PEC1: Predicción de los splice junctions

Machine Learning

Escribir vuestro nombre y apellidos

13 de noviembre, 2021

Índice

```r
# CRAN repository

libraries <- c("class", "gmodels",  "knitr","pROC", "ROCR","devtools", "stringr","ggplot2")
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]
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)
}
```

Descripción

Los *splice junctions* son puntos en una secuencia de ADN en los que se elimina el ADN "superfluo" durante el proceso de síntesis de proteínas en organismos superiores. El problema que se plantea en este conjunto de datos es reconocer, dada una secuencia de ADN, los límites entre los exones (las partes de la secuencia de ADN retenidas después del *splicing*) y los intrones (las partes de la secuencia de ADN que se cortan). Este problema consta de dos subtareas: reconocer los límites exón/intrón (denominados sitios EI) y reconocer los límites intrón/exón (sitios IE). En la comunidad biológica, las fronteras de la IE se denominan `acceptors`, mientras que las fronteras de la EI se denominan `donors`. Todos los ejemplos fueron tomados de Genbank 64.1. Las categorías "EI" e "IE" incluyen "genes con splicing" de los primates en Genbank 64.1. Los ejemplos de no *splicing* fueron tomados de secuencias que se sabe que no incluyen un sitio de *splicing*.

Los datos estan disponibles en la PEC en el fichero `splice.txt`. El archivo contiene 3190 filas que corresponden a las distintas secuencias, y 3 columnas separadas por coma. La primera columna correspondiente a la clase de la secuencia (EI, IE o N), la segunda columna con el nombre identificador de la secuencia y la tercera columna con la secuencia propiamente. Tratandose de secuencias de ADN, apareceran los nucleótidos identificados de manera estandar con las letras A, G, T y C. Además, aparecen otros caracteres entre los caracteres estándar, D, N, S y R, que indican ambigüedad según la siguiente tabla:

```
|caracter | significado   |
|---------|---------------|
|D        | A o G o T     |
|N        | A o G o C o T |
|S        | C o G         |
|R        | A o G         |
```

Para más información acerca del caso se puede consultar el link: https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Splice-junction+Gene+Sequences), del repositorio de machine learning de la Universidad de California-Irvine (UCI).

La manera elegida para representar los datos es un paso crucial en los algoritmos. En el caso que nos ocupa, análisis basados en secuencias, se usará la codificación `one-hot`.

La codificación `one-hot` representa cada nucleótido por un vector de 8 componentes, con 7 de ellas a 0 y una a 1. Pongamos por ejemplo, el nucleótido A se representa por (1,0,0,0,0,0,0,0), el nucleótido G por (0,1,0,0,0,0,0,0), el T por (0,0,1,0,0,0,0,0) y, finalmente, la C por (0,0,0,1,0,0,0,0) y los caracteres de ambiguedad los representaremos, la D por (0,0,0,0,1,0,0,0), la N por (0,0,0,0,0,1,0,0), la S por (0,0,0,0,0,0,1,0) y la R por (0,0,0,0,0,0,0,1).

En esta PEC se implementará un algoritmo `knn` para reconocer los *splice junctions*.

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.

Función para implementar la codificación `one-hot`

Este seria una de las maneras de programar la codificación `one-hot`. Seguro que tu lo has pensado de otra manera.

```
# A,G,T,C,D,N,S,R
one_hot <- function(mydades, seq.pos){

  A<-c(1,0,0,0,0,0,0,0)
  G<-c(0,1,0,0,0,0,0,0)
  T<-c(0,0,1,0,0,0,0,0)
  C<-c(0,0,0,1,0,0,0,0)
  D<-c(0,0,0,0,1,0,0,0)
  N<-c(0,0,0,0,0,1,0,0)
  S<-c(0,0,0,0,0,0,1,0)
  R<-c(0,0,0,0,0,0,0,1)


  df<-data.frame()

  for( s in 1:nrow(mydades)){

    seq<-unlist(strsplit(str_trim(mydades[s,seq.pos]),split=NULL))

    v<-vector()

    for (i in 1:length(seq)){

      xchar<-seq[i]

      if (xchar=="A") v<-c(v,A)
      if (xchar=="G") v<-c(v,G)
      if (xchar=="T") v<-c(v,T)
      if (xchar=="C") v<-c(v,C)
      if (xchar=="D") v<-c(v,D)
      if (xchar=="N") v<-c(v,N)
      if (xchar=="S") v<-c(v,S)
      if (xchar=="R") v<-c(v,R)

  }
    df[s,1:length(v)]<-v    # 480=60*8
  }
  return(df)
```

```
}
```

La función tiene dos argumentos, `mydades` y `seq.pos`, el primero corresponde al data frame y el segundo a la posición de la columna donde está la secuencia en el data frame.

