Machine Learning (PEC3)

Diciembre 2020

Secuencias promotoras en E. Coli

Los promotores son secuencias de ADN que afectan la frecuencia y ubicación del inicio de la transcripción a través de la interacción con la ARN polimerasa.

Este estudio se basa en los ficheros obtenidos de:

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Para más información, se puede recurrir a la siguiente referencia acerca del estudio de promotores en E. Coli: Harley, C. and Reynolds, R. 1987. "Analysis of E. Coli Promoter Sequences." Nucleic Acids Research, 15:2343-2361

Los atributos del fichero de datos son:

1. Un símbolo de {+/-}, indicando la clase ("+" = promotor).
2. El nombre de la secuencia promotora. Las instancias que corresponden a no promotores se denominan por la posición genómica.
3. Las restantes 57 posiciones corresponden a la secuencia.

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, se usará la transformación denominada **one-hot encoding**.

El one-hot encoding representa cada nucleótido por un vector de 4 componentes, con 3 de ellas a 0 y una a 1 indicando el nucleótido. Pongamos por ejemplo, el nucleótido T se representa por (1,0,0,0), el nucleótido C por (0,1,0,0), el nucleótido G por (0,0,1,0) y el nucleótido A por (0,0,0,1).

Por tanto, para una secuencia de 57 nucleótidos, como en nuestro caso, se obtendrá un vector de 4*57=228 componentes, resultado de concatenar los vectores para cada uno de los 57 nucleótidos.

Una vez realizada la transformación, one-hot encoding el objetivo se trata de implementar distintos algoritmos vistos en el curso para predecir si la secuencia es un promotor o no, y comparar sus rendimientos.

Objetivo:

En esta PEC se analizan esto datos mediante la **implementación** de los diferentes **algoritmos estudiados**: *k-Nearest Neighbour, Naive Bayes, Artificial Neural Network, Support Vector Machine, Arbol de Decisión y Random Forest* para **predecir** si una secuencia de ADN es promotor o no.

Puntos importantes:

1. Implementar una función para realizar una transformación one-hot encoding de las secuencias del fichero de datos `promoters.txt`. En caso de no lograr la implementaación de dicha transformación, se puede utilizar el fichero `promoters_onehot.txt` con las secuencias codificados segun un one-hot para completar la actividad.
2. En cada algoritmo hay que realizar las siguientes etapas: 1) Entrenar el modelo 2) Predicción y Evaluación del algoritmo. Será necesario "tunear" diferentes valores de los hiperparámetros del algoritmo para posteriormente evaluar su rendimiento.

3. Se debe aplicar la misma selección de datos training y test en todos los algoritmos. Utilizando la semilla aleatoria `123`, para separar los datos en dos partes, una parte para training (67%) y otra parte para test (33%). Opcionalmente, se puede escoger otro tipo de partición del conjunto de training para hacer la validación como por ejemplo k-fold crossvalidation, boostrap, random splitting, etc. Lo que es importante es mantener la misma selección para todos los algoritmos.

4. En todos los casos se evalua la calidad del algoritmo con la información obtenida de la función `confusionMatrix()` del paquete `caret`.

5. Para la ejecución especifica de cada algoritmo se puede usar la función de cada algoritmo como se presenta en el libro de referencia o usar el paquete `caret` con los diferentes modelos de los algoritmos. O incluso, hacer una versión mixta.

6. Comentario sobre el informe dinámico. Una opción interesante del knitr es poner `cache=TRUE`. Por ejemplo:

   ```
   knitr::opts_chunk$set(echo = FALSE, comment = NULL, cache = TRUE)
   ```

   Con esta opción al ejecutar el informe dinámico crea unas carpetas donde se guardan los resultados de los procesos. Cuando se vuelve a ejecutar de nuevo el informe dinámico solo ejecuta codigo R donde se ha producido cambios, en el resto lee la información previamente descargada. Es una opción muy adecuada cuando la ejecución es muy costosa computacionalmente.

