Curso : MACHINE LEARNING CON R

Tema : Modelos de Clasificación.

Docente : Victor Henostroza

**GUIA DE LABORATORIO 4**

**Objetivos del laboratorio:**

1. **Desarrollar un Modelo de Clasificación con Árbol de Decisión.**
2. **Desarrollar un Modelo de Clasificación con Bosques Aleatorios.**
3. **Desarrollar un Modelo de Clasificación con Máquinas de Soporte Vectorial.**
4. **Desarrollar un Modelo de Clasificación con K-Nearest Neighbors.**
5. **Desarrollar un Modelo de Clasificación con Naive Bayes.**
6. **Desarrollar un Modelo de Clasificación con Redes Neuronales.**

**---------------------------------------------------------------------------------------------------------**

**Objetivo 1:**

Para desarrollar nuestro modelo de clasificación con árbol de decisión utilizaremos el archivo “Cardiotocographic.csv”, el cual nos muestra 22 variables en total, las 21 primeras son las independientes y la última “**NSP**” es nuestra variable dependiente de clasificación. La variable respuesta tiene 3 posibles valores/factores, 1 significa estado normal en las mediciones de la frecuencia cardiaca del feto, 2 significa que se sospecha de una posible anormalidad en la frecuencia cardiaca del feto y 3 significa que el feto tiene alguna patología del corazón. Las demás variables se describen a continuación:

**# Clasificación con Árboles de Decisión**

**# Importar el dataset**

dataset = read.csv('Cardiotocographic.csv')

head(dataset)

str(dataset)

**# Codificar la variable de clasificación como factor**

dataset$NSP = factor(dataset$NSP, levels = c(1,2,3))

**# Dividir los datos en conjunto de entrenamiento y conjunto de test**

library(caTools)

set.seed(1234)

split = sample.split(dataset$NSP, SplitRatio = 0.80)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Ajustar el clasificador con el conjunto de entrenamiento**

# install.packages("rpart")

library(rpart)

classifier = rpart(formula = NSP ~ .,

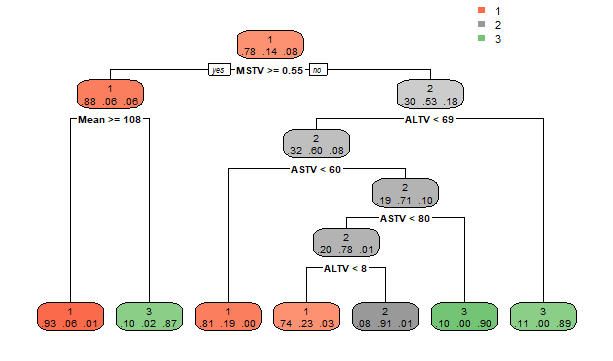
data = training\_set,

minsplit = 100,

minbucket = 10)

library(rpart.plot)

rpart.plot(classifier, extra=4)

****

Del gráfico, apreciamos como cada nodo muestra el valor de la variable respuesta NSP en su parte superior (1, 2 o 3), además, en la parte de debajo de cada nodo se muestra las probabilidades de que los datos hayan sido en ese subconjunto 1, 2 o 3 respectivamente, tomándose la mayor probabilidad para generar la variable respuesta.

**# Predicción de los resultados con el conjunto de testing**

y\_pred = predict(classifier, newdata = testing\_set[,-22],

type = "class")

> y\_pred

6 9 15 19 27 29 30 33 42 44 52 55 59 60 66 68

3 3 1 1 3 1 1 1 1 1 1 3 1 1 1 1

71 78 84 85 92 94 97 103 104 107 109 111 117 133 134 139

1 1 1 1 2 3 2 1 1 1 1 1 3 1 2 1

**# Crear la matriz de confusión**

cm = table(testing\_set[, 22], y\_pred)

> cm

y\_pred

1 2 3

1 327 2 2

2 22 37 0

3 2 2 31

**# Calculado ratio de aciertos del modelo**

ratio\_aciertos = (327+37+31)/425

> ratio\_aciertos

[1] 0.9294118

**Concluimos que nuestro modelo de árbol de decisión nos ha permitido clasificar correctamente el 92.94% de los casos.**

**Objetivo 2:**

Utilizaremos el mismo dataset 'Cardiotocographic.csv con fines comparativos para desarrollar nuestro modelo de clasificación con Random Forest.