Para probar su uso, se presenta el siguiente ejemplo:

```
df.0 <-data.frame(name= c("seq1", "seq2"), seq=c("ACGS", "RSND"))
one_hot(df.0,2)
```

```
  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24
1  1  0  0  0  0  0  0  0  0   0   0   1   0   0   0   0   0   1   0   0   0   0   0   0
2  0  0  0  0  0  0  0  1  0   0   0   0   0   0   1   0   0   0   0   0   0   1   0   0
  V25 V26 V27 V28 V29 V30 V31 V32
1   0   0   0   0   0   0   1   0
2   0   0   0   0   1   0   0   0
```

Desarrollar el script en R que implemente un clasificador knn

Step 1 - Lectura y breve descripción de los datos

Se leen los datos del fichero `splice.txt` o se pueden bajar directamente del link: https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Splice-junction+Gene+Sequences), del repositorio de machine learning de la Universidad de California-Irvine (UCI).

```
mydata <- read.csv(file.path(params$folder.data,params$file), header=FALSE)
```

El conjunto de datos denominado *splice.txt* esta formado por 3190 registros. La primera columna correspondiente a la clase de la secuencia (EI, IE o N), la segunda columna con el nombre identificador de la secuencia y la tercera columna con la secuencia propiamente.

Se muestra los primeros 6 registros para ver si se ha leido bien el fichero

```
head(mydata)
```

```
  V1                     V2
1 EI       ATRINS-DONOR-521
2 EI       ATRINS-DONOR-905
3 EI       BABAPOE-DONOR-30
4 EI      BABAPOE-DONOR-867
5 EI     BABAPOE-DONOR-2817
6 EI     CHPIGECA-DONOR-378
                                                                              V3
1                 CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
2                 AGACCCGCCGGGGAGGCGGAGGACCTGCAGGGTGAGCCCCACCGCCCCTCCGTGCCCCCGC
3                 GAGGTGAAGGACGTCCTTCCCCAGGAGCCGGTGAGAAGCGCAGTCGGGGGCACGGGGATG
4                 GGGCTGCGTTGCTGGTCACATTCCTGGCAGGTATGGGGCGGGGCTTGCTCGGTTTTCCCC
5                 GCTCAGCCCCCAGGTCACCCAGGAACTGACGTGAGTGTCCCCATCCCGGCCCTTGACCCT
6                 CAGACTGGGTGGACAACAAAACCTTCAGCGGTAAGAGAGGGCCAAGCTCAGAGACCACAG
```

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

```
str(mydata)
```

```
'data.frame':   3190 obs. of  3 variables:
 $ V1: chr  "EI" "EI" "EI" "EI" ...
 $ V2: chr  "    ATRINS-DONOR-521" "    ATRINS-DONOR-905" "    BABAPOE-DONOR-30" "    BABAPOE-DONOR-867
```

```
 $ V3: chr  "               CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG" "
```

```
colnames(mydata)<-c("class","name","sequence")
mydata$class<-as.factor(mydata$class)
mydata$name<-as.character(mydata$name)
mydata$sequence<-as.character(mydata$sequence)
str(mydata)
```

```
'data.frame':   3190 obs. of  3 variables:
 $ class   : Factor w/ 3 levels "EI","IE","N": 1 1 1 1 1 1 1 1 1 1 ...
 $ name    : chr  "    ATRINS-DONOR-521" "    ATRINS-DONOR-905" "    BABAPOE-DONOR-30" "    BABAPOE-DON
 $ sequence: chr  "               CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG" "
```

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

```
table(mydata[,1])
```

```
  EI   IE    N
 767  768 1655
```

Step 2 - Transformación - Cambiar la secuencia de nucleótidos en una secuencia numérica mediante one-hot.

Es ejecutar la función `one_hot` para los datos leidos.

```
out <- one_hot(mydata, 3)
```

Ahora se ha creado un nuevo fichero out con el mismo número de filas que secuencias, 3190 , y con tantas columnas como `8 * 60` ya que para cada base se tiene un alfabeto de 8 elementos (4 nucleótidos y 4 que indican ambiguedad en los nucleótidos) y 60 que es la longitud de la secuencia.

```
[1] 3190  480
```