Informe de la PEC

Las soluciones se presentarán mediante un informe dinámico R markdown con la siguiente estructura:

1. Título: igual que el de la PEC, autor, fecha de creación e indice de apartados de la PEC.

2. Sección de lectura y de transformación de los datos. Obtención de los muestras de train y test. Recordar que un primer paso es, si hace falta, transformar las variables leidas al tipo de objeto R adecuado al tipo de variable. (*Puntuación*: 10%)

3. Sección de aplicación de cada algoritmo para la clasificación. Está formado por subsecciones que corresponden a cada algoritmo y en este orden: k-Nearest Neighbour (se explorarán los valores para el número de vecinos $k = 1, 3, 5, 7$), Naive Bayes (se explorará la opción de activar o no 'laplace'), Artificial Neural Network (se explorarán el número de nodos de la capa oculta $n = 4, 5$), Support Vector Machine (se explorarán la funciones kernel lineal y rbf), Árbol de Clasificación (se explorará la opción de activar o no 'boosting') y Random Forest (se explorarán la opción de número de árboles $n = 50, 100$). (*Puntuación*: 60%)

   En cada algoritmo hay que realizar las etapas mencionadas anteriormente.

4. Sección de conclusión y discusión sobre el rendimiento, interpretabilidad, ... de los algoritmos para el problema tratado. Proponer que modelo o modelos son los mejores. (*Puntuación*: 20%)

Un característica que se valorará es hasta que punto es el informe "dinámico". En el sentido de adaptarse el informe a cambios en los datos, es decir, si el fichero de datos cambia el informe se adapta a los nuevos resultados. (*Puntuación*: 10%)

Se subiran al registro de entregas un **zip** con los siguientes ficheros:

1. Fichero ejecutable (.Rmd) que incluya un texto explicativo que detalle los pasos implementados en el script y el código de los análisis. No olvidar de incluir todos los ficheros complementarios que hagan falta para la correcta ejecución: *ficheros de datos, fichero de bibliografia, imagenes, ...*

   NOTA: Para facilitar la ejecución, no usar un ruta fija para la lectura del fichero, asociarlo al area de trabajo donde este el fichero .Rmd.

2. Informe (pdf) resultado de la ejecución del fichero Rmd anterior.

Antes de enviar el zip, se recomienda **verificar la reproducibilidad del fichero .Rmd** para obtener el informe en formato pdf sin ninguna dificultad.

Lectura y de transformación de los datos

```
library(knitr, quietly = TRUE)
```

Inicialmente definimos una función R que nos permitirá aplicar la transformación one-hot encoding.

```
################## data transformation
# There are 4 allowed letters
alphabet_nuc<-c("t","c","g","a")

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

Formato matricial

```
a <- "gataca"
c <- t(one.hot(a, alphabet = alphabet_nuc))
c
```

```
##   [,1] [,2] [,3] [,4]
## g    0    0    1    0
## a    0    0    0    1
## t    1    0    0    0
## a    0    0    0    1
## c    0    1    0    0
## a    0    0    0    1
```

Formato vectorial

```
#Reshape to vector
c <- t(c)
dim(c) <- prod(dim(c))
c
```

```
##  [1] 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1
```

Lectura de los datos

```
df<-read.delim("promoters.txt",head=F,sep=",")
```

Transformamos los datos a una codificación one-hot.

```
o<-data.frame(inic=rep(0,228))

for(k in 1:105){

a<-df[k,3]
a
w<-t(one.hot(a, alphabet = alphabet_nuc))

w <- t(w)
dim(w) <- prod(dim(w))
w<-as.data.frame(as.vector(w))
```

```
colnames(w)<-paste0("p",k)
o<-cbind(o,w)



}
o<-o[,-1]
o<-as.data.frame(t(o))
```

Algoritmos para la clasificación

Antes de aplicar los distintos algoritmos de clasificación, obtendremos los conjuntos de entrenamiento (train) y de test.

Datos de Training y Test

```
seed.cl<-set.seed(123)
train<-sample(105,70)
x.train <- o[train,]
x.test  <- o[-train,]
y.train<-as.factor(df[train,1])
y.test<-as.factor(df[-train,1])
```

Algoritmo K-NN

Para este algoritmo el hiperparámetro que hay que validar es k el número de vecinos que se toma para la clasificacion.