**# Random Forest**

**# Importar el dataset**

dataset = read.csv('Cardiotocographic.csv')

dataset$NSP <- as.factor(dataset$NSP)

str(dataset)

table(dataset$NSP)

**# Dividir los datos en conjunto de entrenamiento y conjunto de test**

library(caTools)

set.seed(1234)

split = sample.split(dataset$NSP, SplitRatio = 0.80)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Ajustar el Random Forest con el conjunto de entrenamiento.**

#install.packages("randomForest")

library(randomForest)

classifier = randomForest(x = training\_set[,-22],

y = training\_set$NSP,

ntree = 300,

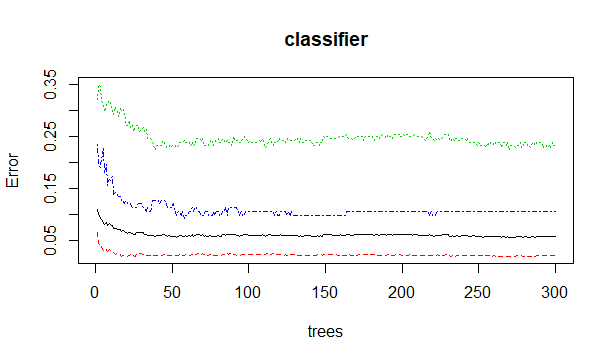
mtry = 8,

importance = TRUE,

proximity = TRUE)

**# Mostrar el ratio de errror del Random Forest**

plot(classifier)



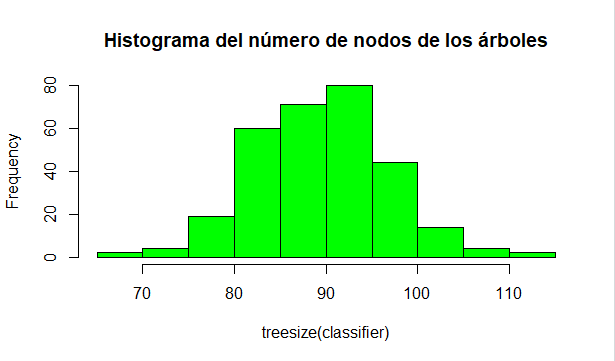
Del gráfico, apreciamos que con el aumento de árboles se reduce y estabiliza cada vez más el error. Es conveniente quedarnos con la menor cantidad de árboles posible cuando ya esté el error estable porque eso permitirá que nuestro modelo no se sobreajuste con el conjunto de entrenamiento. (Se deja como tarea probar con 100 árboles y con 200 árboles).

**# Histograma del Número de nodos para los árboles**

hist(treesize(classifier),

main = "No. of Nodes for the Trees",

col = "green")



Del gráfico, apreciamos que la mayoría de árboles tienen entre 80 y 100 nodos y aproximadamente se distribuyen normalmente las frecuencias de árboles respecto a su número de nodos.

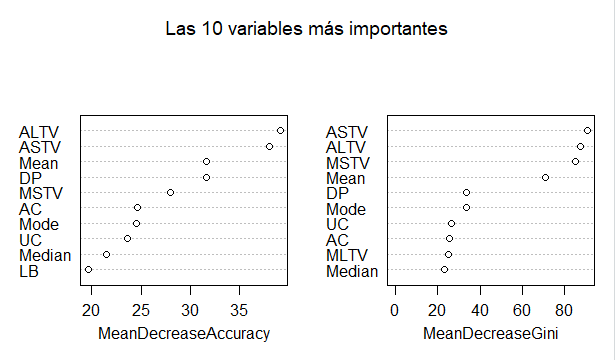
**# Importancia de las variables independientes en el modelo**

varImpPlot(classifier,

sort = T,

n.var = 10,

main = "Las 10 variables más importantes")



El primer gráfico es el más importante, muestra como disminuiría la exactitud de las medias de clasificación en caso faltaran las variables, las variables más significativas son las que se muestran más arriba del gráfico y las menos importantes más abajo.

El segundo gráfico mide la pureza de los nodos al final del árbol dejando las variables por importancia en el modelo, las variables más decisivas en cada nodo son las que se muestran más arriba y las menos importantes más abajo.

