Unidad 5: Predicción de la malignidad/benignidad de un cáncer de mama usando el algoritmo Redes Neuronales Artificiales

Amelia Martínez Sequera

19/11/2020

Índice de contenidos.

1. Objetivo

Se quiere predecir el tipo de cáncer de mama (benigno/maligno) en función de las variables biológicas del cáncer de mama.

2. Algoritmo Redes Neuronales Artificiales.

2.1. ¿Qué es?. Características.

Una red neuronal artificial (ANN) modela la relación entre un conjunto de entradas señales y una señal de salida utilizando un modelo derivado de nuestra comprensión de cómo un el cerebro biológico responde a los estímulos de las entradas sensoriales. Así como un cerebro usa una red de células interconectadas, llamadas neuronas, para crear un procesador paralelo masivo, ANN utiliza una red de neuronas o nodos artificiales para resolver problemas de aprendizaje.

A pesar de su complejidad, se pueden aplicar fácilmente a problemas del mundo real. Son aprendices versátiles que se pueden aplicar a casi cualquier tarea de aprendizaje: clasificación, predicción numérica e incluso sin supervisión reconocimiento de patrones.

Características:

Las redes neuronales utilizan neuronas como bloques de construcción para construir modelos complejos de datos. Aunque existen numerosas variantes de redes neuronales, cada una puede definirse en términos de las siguientes características:

• Una **función de activación**, que transforma las señales de entrada combinadas de una neurona en una sola señal de salida para ser transmitida más en la red

• Una **topología** (o arquitectura) de red, que describe el número de neuronas en el modelo, así como el número de capas y la forma en que están conectados (dirección en la que viaja la información)

• El **algoritmo de entrenamiento** que especifica cómo se establecen en orden los pesos de conexión para inhibir o excitar neuronas en proporción a la señal de entrada

2.2. Tabla de fortalezas y debilidades del algoritmo ANN (*multilayer feedforward networks that use the backpropagation algorithm*)

Fortalezas	Debilidades
Puede adaptarse a problemas de predicción de clase o numérica	Computacionalmente, extremadamente intensivo y lento de entrenar, particularmente si la topología de la red es compleja
Capaz de modelar patrones más complejos que cualquier otro algoritmo	Propenso a sobreajustar el entrenamiento
Hace pocas suposiciones sobre las relaciones subyacentes de los datos	Da como resultado un modelo de caja negra compleja que es difícil, si no imposible, interpretar.

3. Ejemplo algoritmo ANN

Step1. Data set.

Se ha registrado la información de 9 variables biológicas y el tipo de cáncer: benign / malignant (ver el dataset) en 683 canceres de mama. Los valores de las 9 variables biológicas predictoras son ordinales de 0 a 10. El fichero se llama BreastCancer2.csv

Step2. Explorando y procesando los datos.

```
setwd("~/UOC/ML/unidad5")
BreastCancer2 <- read.csv("BreastCancer2.csv")
str(BreastCancer2)
```

```
## 'data.frame':    683 obs. of  10 variables:
##  $ Cl.thickness   : int  5 5 3 6 4 8 1 2 2 4 ...
##  $ Cell.size      : int  1 4 1 8 1 10 1 1 1 2 ...
##  $ Cell.shape     : int  1 4 1 8 1 10 1 2 1 1 ...
##  $ Marg.adhesion  : int  1 5 1 1 3 8 1 1 1 1 ...
##  $ Epith.c.size   : int  2 7 2 3 2 7 2 2 2 2 ...
##  $ Bare.nuclei    : int  1 10 2 4 1 10 10 1 1 1 ...
##  $ Bl.cromatin    : int  3 3 3 3 3 9 3 3 1 2 ...
##  $ Normal.nucleoli: int  1 2 1 7 1 7 1 1 1 1 ...
##  $ Mitoses        : int  1 1 1 1 1 1 1 1 5 1 ...
##  $ Class          : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...
```