```
library(caret)
```

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

```
## Loading required package: ggplot2
```

```
library(class)

# grid de valores k
Hyper<-c(1,3,5,7)

knn_model_1<-knn(x.train, x.test, y.train, k =Hyper[1])
knn_model_1
```

```
##  [1] + + + + + + + + + + + + + + + + + - - - - + - + + - + - + - + - + - + - - +
## Levels: - +
```

```
knn_model_2<-knn(x.train, x.test, y.train, k = Hyper[2])
knn_model_2
```

```
##  [1] + + + + + + + + + + + + + - + + - - - - + - - - - + - + - - - - + + - -
## Levels: - +
```

```
knn_model_3<-knn(x.train, x.test, y.train, k = Hyper[3])
knn_model_3
```

```
##  [1] + + + + + + + + + + + + + - + + - - - - + - + - - + - + + + - + + - -
## Levels: - +
```

```
knn_model_4<-knn(x.train, x.test, y.train, k = Hyper[4])
knn_model_4
```

```
##  [1] + + + + + + + + + + + + + + + + + - - - - + - - - - + - + - - - + + - -
## Levels: - +
```

```
CM_knn_1<-confusionMatrix(knn_model_1, y.test)
CM_knn_2<-confusionMatrix(knn_model_2, y.test)
CM_knn_3<-confusionMatrix(knn_model_3, y.test)
CM_knn_4<-confusionMatrix(knn_model_4, y.test)
Resum_knn<-rbind(round(CM_knn_1$overall[1:4],3),
                 round(CM_knn_2$overall[1:4],3),
                 round(CM_knn_3$overall[1:4],3),
                 round(CM_knn_4$overall[1:4],3))
Resum_knn<-cbind(Hyper,Resum_knn)
kable(Resum_knn, caption= paste("Algoritmo kNN",sep=""))
```

Table 1: Algoritmo kNN

Hyper	Accuracy	Kappa	AccuracyLower	AccuracyUpper
1	0.743	0.491	0.567	0.875
3	0.800	0.602	0.631	0.916
5	0.714	0.434	0.537	0.854
7	0.829	0.659	0.664	0.934

De la tabla de precisiones (Accuracy) se observa que el valor de k que proporciona una mayor precisión es $k = 7$. Podemos visualizar la matriz de confusión para dicho valor de k.

```
CM_knn_4
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -  +
##          - 13  1
##          +  5 16
##
##                Accuracy : 0.8286
##                  95% CI : (0.6635, 0.9344)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : 0.0001126
##
##                   Kappa : 0.6591
##
##  Mcnemar's Test P-Value : 0.2206714
##
##             Sensitivity : 0.7222
##             Specificity : 0.9412
##          Pos Pred Value : 0.9286
##          Neg Pred Value : 0.7619
##              Prevalence : 0.5143
##          Detection Rate : 0.3714
##    Detection Prevalence : 0.4000
##       Balanced Accuracy : 0.8317
```

```
##
##        'Positive' Class : -
##
```

Algoritmo Naive Bayes

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.3
```

Para el Naive Bayes, el hiperparámetro es aplicar suavizado laplace o no.

```
NB1 <- naiveBayes(x.train, y.train, type="raw", laplace=0)
NB2 <- naiveBayes(x.train, y.train, type="raw", laplace=1)
```

```
predNB1 <- predict(NB1, x.test, type="class")
predNB2 <- predict(NB2, x.test, type="class")
evalNB1 <- confusionMatrix(predNB1, y.test)
evalNB2 <- confusionMatrix(predNB2, y.test)
```

```
Hyper <- data.frame(Hyper=c(0,1))
resumNB <- rbind(round(evalNB1$overall[1:4],3),round(evalNB2$overall[1:4],3))
resumNB <- cbind(Hyper,resumNB)
kable(resumNB, caption= paste("Algoritmo Naive Bayes ",
sep=""))
```

Table 2: Algoritmo Naive Bayes

Hyper	Accuracy	Kappa	AccuracyLower	AccuracyUpper
0	0.886	0.77	0.733	0.968
1	0.886	0.77	0.733	0.968

Se observa que la aplicación de laplace o no, no tiene incidencia en esta problema. Obtengamos la matriz de confusión.

```
evalNB2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -  +
##          - 17  3
##          +  1 14
##
##                Accuracy : 0.8857
##                  95% CI : (0.7326, 0.968)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : 3.724e-06
##
##                   Kappa : 0.7705
##
##  Mcnemar's Test P-Value : 0.6171
##
##             Sensitivity : 0.9444
##             Specificity : 0.8235
```

```
##          Pos Pred Value : 0.8500
##          Neg Pred Value : 0.9333
##              Prevalence : 0.5143
##          Detection Rate : 0.4857
##    Detection Prevalence : 0.5714
##       Balanced Accuracy : 0.8840
##
##        'Positive' Class : -
##
```