**# Número de veces que fue utilizada una variable en la clasificación**

varUsed(classifier)

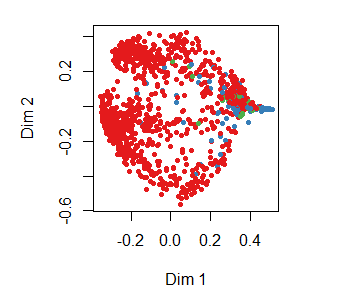
> varUsed(classifier)

[1] 1434 1036 1105 1812 483 58 768 2372 1328 2378 1671 1528 1519 1516 1166 383

[17] 1453 1778 1379 1026 494

**# Gráfica de escalamiento multidimensional de matriz de proximidad**

MDSplot(classifier, training\_set$NSP)

****

Del gráfico, apreciamos una proyección en dos dimensiones de cada uno de los factores elegidos en el conjunto de entrenamiento, esto solo sirve para ver cómo se generan zonas con mayores cantidades de puntos de cada color distinto correspondiente a los niveles del factor.

**# La siguiente instrucción presentará una matrix de confusión inicial y los errores en porcentaje que se comenten en la clasificación para cada uno de los posibles valores de NSP**

> print(classifier)

Call:

randomForest(x = training\_set[, -22], y = training\_set$NSP, ntree = 300, mtry = 8, importance = TRUE, proximity = TRUE)

Type of random forest: classification

Number of trees: 300

No. of variables tried at each split: 8

OOB estimate of error rate: 5.82%

Confusion matrix:

1 2 3 class.error

1 1295 22 7 0.02190332

2 53 181 2 0.23305085

3 9 6 126 0.10638298

**# Predicción de los resultados con el conjunto de testing**

y\_pred = predict(classifier, newdata = testing\_set[,-22])

> y\_pred

6 9 15 19 27 29 30 33 42 44 52 55 59 60 66 68

3 3 1 1 3 1 1 1 1 1 1 2 1 1 1 1

71 78 84 85 92 94 97 103 104 107 109 111 117 133 134 139

2 1 1 1 2 3 2 1 1 1 1 1 3 1 2 2

**# Crear la matriz de confusión**

cm = table(testing\_set[, 22], y\_pred)

> cm

y\_pred

1 2 3

1 326 4 1

2 11 47 1

3 1 3 31

**# Calculado ratio de aciertos del modelo**

ratio\_aciertos = (326+47+31)/425

> ratio\_aciertos

[1] 0.9505882

**Concluimos que nuestro modelo de Random Forest nos ha permitido clasificar correctamente el 95.06% de los casos, el cual es superior al modelo de Árbol de Decisión anterior.**

**Objetivo 3:**

Siguiendo con el mismo dataset, ahora haremos nuestro modelo de Máquinas de Soporte Vectorial con dos tipos de Kernel (transformaciones) la primera será con un Kernel Lineal y la segunda con un Kernel Gausiano.

**# SVM**

**# Importar el dataset**

dataset = read.csv('Cardiotocographic.csv')

dataset$NSP <- as.factor(dataset$NSP)

str(dataset)

table(dataset$NSP)

**# Dividir los datos en conjunto de entrenamiento y conjunto de test**

# install.packages("caTools")

library(caTools)

set.seed(1234)

split = sample.split(dataset$NSP, SplitRatio = 0.80)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Escalado de valores**

training\_set[,-22] = scale(training\_set[,-22])

testing\_set[,-22] = scale(testing\_set[,-22])

**# Ajustar el SVM de Kernel Lineal con el conjunto de entrenamiento (caso 1).**

#install.packages("e1071")

library(e1071)

classifier = svm(formula = NSP ~ .,

data = training\_set,

type = "C-classification",

kernel = "linear")

**# Ajustar el SVM de Kernel Radial con el conjunto de entrenamiento (caso 2).**

#install.packages("e1071")

library(e1071)

classifier2 = svm(formula = NSP ~ .,

data = training\_set,

type = "C-classification",

kernel = "radial")

**# Predicción de los resultados con el conjunto de testing (caso 1)**

y\_pred = predict(classifier, newdata = testing\_set[,-22])