Los algoritmos ANN funcionan mejor cuando los datos de entrada se escalan a un rango estrecho alrededor de cero, y aquí vemos valores que van desde cero hasta 10. Normalmente, la solución a este problema es cambiar la escala de los datos con una normalización o función de estandarización. Si los datos siguen una distribución normal se puede usar la estandarización a través de la función scale () incorporada de R. Por otro lado, si los datos siguen una distribución uniforme o siguen una distribución no normal, la normalización a un rango 0-1 puede ser más apropiada. En este caso, usaremos este último.

```
normalize <- function(x) {
return((x - min(x)) / (max(x) - min(x)))
}
```

La función neuralnet no admite variables factor o categóricas. Por tanto, hay que transforma la variable "tipo de cáncer" a binaria. Para este cometido, se debe crear dos variables que sustituyen a la variable original. Una será la variable cáncer benigno (B) que tiene valores (TRUE o 1) en los casos "benignos" y (FALSE o 0) en los casos malignos. De manera semejante, pero contraria se crea la variable cáncer maligno (M). Observar que variables lógicas (TRUE/FALSE) si que admite la función neuralnet porque después las transforma a binarias internamente. Al crear dos variables binarias como variables a predecir la red neuronal tendrá 2 nodos de output. El modelo en notación formula seria una cosa como:

M + B ~ Cl.thickness + Cell.size + Cell.shape + Marg.adhesion + Epith.c.size + Bare.nuclei + Bl.cromatin + Normal.nucleoli + Mitoses

```
BreastCancer2$Benign <- ifelse(BreastCancer2$Class=="benign", 1, 0)
BreastCancer2$Malignant <- 1 - BreastCancer2$Benign
str(BreastCancer2)
```

```
## 'data.frame':    683 obs. of  12 variables:
##  $ Cl.thickness   : int  5 5 3 6 4 8 1 2 2 4 ...
##  $ Cell.size      : int  1 4 1 8 1 10 1 1 1 2 ...
##  $ Cell.shape     : int  1 4 1 8 1 10 1 2 1 1 ...
##  $ Marg.adhesion  : int  1 5 1 1 3 8 1 1 1 1 ...
##  $ Epith.c.size   : int  2 7 2 3 2 7 2 2 2 2 ...
##  $ Bare.nuclei    : int  1 10 2 4 1 10 10 1 1 1 ...
##  $ Bl.cromatin    : int  3 3 3 3 3 9 3 3 1 2 ...
##  $ Normal.nucleoli: int  1 2 1 7 1 7 1 1 1 1 ...
##  $ Mitoses        : int  1 1 1 1 1 1 1 1 5 1 ...
##  $ Class          : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...
##  $ Benign         : num  1 1 1 1 1 0 1 1 1 1 ...
##  $ Malignant      : num  0 0 0 0 0 1 0 0 0 0 ...
```

```
BC2<- BreastCancer2[,-10]
BC2_norm <- as.data.frame(lapply(BC2, normalize))
# aplicamos a cada columna nuestra función de normalización
```

Comprobamos que los datos han quedado correctamente normalizados entre 0 y 1:

```
summary(BC2_norm$Bl.cromatin)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1111  0.2222  0.2717  0.4444  1.0000
```

El dataset se dividirá en 2/3 training y 1/3 test:

```
set.seed(12345)
train<-sample(1:nrow(BC2_norm),round(2*nrow(BC2_norm)/3,0))
bc2train <- BC2_norm[train,]
bc2test <- BC2_norm[-train,]
```

Step3. Entrenando el modelo con los datos.

Antes de ejecutar la función neuralnet() poner como semilla generadora de los pesos iniciales el valor de set.seed(1234567). modelo de Red Neuronal Artificial con hidden=1 y hidden= 3.

Comenzaremos por entrenar la red feedforward multicapa más simple, con solo un único nodo oculto:

```
library(neuralnet)
set.seed(1234567)
bc2_model <- neuralnet(Malignant + Benign ~ Cl.thickness + Cell.size + Cell.shape +
                         Marg.adhesion + Epith.c.size + Bare.nuclei + Bl.cromatin +
                         Normal.nucleoli + Mitoses, data=bc2train)
```

Representamos gráficamente el modelo

```
plot(bc2_model)
```

Es un modelo con un único nodo de entrada para cada variable, con un solo nodo oculto, y con dos nodos de salida que predicen la malignidad y la benignidad. También se muestran los pesos para cada una de las conexiones, al igual que los términos de sesgo (en azul), que son constantes que permiten que el valor en los nodos indicados se mueva hacia arriba o hacia abajo, muy parecido a la intersección en una ecuación lineal.