Algoritmo Neural Networks

```
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 4.0.3
```

Obtengamos una codificación one-hot de la variable respuesta.

```
df<-cbind(y.train,x.train)
df$cl1<-ifelse(y.train=="+",1,0)
df$cl2<-ifelse(y.train!="+",1,0)
```

Preparamos la sintaxis de la fórmula del modelo NN

```
resp<- names(df[230:231])
feat<- names(df[2:229])
resp_var <-paste(resp, collapse=" + ")
(fmla <- as.formula(paste(resp_var, " ~ ",  paste(feat, collapse= "+"))))
```

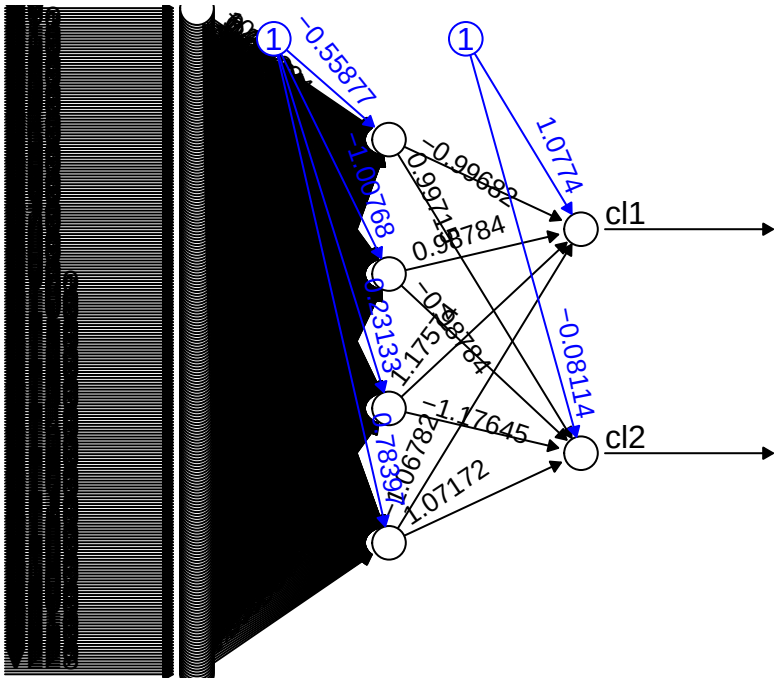
```
## cl1 + cl2 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 +
##     V11 + V12 + V13 + V14 + V15 + V16 + V17 + V18 + V19 + V20 +
##     V21 + V22 + V23 + V24 + V25 + V26 + V27 + V28 + V29 + V30 +
##     V31 + V32 + V33 + V34 + V35 + V36 + V37 + V38 + V39 + V40 +
##     V41 + V42 + V43 + V44 + V45 + V46 + V47 + V48 + V49 + V50 +
##     V51 + V52 + V53 + V54 + V55 + V56 + V57 + V58 + V59 + V60 +
##     V61 + V62 + V63 + V64 + V65 + V66 + V67 + V68 + V69 + V70 +
##     V71 + V72 + V73 + V74 + V75 + V76 + V77 + V78 + V79 + V80 +
##     V81 + V82 + V83 + V84 + V85 + V86 + V87 + V88 + V89 + V90 +
##     V91 + V92 + V93 + V94 + V95 + V96 + V97 + V98 + V99 + V100 +
##     V101 + V102 + V103 + V104 + V105 + V106 + V107 + V108 + V109 +
##     V110 + V111 + V112 + V113 + V114 + V115 + V116 + V117 + V118 +
##     V119 + V120 + V121 + V122 + V123 + V124 + V125 + V126 + V127 +
##     V128 + V129 + V130 + V131 + V132 + V133 + V134 + V135 + V136 +
##     V137 + V138 + V139 + V140 + V141 + V142 + V143 + V144 + V145 +
##     V146 + V147 + V148 + V149 + V150 + V151 + V152 + V153 + V154 +
##     V155 + V156 + V157 + V158 + V159 + V160 + V161 + V162 + V163 +
##     V164 + V165 + V166 + V167 + V168 + V169 + V170 + V171 + V172 +
##     V173 + V174 + V175 + V176 + V177 + V178 + V179 + V180 + V181 +
##     V182 + V183 + V184 + V185 + V186 + V187 + V188 + V189 + V190 +
##     V191 + V192 + V193 + V194 + V195 + V196 + V197 + V198 + V199 +
##     V200 + V201 + V202 + V203 + V204 + V205 + V206 + V207 + V208 +
##     V209 + V210 + V211 + V212 + V213 + V214 + V215 + V216 + V217 +
##     V218 + V219 + V220 + V221 + V222 + V223 + V224 + V225 + V226 +
##     V227 + V228
```

El hiperparámetro que se considera es el número de nodos de la capa oculta.

