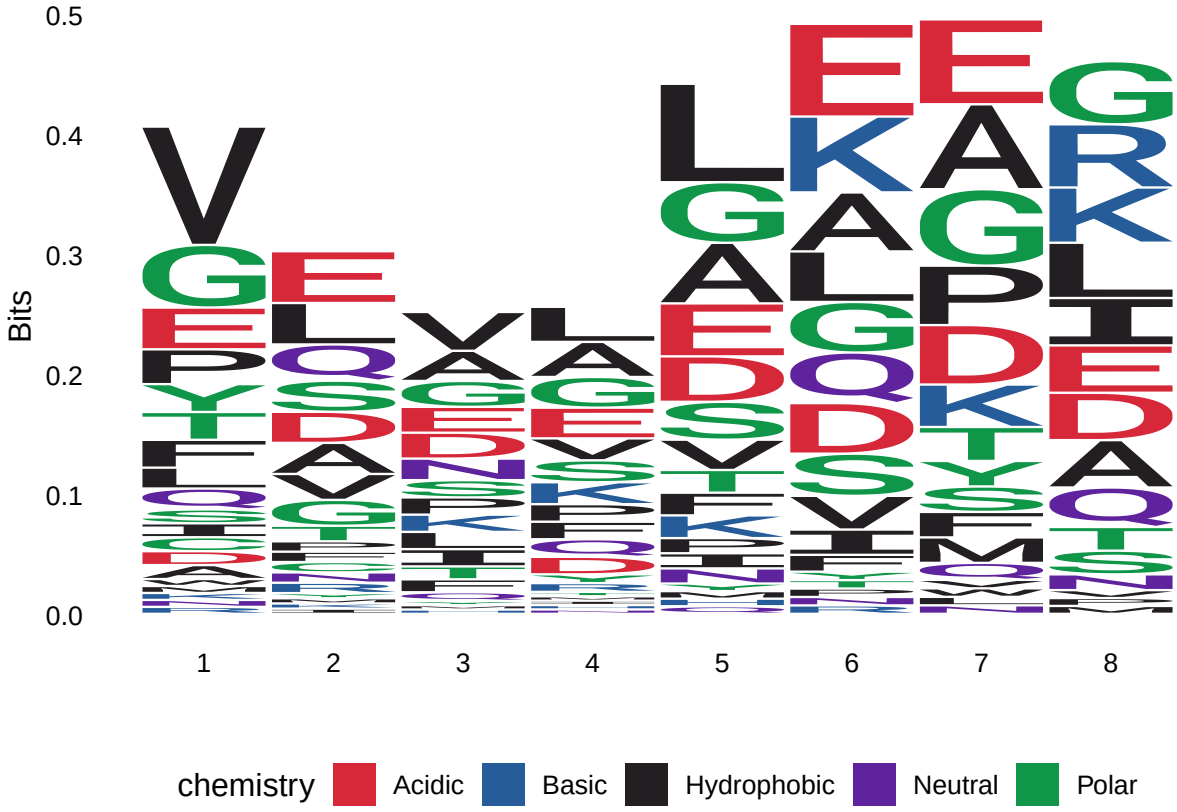
Para valores originales

```
#Hacemos los datos compatibles.
x1 <- as.data.frame(datos_test_pred5)
y1 <- secuencias[round(0.67*nrow(datos_n)):length(datos_n$V1),]
y1 <- y1[1]

#Creamos el vector
x1y1 <- data.frame(y1, x1)
x1y1 <- subset(x1y1, x1y1[,2] == 1)
x1y1 <- as.vector(x1y1$Octamero)

#Aplicamos la funcion
ggseqlogo(x1y1, method="bits")
```



Para los valores generados

```
#Hacemos los datos compatibles.
x2 <- as.data.frame(datos_test_pred5_g)
y2 <- secuencias[round(0.67*nrow(datos_n_g)):length(datos_n_g$V1),]
y2 <- y2[1]

#Creamos el vector
x2y2 <- data.frame(y2, x2)
x2y2 <- subset(x2y2, x2y2[,2] == 1)
x2y2 <- as.vector(x2y2$Octamero)

#Aplicamos la funcion
ggseqlogo(x2y2)
```



Bibliografía

Kramer, Oliver. 2013. *Dimensionality Reduction with Unsupervised Nearest Neighbors*. Springer.

Lantz, Brett. 2013. *Machine Learning with r*. Packt publishing ltd.

Mucherino, Antonio, Petraq Papajorgji, and Panos M Pardalos. 2009. *Data Mining in Agriculture*. Vol. 34. Springer Science & Business Media.

Wlodawer, Alexander, and Jiri Vondrasek. 1998. "Inhibitors of HIV-1 Protease: A Major Success of Structure-Assisted Drug Design." *Annual Review of Biophysics and Biomolecular Structure* 27 (1): 249–84.