El valor entre cada nodo de entrada y el nodo oculto es similar a los coeficientes de regresión.

R informa del número de pasos de entrenamiento y del SSE (Sum of Squared Errors), que es la suma de los cuadrados predichos menos los valores reales. Un SSE más bajo implica mejor rendimiento predictivo. Esto es útil para estimar el rendimiento del modelo en los datos de entrenamiento, pero nos dice poco sobre cómo funcionará con datos no vistos.

Step4. Evaluando el modelo

El diagrama de topología de la red nos muestra la caja negra del ANN, pero no proporciona mucha información sobre qué tan bien se ajusta el modelo a los datos futuros. Para generar predicciones en el conjunto de datos de prueba, podemos usar compute () de la siguiente manera:

```
model_results <- compute(bc2_model, bc2test[1:9])
```

La función compute () funciona un poco diferente a las funciones predict (). Devuelve una lista con dos componentes: *neurons*, que almacena las neuronas para cada capa en la red, y *net.result*, que almacena la predicción valores.

```
predicted_bc2<- model_results$net.result
```

Debido a que este es un problema de predicción numérica más que un problema de clasificación, no se puede usar una matriz de confusión para examinar la precisión del modelo. En cambio, debemos medir la correlación entre la malignidad/benignidad prevista y el valor real. Esta proporciona información sobre la fuerza de la asociación lineal entre las dos variables.

```
cor(predicted_bc2, bc2test$Benign)
```

```
##            [,1]
## [1,] -0.9235462
## [2,]  0.9235462
```

```
cor(predicted_bc2, bc2test$Malignant)
```

```
##            [,1]
## [1,]  0.9235462
## [2,] -0.9235462
```

Las correlaciones cercanas a 1 indican fuertes relaciones lineales entre dos variables.

Se obtienen valores bastante altos en este caso, pero podemos probar de mejorar el modelo.

Step5. Mejorando el modelo.

Redes con topologías más complejas son capaces de aprender conceptos más difíciles. Veamos qué sucede cuando aumentamos el número de nodos ocultos a 3. Usamos la función neuralnet () como antes, pero agregamos el parámetro hidden = 3:

```
set.seed(1234567)
bc2_model3 <- neuralnet(Malignant + Benign ~ Cl.thickness + Cell.size +
                          Cell.shape + Marg.adhesion + Epith.c.size +
                          Bare.nuclei + Bl.cromatin + Normal.nucleoli +
                          Mitoses, data=bc2train, hidden = 3)
```

```
plot(bc2_model3)
```

Se observa que el error se ha reducido. Pero, sin embargo, la correlación ha disminuido. Esto seguramente ocurre debido al aumento del sesgo. Cuando las variables predictoras están correlacionadas, se produce sesgo de diferente signo en los coeficientes de regresión del resto de predictores. En general, para modelos uniecuacionales con dos o más variables predictoras se puede demostrar analíticamente que los errores de medida en los predictores producen sesgo en las estimaciones de las pendientes y del punto de corte, y que este sesgo se extiende a todos los valores de beta, aun cuando alguna de las variables predictoras esté libre de error.

```
model_results3 <- compute(bc2_model3, bc2test[1:9])
predicted_bc23<- model_results3$net.result
cor(predicted_bc23, bc2test$Benign)
```

```
##            [,1]
## [1,] -0.9100265
## [2,]  0.9088408
```

```
cor(predicted_bc23, bc2test$Malignant)
```

```
##            [,1]
## [1,]  0.9100265
## [2,] -0.9088408
```