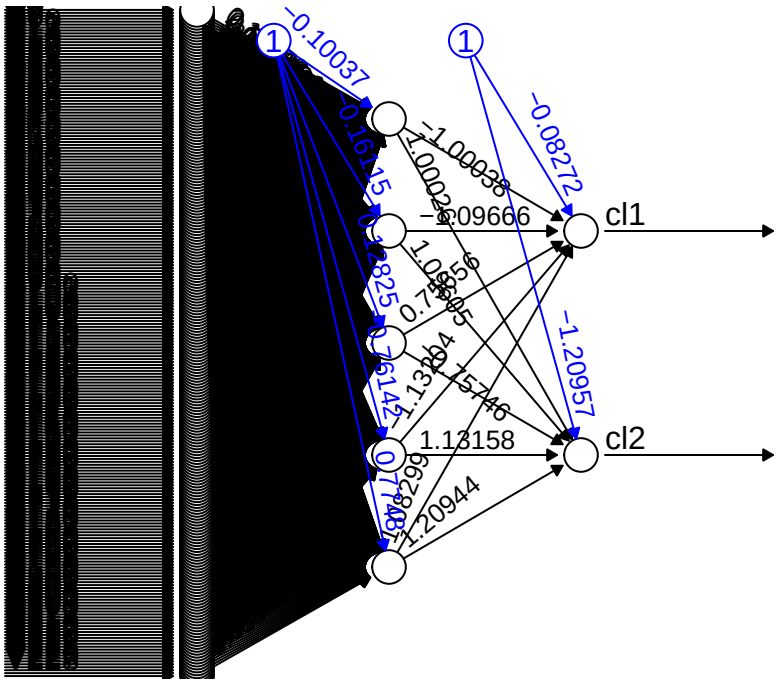
```
set.seed(123) # to guarantee repeatable results
nn_model_1 <- neuralnet(fmla,
                        data = df[,c(2:231)],
                        hidden=4)

nn_model_2 <- neuralnet(fmla,
                        data = df[,c(2:231)],
                        hidden=5)
```

```
plot(nn_model_1, rep="best")
```




```
plot(nn_model_2, rep="best")
```




```
# obtain model results
nn_model_1_results <- compute(nn_model_1, x.test)$net.result
nn_model_2_results <- compute(nn_model_2, x.test)$net.result

# Put multiple binary output to categorical output
maxidx <- function(arr) {
  return(which(arr == max(arr)))
}
idx <- apply(nn_model_1_results , 1, maxidx)
prediction <- factor(idx,levels=1:2,labels=c("+","-"))
prediction<-relevel(prediction, c("-"))
res1 <- table(prediction, y.test)


idx <- apply(nn_model_2_results , 1, maxidx)
prediction <- factor(idx,levels=1:2,labels=c("+","-"))
prediction<-relevel(prediction, c("-"))
res2 <- table(prediction, y.test)
```

Obtengamos las matrices de confusión de cada modelo.

```
evalNN_1<- confusionMatrix(res1)
evalNN_2<- confusionMatrix(res2)
```

```
Hyper <- data.frame(Hyper=c(4,5))
resumNN <- rbind(round(evalNN_1$overall[1:4],3),round(evalNN_2$overall[1:4],3))
resumNN <- cbind(Hyper,resumNN)
kable(resumNN, caption= paste("Algoritmo NN ",
sep=""))
```

Table 3: Algoritmo NN

Hyper	Accuracy	Kappa	AccuracyLower	AccuracyUpper
4	0.657	0.316	0.478	0.809
5	0.857	0.715	0.697	0.952