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

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

```
  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
1  0  0  0  1  0  0  0  0  0   0   0   1   0   0   0   0   1   0   0   0
2  1  0  0  0  0  0  0  0  0   1   0   0   0   0   0   0   1   0   0   0
3  0  1  0  0  0  0  0  0  1   0   0   0   0   0   0   0   0   1   0   0
4  0  1  0  0  0  0  0  0  0   1   0   0   0   0   0   0   0   1   0   0
5  0  1  0  0  0  0  0  0  0   0   0   1   0   0   0   0   0   0   1   0
6  0  0  0  1  0  0  0  0  1   0   0   0   0   0   0   0   0   1   0   0
```

Subset secuencias de las clases `EI` y `N`

Seleccionar los registros de las clases `EI` y `N`.

```
# subsets
subset.sj.N <- subset(mydata,
                      mydata$class!="IE")
subset.sj.N.oh <- subset(out,
                      mydata$class!="IE")
# drop levels
subset.sj.N$class <- droplevels(subset.sj.N$class)
# check
#dim(subset.sj.N.oh)
#dim(subset.sj.N)
```

El número de secuencias de cada clase es:

```
table(subset.sj.N$class)
```

```
  EI    N
 767 1655
```

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(subset.sj.N),round(2*nrow(subset.sj.N)/3,0))
# create training and test data
out_training<-subset.sj.N.oh[train,]
out_test<-subset.sj.N.oh[-train,]
# create labels for training and test data
class_training<-subset.sj.N[train,1]
class_test<-subset.sj.N[-train,1]
```

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=1)
```

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:  807


             | test_pred
  class_test |        EI |         N | Row Total |
-------------|-----------|-----------|-----------|
```

```
          EI |       231 |        16 |       247 |
             |     0.935 |     0.065 |     0.306 |
             |     0.638 |     0.036 |           |
             |     0.286 |     0.020 |           |
-------------|-----------|-----------|-----------|
           N |       131 |       429 |       560 |
             |     0.234 |     0.766 |     0.694 |
             |     0.362 |     0.964 |           |
             |     0.162 |     0.532 |           |
-------------|-----------|-----------|-----------|
Column Total |       362 |       445 |       807 |
             |     0.449 |     0.551 |           |
-------------|-----------|-----------|-----------|
```

```
table(class_test)
```

```
class_test
 EI   N
247 560
```

Como hay que hacerlo para varios valores de k, es preferible realizar un proceso iteractivo como se muestra en las secciones siguientes.

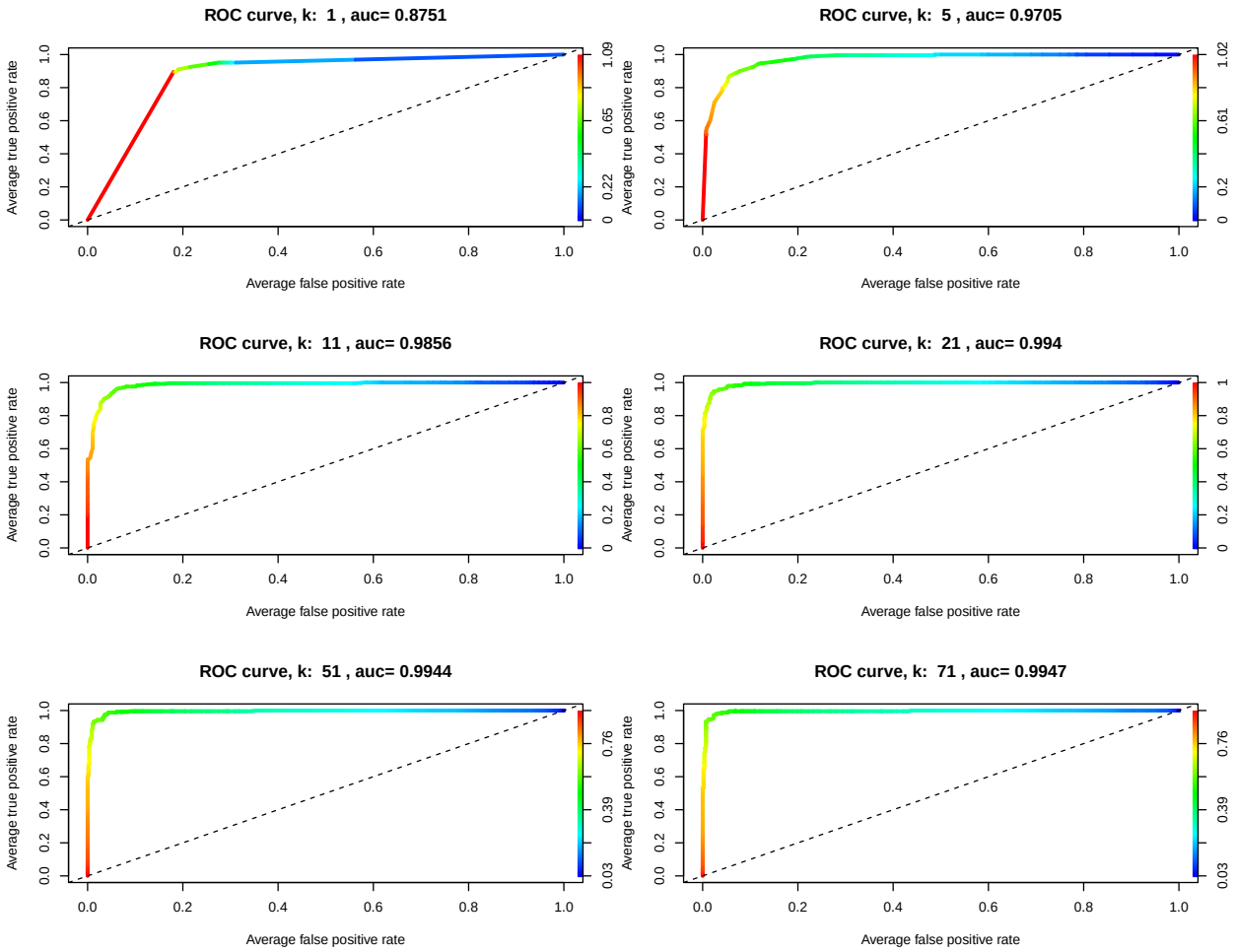
Curva ROC del clasificador kNN para diversos valores de k

Se realiza un proceso iteractivo para cada valor de `k` donde se entrena el algoritmo kNN, evalua y se obtiene su curva ROC.

Para realizar la curva ROC de cada valor de `k` se necesita obtener las probabilidades de la clase positiva, en este caso es la clase `EI`, 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(1,5,11,21,51,71)
vec.auc <- NULL
par(mfrow=c(3,2))
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 == "EI")
  prob1 <- ifelse(test_pred == "EI", 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("N","EI"))
  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))
```