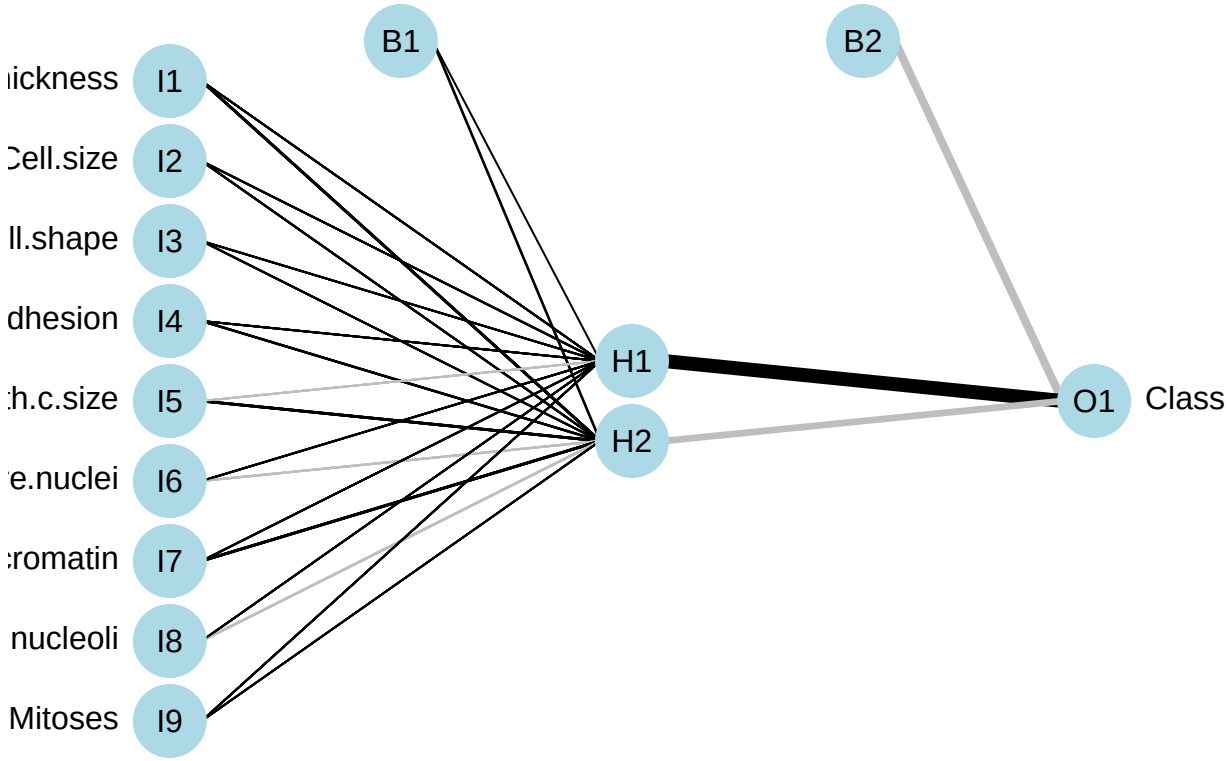
4.Paquete caret: modelo nnet

Analizamos el mismo dataset pero utilizando nnet(). Usar el modelo nnet admite variables categóricas (factor) para predecir, por tanto no hace falta hacer la transformación binaria.