> y\_pred

6 9 15 19 27 29 30 33 42 44 52 55 59 60 66 68

3 3 1 1 3 2 1 1 1 1 1 2 1 1 1 1

71 78 84 85 92 94 97 103 104 107 109 111 117 133 134 139

1 1 1 1 2 2 2 1 1 1 1 1 3 2 1 2

**# Predicción de los resultados con el conjunto de testing (caso 2)**

y\_pred2 = predict(classifier2, newdata = testing\_set[,-22])

> y\_pred2

6 9 15 19 27 29 30 33 42 44 52 55 59 60 66 68

3 3 1 1 1 1 1 1 1 1 1 2 1 1 1 1

71 78 84 85 92 94 97 103 104 107 109 111 117 133 134 139

1 1 1 1 2 2 2 1 1 1 1 1 3 2 2 2

**# Crear la matriz de confusión (caso 1)**

cm = table(testing\_set[, 22], y\_pred)

> cm

y\_pred

1 2 3

1 315 13 3

2 15 38 6

3 0 6 29

**# Crear la matriz de confusión (caso 2)**

cm2 = table(testing\_set[, 22], y\_pred2)

> cm2

y\_pred2

1 2 3

1 324 5 2

2 21 38 0

3 1 6 28

**# Calculado ratio de aciertos del modelo (caso 1)**

ratio\_aciertos = (315+38+29)/425

> ratio\_aciertos

[1] 0.8988235

**# Calculado ratio de aciertos del modelo (caso 2)**

ratio\_aciertos2 = (324+38+28)/425

> ratio\_aciertos2

[1] 0.9176471

**Concluimos que nuestro modelo de SVM ha generado un ratio de aciertos del 89.88% y de 91.77% para sus corridas con Kernel Lineal y Kernel Gausiano respectivamente.**

**Objetivo 4:**

Ahora realizaremos un modelo de clasificación con K-Nearest Neighbors para el mismo conjunto de datos de los ejemplos anteriores.

**# K - Nearest Neighbors (K-NN)**

**# Importar el dataset**

dataset = read.csv('Cardiotocographic.csv')

head(dataset)

str(dataset)

**# Codificar la variable de clasificación como factor**

dataset$NSP = factor(dataset$NSP, levels = c(1,2,3))

**# Dividir los datos en conjunto de entrenamiento y conjunto de test**

library(caTools)

set.seed(1234)

split = sample.split(dataset$NSP, SplitRatio = 0.80)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Escalado de valores**

training\_set[,-22] = scale(training\_set[,-22])

testing\_set[,-22] = scale(testing\_set[,-22])

**# Ajustar el clasificador con el conjunto de entrenamiento**

**# y hacer las predicciones con el conjunto de testing.**

library(class)

y\_pred = knn(train = training\_set[,-22],

test = testing\_set[,-22],

cl = training\_set[,22],

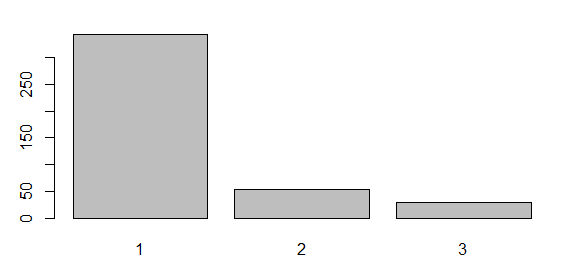
k = 5)

> y\_pred

[1] 3 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 3 2 2 2 1 1 1 1 1 1

[40] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 2 3 2 2 3 2 1 3 2 2 3 3 2 1

> plot(y\_pred)

****

Del gráfico, vemos que la mayor cantidad de predicciones caen en el valor 1 de la variable respuesta, seguida por el valor 2 y finalmente por el valor 3.