Evaluación del clasificador kNN para varios valores de k

Como hay que evaluar el algoritmo kNN para diferentes valores de `k`, en particular seran 1, 5, 11, 21, 51, 71 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$file, sep=""))
```

Cuadro 2: Algoritmo kNN: splice.txt

valor k	# falsos negativos	# falsos positivos	% mal clasificados	AUC
1	138	18	19.330855	0.8751012
5	86	10	11.895911	0.9704634
11	68	2	8.674102	0.9855733
21	53	2	6.815366	0.9940103
51	38	2	4.956630	0.9944079
71	38	1	4.832714	0.9947260

Entre los 6 valores de `k` evaluados el que tiene un menor porcentaje de clasificación erronea es el valor **71** con un valor de **4.83271** % mal clasificados. El rango de % mal clasificados está entre *4.83271* y *19.33086*. Así que la diferencia entre el valor mínimo y máximo es *14.49814* y por tanto, es grande la diferencia.

Observar que el promedio de falsos negativos es *70.17* y su desviacion estandard *38*. Para el caso de falsos positivos es *5.83* y su desviacion estandard *6.82*. En promedio hay más *falsos negativos* con una relación de *35* veces respecto al menor tipo de error.

Respecto al AUC, el valor de `k` con mayor AUC es **71** con un AUC de **0.9947**. El rango de valores de AUC está entre *0.8751* y *0.9947*. Así que, la diferencia entre el valor mínimo y máximo es *0.1196* y por tanto, es pequeña la diferencia.

Subset secuencias de las clases `IE` y `N`

Seleccionar los registros de las clases `IE` y `N`.

```
# subsets
subset.sj.N <- subset(mydata,
                      mydata$class!="EI")
subset.sj.N.oh <- subset(out,
                      mydata$class!="EI")
# drop levels
subset.sj.N$class <- droplevels(subset.sj.N$class)
# check
#dim(subset.sj.N.oh)
#dim(subset.sj.N)
```

El número de secuencias de cada clase es:

```
table(subset.sj.N$class)
```

```
  IE    N
 768 1655
```

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(subset.sj.N),round(2*nrow(subset.sj.N)/3,0))
# create training and test data
out_training<-subset.sj.N.oh[train,]
out_test<-subset.sj.N.oh[-train,]
# create labels for training and test data
```

```
class_training<-subset.sj.N[train,1]
class_test<-subset.sj.N[-train,1]
```