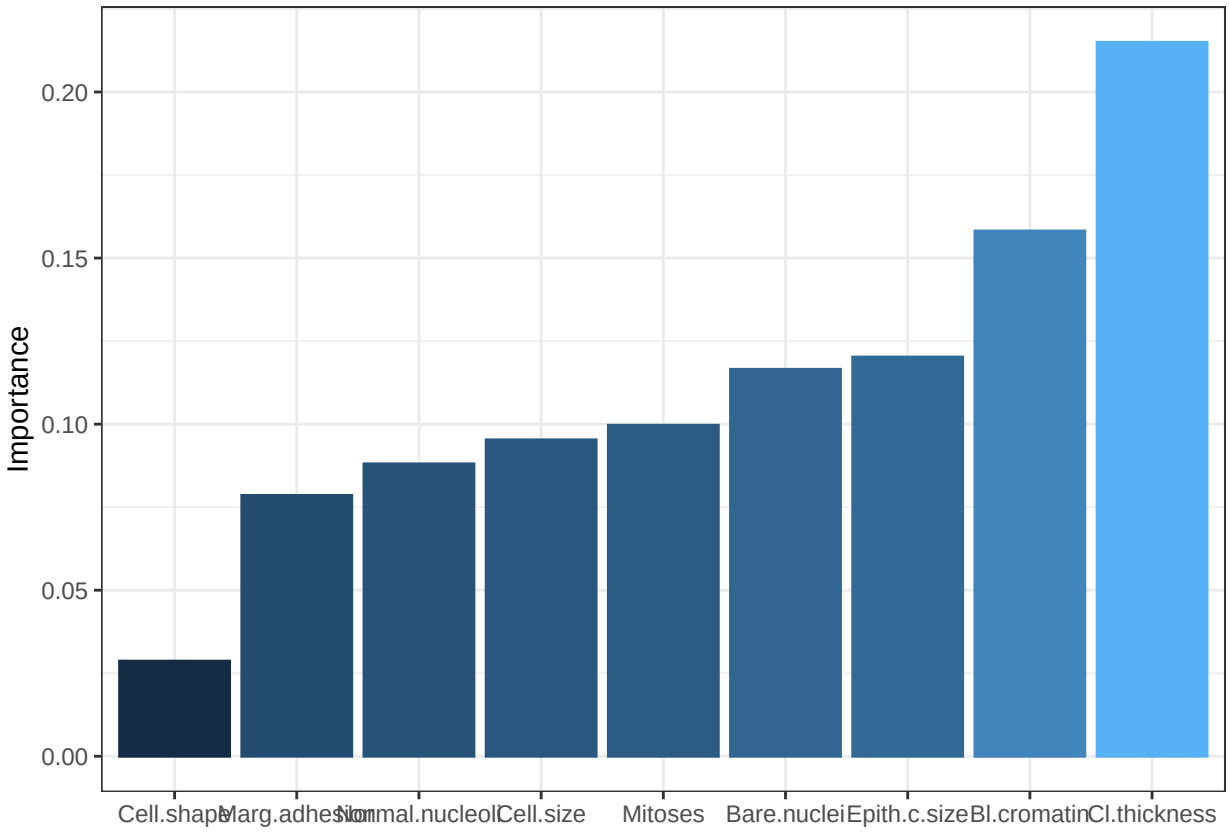
```
# creamos la red neural
library(caret)
library(nnet)
library(NeuralNetTools)
library(klaR)
BCdata<- BreastCancer2[,1:10]
set.seed(12345)
train<-sample(1:nrow(BCdata),round(2*nrow(BCdata)/3,0))
data_train <- BCdata[train,]
data_test <- BCdata[-train,]
mod <- nnet(Class ~ Cl.thickness + Cell.size +Cell.shape + Marg.adhesion +
              Epith.c.size +Bare.nuclei + Bl.cromatin + Normal.nucleoli +
              Mitoses, data=data_train, size = 2)
```

```
## # weights:  23
## initial  value 305.563070
## iter  10 value 83.758376
## iter  20 value 35.800202
## iter  30 value 29.141114
## iter  40 value 27.987754
## iter  50 value 25.027758
## iter  60 value 24.422106
## iter  70 value 24.330876
## iter  80 value 24.256169
## iter  90 value 24.052455
## iter 100 value 23.995344
## final  value 23.995344
## stopped after 100 iterations
```

```
# plot
par(mar = numeric(4))
plotnet(mod)
```