**# Crear la matriz de confusión**

cm = table(testing\_set[, 22], y\_pred)

> cm

y\_pred

1 2 3

1 324 4 3

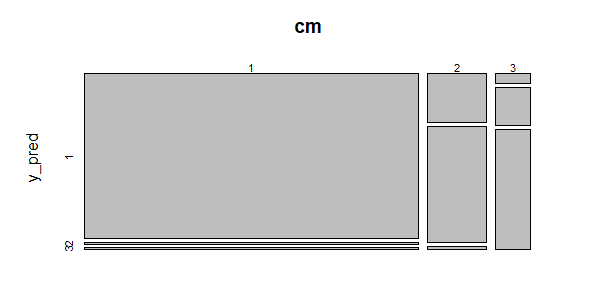
2 17 41 1

3 2 8 25

**# Calculado ratio de aciertos del modelo**

ratio\_aciertos = (324+41+25)/425

> plot(cm)



Del gráfico, apreciamos a través de la anchura de cada categoría como es que caen la mayoría de las predicciones (1 es el más ancho, seguido por el 2 y finalmente el 3), además, por columnas, se aprecian los datos pintados de acuerdo a la misma posición de datos en el matriz de confusión, siendo el valor de la diagonal principal quien genera mayores alturas correspondientes a más datos predichos.

> ratio\_aciertos

[1] 0.9176471

**Concluimos que nuestro modelo de clasificación con K-Nearest Neighbors nos dio un total de 91.77% de precisión en sus clasificaciones.**

**Objetivo 5:**

Ahora realizaremos un modelo de clasificación con Naive Bayes para el mismo conjunto de datos de los ejemplos anteriores.

**# Naive Bayes**

**# Importar el dataset**

dataset = read.csv('Cardiotocographic.csv')

head(dataset)

str(dataset)

**# Visualizando grupos de la variable dependiente (NSP) vs una variable independiente (ALTV)**

library(dplyr)

library(ggplot2)

pairs.panels(dataset[-22])

dataset %>%

ggplot(aes(x=NSP, y=ALTV, fill = NSP)) +

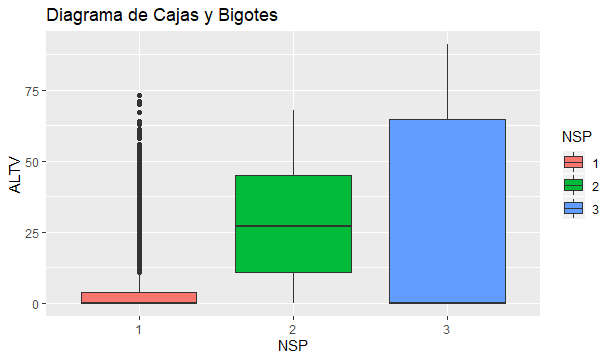
geom\_boxplot() +

ggtitle("Diagrama de Cajas y Bigotes")

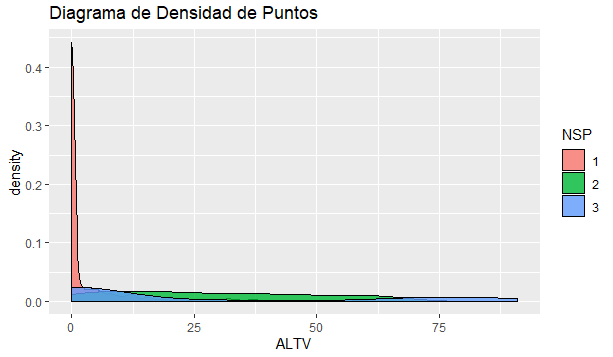
dataset %>% ggplot(aes(x=ALTV, fill = NSP)) +

geom\_density(alpha=0.8, color= 'black') +

ggtitle("Diagrama de Densidad de Puntos")



Del gráfico, apreciamos cuando nuestra variable respuesta NSP = 1 se da para valores ALTV muy cercanos a cero, en cambio, el para NSP = 2 contempla mucha variabilidad de ALTV (entre 10 y 40 aproximadamente), peor aún para NSP = 3, allí la variabilidad de ALTV está mucho más amplia (entre 0 y 63 aproximadamente). Además, vemos que los diagramas de cajas de NSP = 2 y NSP = 3 se superponen (esto puede generar una mala predicción a futuro de alguna de estas categorías).



Del gráfico, vemos la densidad de puntos para cada una de las tres clases del factor respecto a los valores de ALTV en el eje x, vemos que hay una región cercana a cero en ALTV donde casi siempre se está clasificando como 1 a NSP, sin embargo, hay una zona común entre 15 y 60 para ALTV en donde se superponen con no mucha claridad las observaciones 2 y 3 de NSP (esto puede hacer que alguno de los dos valores NSP pueda ser mal predicho a futuro).