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=1)
```

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:  808


             | test_pred
  class_test |        IE |         N | Row Total |
-------------|-----------|-----------|-----------|
          IE |       226 |        22 |       248 |
             |     0.911 |     0.089 |     0.307 |
             |     0.691 |     0.046 |           |
             |     0.280 |     0.027 |           |
-------------|-----------|-----------|-----------|
           N |       101 |       459 |       560 |
             |     0.180 |     0.820 |     0.693 |
             |     0.309 |     0.954 |           |
             |     0.125 |     0.568 |           |
-------------|-----------|-----------|-----------|
Column Total |       327 |       481 |       808 |
             |     0.405 |     0.595 |           |
-------------|-----------|-----------|-----------|
```

```
table(class_test)
```

```
class_test
 IE   N
248 560
```

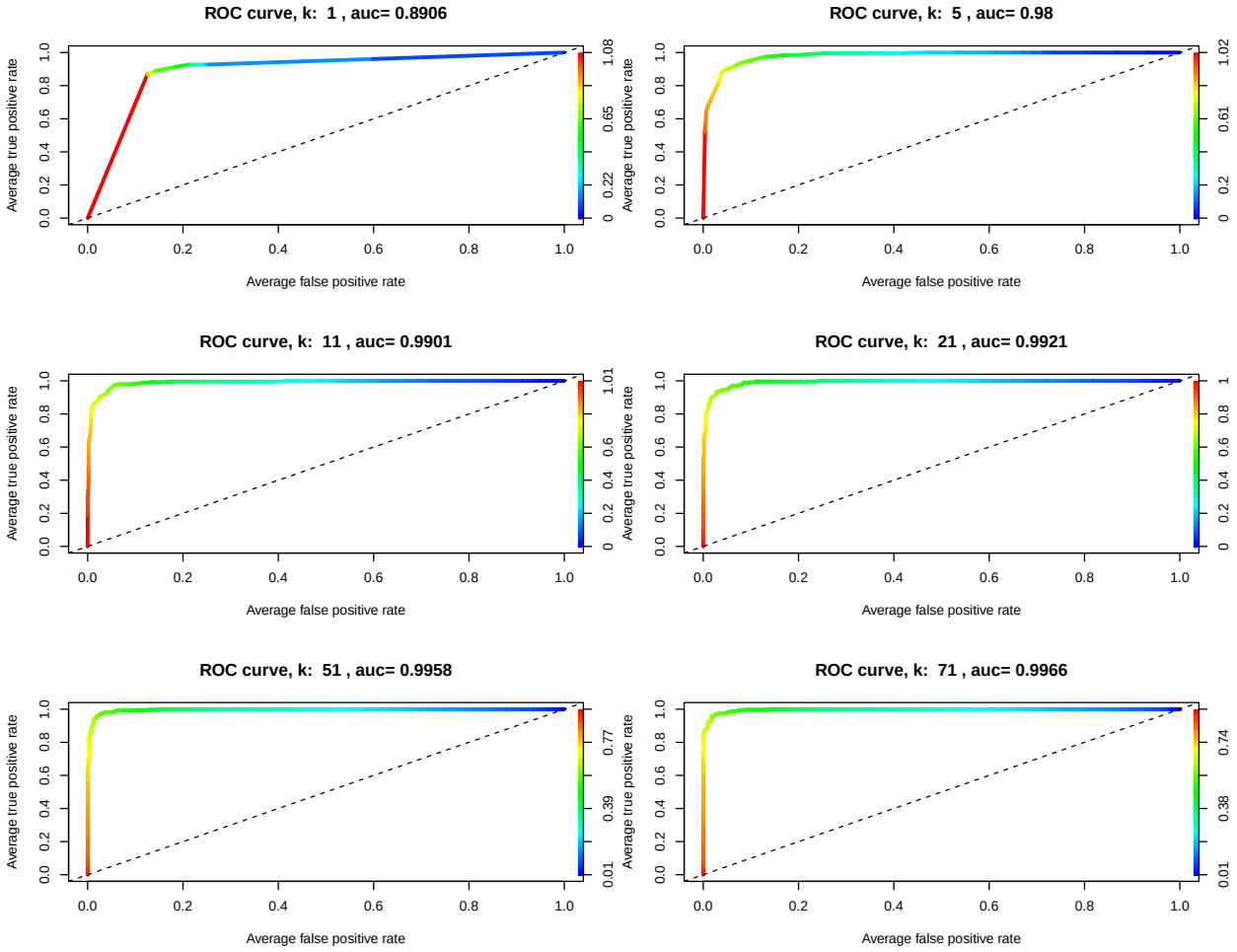
Como hay que hacerlo para varios valores de k, es preferible realizar un proceso iteractivo como se muestra en las secciones siguientes.

Curva ROC del clasificador kNN para diversos valores de k

Se realiza un proceso iteractivo para cada valor de `k` donde se entrena el algoritmo kNN, evalua y se obtiene su curva ROC.