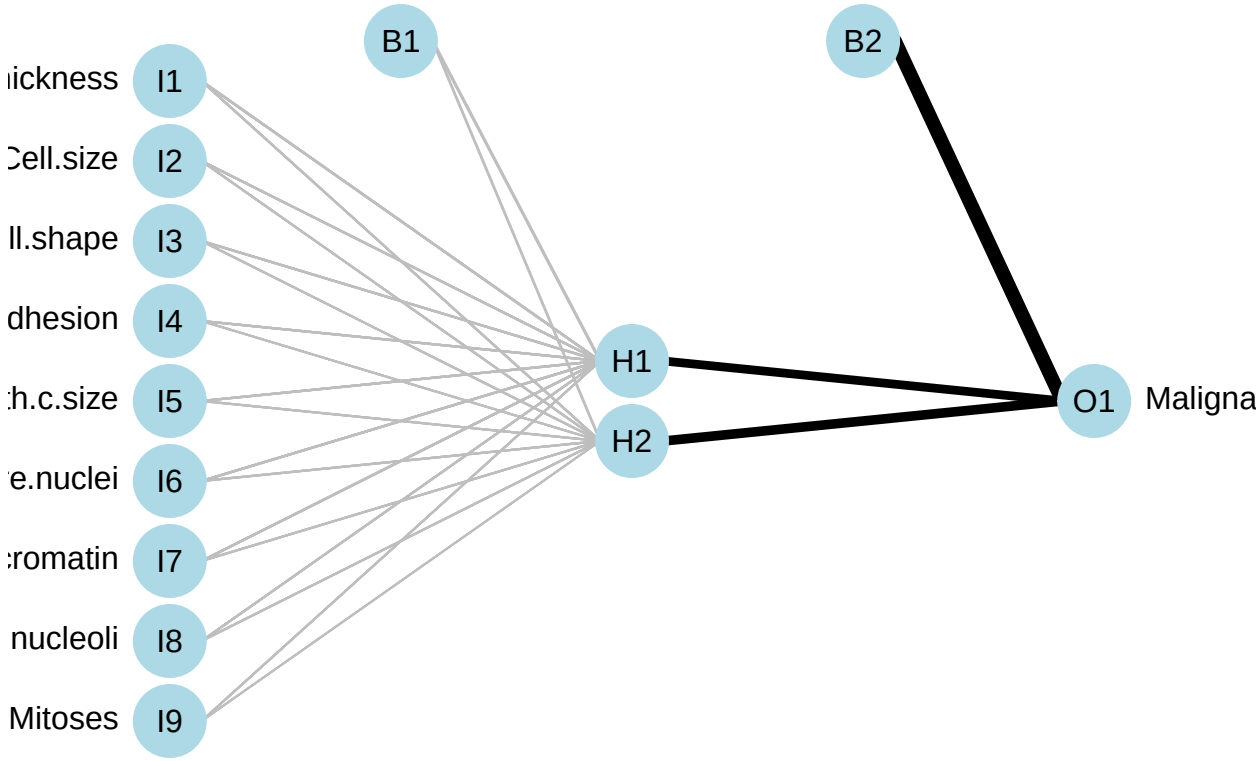

```
# importancia de cada variable
garson(mod)
```



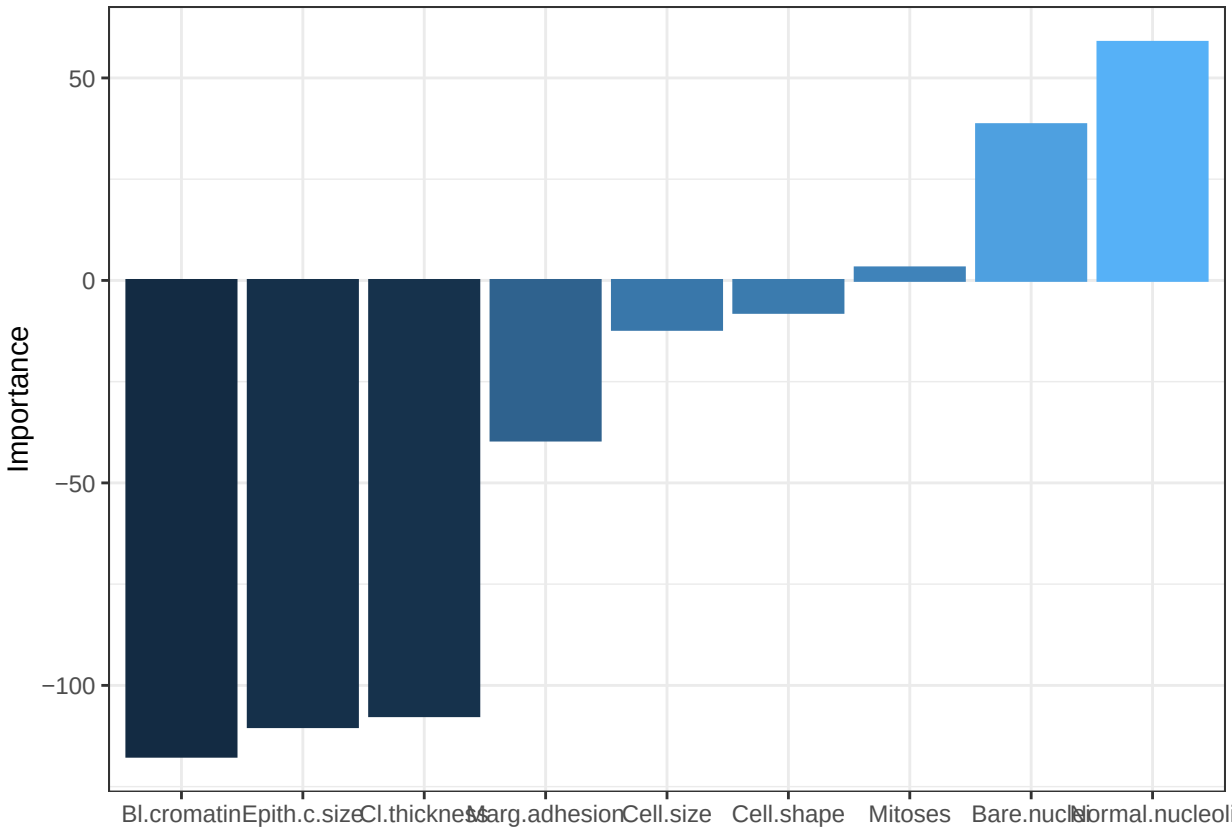

```
#también podemos plantear el modelo con la dos variables binarias:
mod2<- nnet(Malignant + Benign ~ Cl.thickness + Cell.size +
                          Cell.shape + Marg.adhesion + Epith.c.size +
                          Bare.nuclei + Bl.cromatin + Normal.nucleoli +
                          Mitoses, data=bc2train, size=2)
```

```
## # weights:  23
## initial  value 102.369906
## final  value 0.000000
## converged
```