Se observa claramente que el mejor modelo NN es el que tiene 5 nodos en la capa oculta. Mostramos la matriz de confusión correspondiente a este modelo NN.

```
evalNN_2
```

```
## Confusion Matrix and Statistics
##
##           y.test
## prediction  -  +
##          - 14  1
##          +  4 16
##
##                Accuracy : 0.8571
##                  95% CI : (0.6974, 0.9519)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : 2.275e-05
##
##                   Kappa : 0.7154
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.7778
##             Specificity : 0.9412
##          Pos Pred Value : 0.9333
##          Neg Pred Value : 0.8000
##              Prevalence : 0.5143
##          Detection Rate : 0.4000
##    Detection Prevalence : 0.4286
##       Balanced Accuracy : 0.8595
##
##        'Positive' Class : -
##
```

Algoritmo SVM

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

Para los modelos SVM el hiperparámtero es el tipo de kernel, que se limitará a los tipos vanilla (=lineal) o rbf (=gausiano).

```
mydata_model1 <- ksvm(y.train ~ ., data = x.train,kernel = "vanilla")
```

```
##  Setting default kernel parameters
```

```
mydata_model2 <- ksvm(y.train ~ ., data = x.train,kernel = "rbf")
```

```
# predictions on testing dataset
mydata_predict1 <- predict(mydata_model1, x.test)
res1 <- table(mydata_predict1, y.test)
svm_1<-confusionMatrix(res1)

mydata_predict2 <- predict(mydata_model2, x.test)
res2 <- table(mydata_predict2, y.test)
svm_2<-confusionMatrix(res2)
```

```
Hyper <- data.frame(Hyper=c("lineal", "rbf"))
resumSVM <- rbind(round(svm_1$overall[1:4],3),round(svm_2$overall[1:4],3))
resumSVM <- cbind(Hyper,resumSVM)
kable(resumSVM, caption= paste("Algoritmo SVM ",
sep=""))
```

Table 4: Algoritmo SVM

Hyper	Accuracy	Kappa	AccuracyLower	AccuracyUpper
lineal	0.886	0.772	0.733	0.968
rbf	0.943	0.886	0.808	0.993

Se observa una mayor precisión (0.943) con el tipo rbf, siendo la matriz de confusión.

```
svm_2
```

```
## Confusion Matrix and Statistics
##
##                 y.test
## mydata_predict2  -  +
##                - 17  1
##                +  1 16
##
##                Accuracy : 0.9429
##                  95% CI : (0.8084, 0.993)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : 4.406e-08
##
##                   Kappa : 0.8856
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9444
##             Specificity : 0.9412
##          Pos Pred Value : 0.9444
##          Neg Pred Value : 0.9412
##              Prevalence : 0.5143
##          Detection Rate : 0.4857
##    Detection Prevalence : 0.5143
##       Balanced Accuracy : 0.9428
##
##        'Positive' Class : -
```

##

Algoritmo Classification Tree

```
library(C50)
```

En este algoritmo el hiperparámetro que se considera es aplicar o no muestreo bootstrap.

```
df<-df[,1:229]
tree_model1 <- C5.0(y.train ~ ., data = df)
tree_model2 <- C5.0(y.train ~ ., data = df,trial=50)
```

```
prediction1 <- predict(tree_model1 , x.test)
evalC50 <- confusionMatrix(prediction1, y.test)
```

```
prediction2 <- predict(tree_model2 , x.test)
evalC50t <- confusionMatrix(prediction2, y.test)
```

```
Hyper <- data.frame(Hyper=c("SimpleTree", "Boosting"))
resumTree <- rbind(round(evalC50$overall[1:4],3),round(evalC50t$overall[1:4],3))
resumTree<- cbind(Hyper,resumTree)
kable(resumTree, caption= paste("Algoritmo Trees ",
sep=""))
```

Table 5: Algoritmo Trees

Hyper	Accuracy	Kappa	AccuracyLower	AccuracyUpper
SimpleTree	0.886	0.770	0.733	0.968
Boosting	0.914	0.828	0.769	0.982

Se observa que la aplicación de muestreo bootstrap conlleva un modelo con mayor precisión. Siendo la matriz de confusión.

```
evalC50t
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -  +
##          - 17  2
##          +  1 15
##
##                Accuracy : 0.9143
##                  95% CI : (0.7694, 0.982)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : 4.742e-07
##
##                   Kappa : 0.8282
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9444
##             Specificity : 0.8824
##          Pos Pred Value : 0.8947
##          Neg Pred Value : 0.9375
```

```
##              Prevalence : 0.5143
##          Detection Rate : 0.4857
##    Detection Prevalence : 0.5429
##       Balanced Accuracy : 0.9134
##
##        'Positive' Class : -
##
```