Para realizar la curva ROC de cada valor de `k` se necesita obtener las probabilidades de la clase positiva, en este caso es la clase `IE`, 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(1,5,11,21,51,71)
vec.auc <- NULL
par(mfrow=c(3,2))
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 == "IE")
  prob1 <- ifelse(test_pred == "IE", 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("N","IE"))
  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))
```

Evaluación del clasificador kNN para varios valores de k

Como hay que evaluar el algoritmo kNN para diferentes valores de `k`, en particular seran 1, 5, 11, 21, 51, 71 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$file, sep=""))
```

Cuadro 3: Algoritmo kNN: splice.txt

valor k	# falsos negativos	# falsos positivos	% mal clasificados	AUC
1	96	22	14.603960	0.8905638
5	86	5	11.262376	0.9800223
11	71	2	9.034653	0.9901174
21	62	1	7.797030	0.9921263

valor k	# falsos negativos	# falsos positivos	% mal clasificados	AUC
51	54	2	6.930693	0.9957949
71	45	2	5.816832	0.9966410

Entre los 6 valores de `k` evaluados el que tiene un menor porcentaje de clasificación erronea es el valor **71** con un valor de **5.81683** % mal clasificados. El rango de % mal clasificados está entre *5.81683* y *14.60396*. Así que la diferencia entre el valor mínimo y máximo es *8.78713* y por tanto, es grande la diferencia.

Observar que el promedio de falsos negativos es *69* y su desviacion estandard *19.35*. Para el caso de falsos positivos es *5.67* y su desviacion estandard *8.12*. En promedio hay más *falsos negativos* con una relación de *34* veces respecto al menor tipo de error.

Respecto al AUC, el valor de `k` con mayor AUC es **71** con un AUC de **0.9966**. El rango de valores de AUC está entre *0.8906* y *0.9966*. Así que, la diferencia entre el valor mínimo y máximo es *0.1061* y por tanto, es pequeña la diferencia.

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)

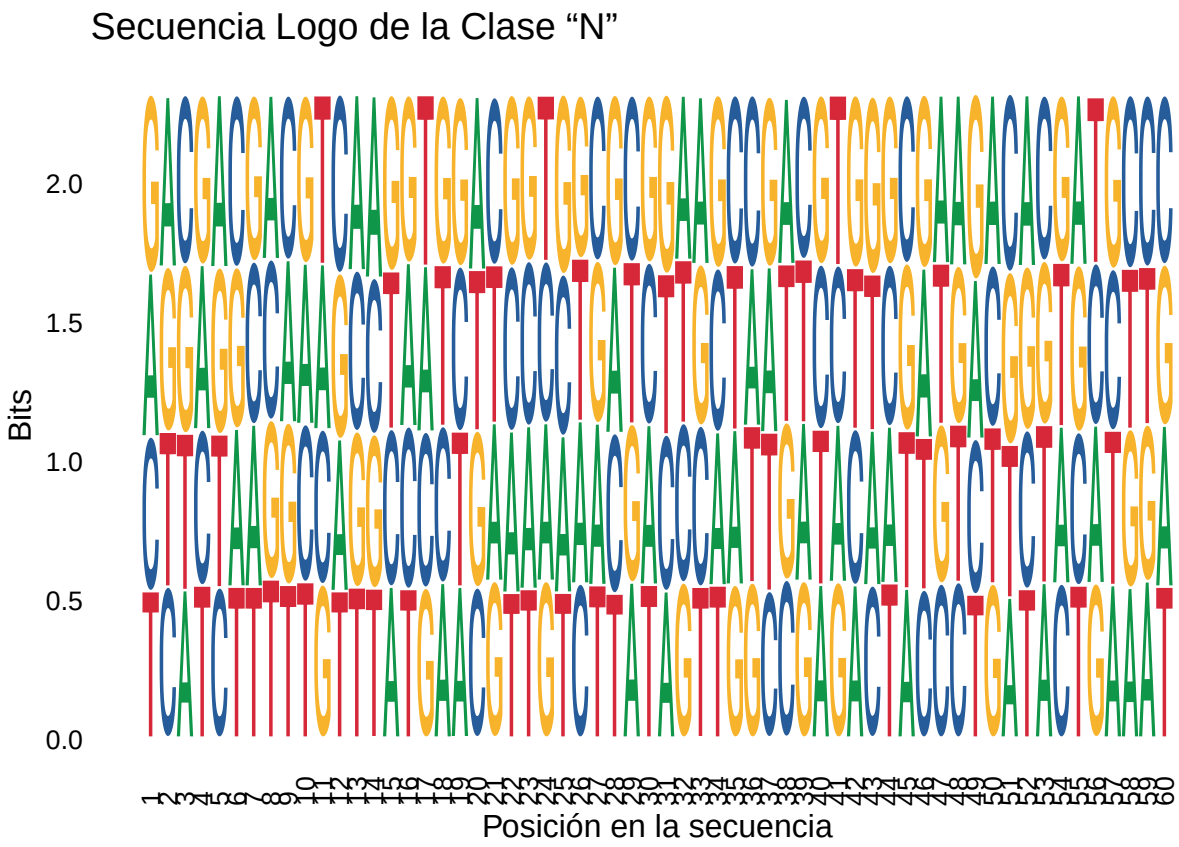
Para realizar esta representación se usa el paquete `ggseqlogo` descargable desde github o CRAN.