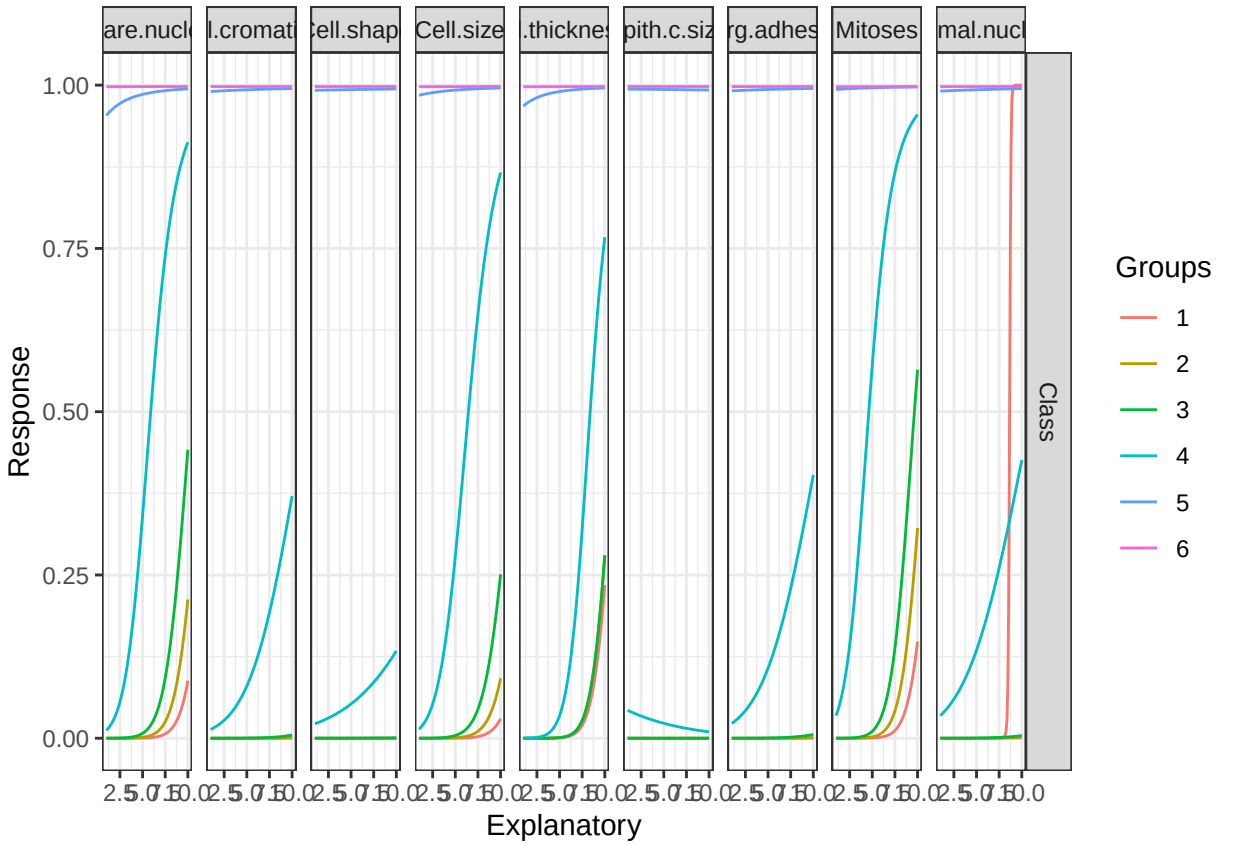
```
# plot
par(mar = numeric(4))
plotnet(mod2)
```




```
#importancia de cada variable con la función olden()
olden(mod)
```




```
#analisis de sensibilidad
lekprofile(mod)
```




```
# hacemos predicciones
x_test <- data_test[,1:9]
y_test <- data_test[,10]

predictions <- predict(mod, x_test)
```