Algoritmo Random Forest

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

En Random Forest vamos a considerar el hipeparámetro **ntree** que corresponde al número de árboles que se incluirán en el modelo.

```
df<-df[,1:229]
mydata_model1 <- randomForest(y.train ~ ., data = df, ntree= 50)
mydata_model2 <- randomForest(y.train ~ ., data = df, ntree= 100)
```

```
prediction <- predict(mydata_model1 , x.test)
RF1<-confusionMatrix(prediction, y.test)
```

```
prediction <- predict(mydata_model2 , x.test)
RF2<-confusionMatrix(prediction, y.test)
```

```
Hyper <- data.frame(Hyper=c(50, 100))
resumRF <- rbind(round(RF1$overall[1:4],3),round(RF2$overall[1:4],3))
resumRF<- cbind(Hyper,resumRF)
kable(resumRF, caption= paste("Algoritmo Random Forest ",
sep=""))
```

Table 6: Algoritmo Random Forest

Hyper	Accuracy	Kappa	AccuracyLower	AccuracyUpper
50	0.886	0.770	0.733	0.968
100	0.914	0.828	0.769	0.982

Se observa que para el hiperparámetro **ntree** el valor que proporciona mayor precisión es 50. Obteniendo una precisión de 0.914. La matriz de confusión es:

```
RF1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -  +
```

```
##          - 17  3
##          +  1 14
##
##                Accuracy : 0.8857
##                  95% CI : (0.7326, 0.968)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : 3.724e-06
##
##                   Kappa : 0.7705
##
##  Mcnemar's Test P-Value : 0.6171
##
##             Sensitivity : 0.9444
##             Specificity : 0.8235
##          Pos Pred Value : 0.8500
##          Neg Pred Value : 0.9333
##              Prevalence : 0.5143
##          Detection Rate : 0.4857
##    Detection Prevalence : 0.5714
##       Balanced Accuracy : 0.8840
##
##        'Positive' Class : -
##
```

Conclusiones

```
algoritmos<- data.frame(algoritmo=c(rep("kNN",4), rep("Naive Bayes",2), rep("ANN",2),rep("SVM",2), rep(



kable(cbind(algoritmos,
            rbind(Resum_knn,resumNB,resumNN,resumSVM,resumTree,resumRF)),
            caption= paste("Algoritmos",sep=""))
```

Table 7: Algoritmos

algoritmo	Hyper	Accuracy	Kappa	AccuracyLower	AccuracyUpper
kNN	1	0.743	0.491	0.567	0.875
kNN	3	0.800	0.602	0.631	0.916
kNN	5	0.714	0.434	0.537	0.854
kNN	7	0.829	0.659	0.664	0.934
Naive Bayes	0	0.886	0.770	0.733	0.968
Naive Bayes	1	0.886	0.770	0.733	0.968
ANN	4	0.657	0.316	0.478	0.809
ANN	5	0.857	0.715	0.697	0.952
SVM	lineal	0.886	0.772	0.733	0.968
SVM	rbf	0.943	0.886	0.808	0.993
C5.0	SimpleTree	0.886	0.770	0.733	0.968
C5.0	Boosting	0.914	0.828	0.769	0.982
RF	50	0.886	0.770	0.733	0.968
RF	100	0.914	0.828	0.769	0.982

A partir de la precisión estimada mediante del conjunto de test, concluimos que el modelo basado en el algoritmo SVM con kernel gaussiano es el tiene mayor precisión. A continuación, y con la misma precisión, encontramos los algoritmos: Classification Trees con muestreo bootstrap y el algoritmo de Random Forest basado en 50 árboles. En el contexto de esta práctica donde las variables son fruto de una codificación one-hot no tiene mucha relevancia la interpretación de las variables, en otros contextos, en cambio puede tener mucho interés la interpretabilidad del modelo. Los modelos SVM con kernel no lineal no permiten de manera directa la interpretabilidad, a diferencia de los modelos basados en árboles. Así pues, podria tener interés valorar no sólo la precisión si no tambien la interpretabilidad para elegir un clasificador.