Secuencia logo para la clase N

```
seq_select <- str_trim(mydata[mydata$class=="N",3])
p1=ggseqlogo(seq_select, method = 'bits', col_scheme='nucleotide')+
   theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=0)) +
  labs(title = "Secuencia Logo de la Clase "N"") +
  xlab("Posición en la secuencia") + ylab("Bits")
```

```
Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> = "none")`
instead.
```

```
print(p1)
```



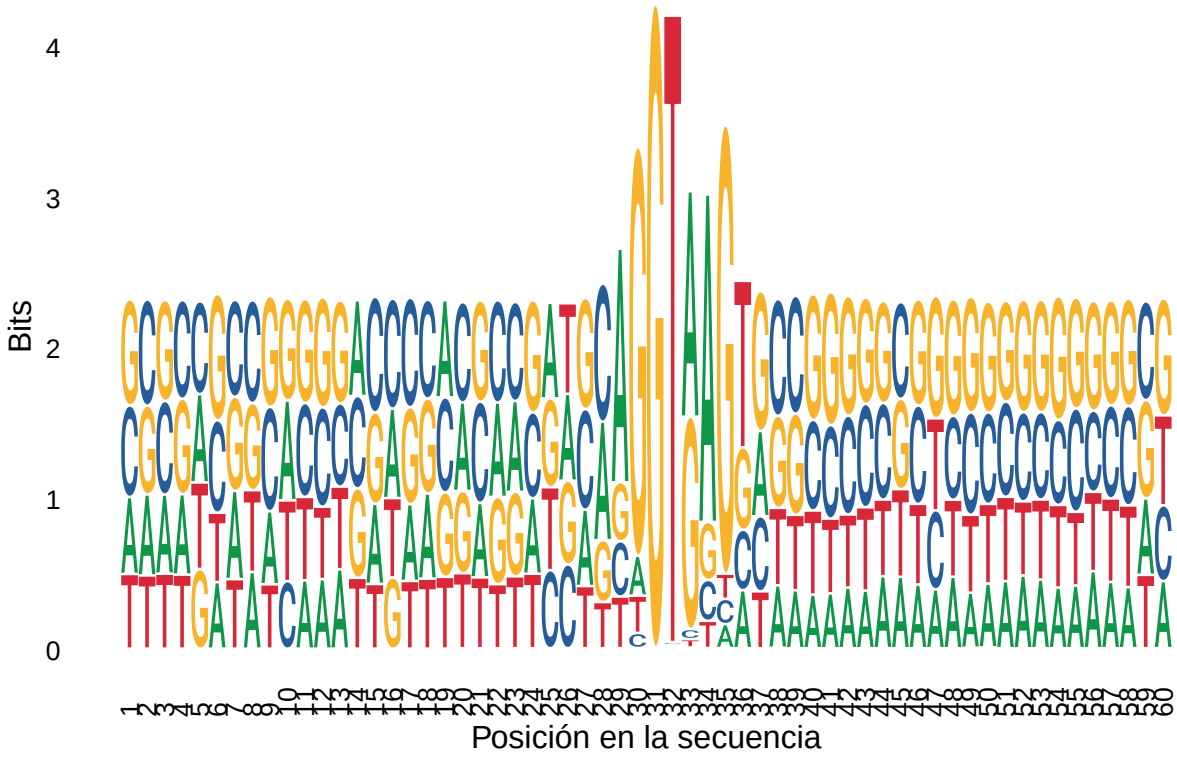
El patrón mostrado es muy uniforme, no hay ninguna posición con algún nucleótido predominante. Además, los valores obtenidos de bits son bajos.