Evaluamos el modelo:

```
confusionMatrix(predictions, y_test, positive="malignant")
```

Bootstrap

Implica tomar muestras aleatorias del conjunto de datos (con re-selección) con las cuales se evalua el modelo. En conjunto, los resultados proporcionan una indicación de la varianza de los modelos. Por lo general, se realiza una gran cantidad de iteraciones de remuestreo (miles o miles de veces).

```
train_control <- trainControl(method="boot", number=25)
# entrenamos el modelo
model <- train(Class~., data=BCdata, trControl=train_control, method="nb")
print(model)
```

k-fold Cross Validation

El método de validación cruzada de k-veces implica dividir el conjunto de datos en k-subconjuntos. Para cada vez un subconjunto se mantiene mientras el modelo se entrena en todos los demás subconjuntos.

```
in_train <- createDataPartition(BCdata$Class, p = 0.67, list = FALSE)
BCdata_train <- BCdata[in_train, ]
BCdata_test <- BCdata[-in_train, ]

folds <- createFolds(BCdata$Class, k = 5)
BC_test <- BCdata[folds$Fold1, ]
BC_train <- BCdata[-folds$Fold1, ]

library(C50)
library(irr)
```

```
## Loading required package: lpSolve
```

```
set.seed(123)
cv_results <- lapply(folds, function(x) {
BCdata_train <- BCdata[-x, ]
BCdata_test <- BCdata[x, ]
BCdata_model <- C5.0(Class ~ ., data = BCdata_train)
BCdata_pred <- predict(BCdata_model, BCdata_test)
BCdata_actual <- BCdata_test$Class
kappa <- kappa2(data.frame(BCdata_actual, BCdata_pred))$value
return(kappa)
})
# el estadístico kappa se registra en una lista dentro de cv_results
str(cv_results)
```

```
## List of 5
##  $ Fold1: num 0.873
##  $ Fold2: num 0.838
##  $ Fold3: num 0.87
##  $ Fold4: num 0.887
##  $ Fold5: num 0.935
```

```
mean(unlist(cv_results))
```

```
## [1] 0.8806012
```

5. Referencias

https://github.com/fawda123/NeuralNetTools

Lantz, Brett. Machine Learning with R. Packt Publishing, 2015. 452 p. ISBN 9781784394523

https://cran.r-project.org/web/packages/NeuralNetTools/NeuralNetTools.pdf

NeuralNetTools: Visualization and Analysis Tools for Neural Networks Marcus W. Beck https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6262849/

https://www.machinelearningplus.com/machine-learning/caret-package

https://cran.r-project.org/web/packages/caret/vignettes/caret.html

Predictive Modeling with R and the caret Package: https://www.r-project.org/nosvn/conferences/useR-2013/Tutorials/kuhn/user_caret_2up.pdf

http://www.di.fc.ul.pt/~jpn/r/neuralnets/neuralnets.html