**# Codificar la variable de clasificación como factor**

dataset$NSP = factor(dataset$NSP, levels = c(1,2,3))

**# Dividir los datos en conjunto de entrenamiento y conjunto de test**

library(caTools)

set.seed(1234)

split = sample.split(dataset$NSP, SplitRatio = 0.80)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Escalado de valores**

training\_set[,-22] = scale(training\_set[,-22])

testing\_set[,-22] = scale(testing\_set[,-22])

**# Ajustar el clasificador con el conjunto de entrenamiento.**

#install.packages("e1071")

library(naivebayes)

library(e1071)

classifier = naiveBayes(x = training\_set[,-22],

y = training\_set$NSP)

> classifier

Naive Bayes Classifier for Discrete Predictors

Call:

naiveBayes.default(x = training\_set[, -22], y = training\_set$NSP)

A-priori probabilities:

training\_set$NSP

1 2 3

0.77836567 0.13874192 0.08289242

De la línea anterior en consola, se aprecia que existe una alta tendencia de que ocurra más NSP = 1 que NSP = 2 o NSP = 3, siendo esta última la probabilidad más baja de ocurrencia.

**# Predicción de los resultados con el conjunto de testing**

y\_pred = predict(classifier, newdata = testing\_set[,-22])

> y\_pred

[1] 3 1 3 3 1 3 3 1 1 1 1 3 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1

[40] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 3 1 3 3 1 1 3 1 1 3 3 1 1

**# Crear la matriz de confusión**

cm = table(testing\_set[, 22], y\_pred)

> cm

y\_pred

1 2 3

1 318 0 13

2 50 0 9

3 5 0 30

**# Calculado ratio de aciertos del modelo**

ratio\_aciertos = (318+0+30)/425

> ratio\_aciertos

[1] 0.8188235

**Concluimos que nuestro modelo de clasificación Naive Bayes nos dio un total de 81.88% de precisión en sus clasificaciones, la más baja en relación a los métodos anteriores hasta el momento.**

**Objetivo 6:**

Finalmente, haremos nuestro último modelo de clasificación con redes neuronales, en este caso utilizaremos solo una capa interna con 5 nodos por fines de procesamiento (verán que demorará bastante en que converja el modelo).

**# Red Neuronal**

**# Importar el dataset**

dataset = read.csv('Cardiotocographic.csv')

dataset$NSP <- as.factor(dataset$NSP)

str(dataset)

table(dataset$NSP)

**# Dividir los datos en conjunto de entrenamiento y conjunto de test**

# install.packages("caTools")

library(caTools)

set.seed(1234)

split = sample.split(dataset$NSP, SplitRatio = 0.80)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Escalado de valores**

training\_set[,-22] = scale(training\_set[,-22])

testing\_set[,-22] = scale(testing\_set[,-22])

**# Ajustar la Red Neuronal con el conjunto de entrenamiento**

library(nnet)

set.seed(1234)

classifier <- multinom(NSP~.,

data=training\_set,

maxit = 500,

trace=T)

**# Predicción de los resultados con el conjunto de testing**

y\_pred = predict(classifier, type = "class", newdata = testing\_set[,-22])

> y\_pred

[1] 3 3 1 1 3 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 3 2 2 2 2 1 1 1 1 1 1

[40] 1 1 2 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 3 2 2 2 3 2 2 3 3 2 3 3 3 3 3 1 1

**# Crear la matriz de confusión**

**cm = table(testing\_set[, 22], y\_pred)**

> cm

y\_pred

1 2 3

1 314 12 5

2 17 36 6

3 0 7 28

**# Calculado ratio de aciertos del modelo**

**ratio\_aciertos = (314+36+28)/425**

> ratio\_aciertos

[1] 0.8894118

Concluimos que nuestro modelo de red neuronal nos dio un total de aciertos de 88.94% en su clasificación.

**Conclusión Final:**

Luego de haber realizado los distintos modelos de clasificación para nuestro ejemplo del dataset “Cardiotocographic”, el mejor modelo será el que obtuvo el mayor ratio de aciertos luego de habérsele aplicado las matrices de confusión a cada uno de ellos, por lo tanto, el mejor modelo fue el de Random Forest con 95.06% de ratio de aciertos