Secuencia logo para la clase EI

```
seq_select <- str_trim(mydata[mydata$class=="EI",3])
p1=ggseqlogo(seq_select, method = 'bits', col_scheme='nucleotide')+
   theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=0)) +
  labs(title = "Secuencia Logo de la Clase "EI"") +
  xlab("Posición en la secuencia") + ylab("Bits")
```

```
Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> = "none")`
instead.
```

```
print(p1)
```



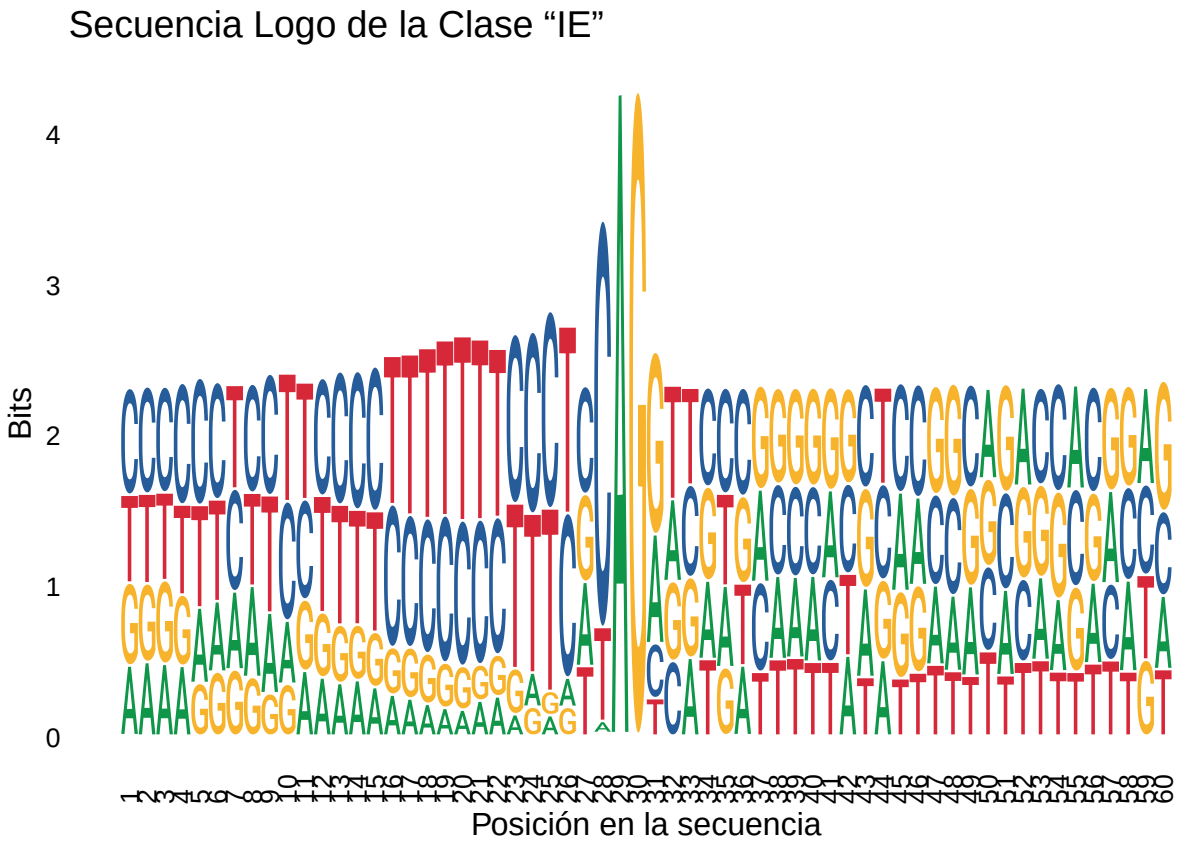
En cambio, para esta clase se observa claramente un patrón menos variable en las posiciones de 29 a 36 sobre el resto de posiciones. Además, estas posiciones tienen un valor de bits más alto respecto al resto de posiciones. En particular, los nucleótidos **GGT [A / G] AG** entre las posiciones 30 a 35.

Secuencia logo para la clase IE

```
seq_select <- str_trim(mydata[mydata$class=="IE",3])
p1=ggseqlogo(seq_select, method = 'bits', col_scheme='nucleotide')+
   theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=0)) +
  labs(title = "Secuencia Logo de la Clase "IE"") +
  xlab("Posición en la secuencia") + ylab("Bits")
```

```
Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> = "none")`
instead.
```

```
print(p1)
```



Para esta clase se observa claramente un patrón menos variable en las posiciones de 28 a 30, **CAG**, sobre el resto de posiciones. Además, estas posiciones tienen un valor de bits más alto respecto al resto de posiciones.

Observar que los `splices junctions` tienen una posición central con Guanina siempre.

Referencias

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