Árboles de decisión, Random Forest, Gradient Boosting y C5.0

Los árboles de decisión son modelos predictivos formados por reglas binarias (si/no) con las que se consigue repartir las observaciones en función de sus atributos y predecir así el valor de la variable respuesta. Estos son similares a diagramas de flujo, en los que llegamos a puntos en los que se toman decisiones de acuerdo a una regla.

Muchos métodos predictivos generan modelos globales en los que una única ecuación se aplica a todo el espacio muestral. Cuando el caso de uso implica múltiples predictores, que interaccionan entre ellos de forma compleja y no lineal, es muy difícil encontrar un único modelo global que sea capaz de reflejar la relación entre las variables. Los métodos estadísticos y de aprendizaje computacional basados en árboles engloban a un conjunto de técnicas supervisadas no paramétricas que consiguen segmentar el espacio de los predictores en regiones simples, dentro de las cuales es más sencillo manejar las interacciones. Es esta característica la que les proporciona gran parte de su potencial.

En el campo del aprendizaje computacional supervisado, hay distintas maneras de obtener árboles de decisión, una de las más conocidas es CART: Classification And Regression Trees. Tenemos una variable objetivo (dependiente) y nuestra meta es obtener una función que nos permita predecir, a partir de variables predictoras (independientes), el valor de la variable objetivo para casos desconocidos. Se usa clasificación cuando la variable objetivo es discreta, mientras que se usa regresión cuando es continua. La implementación particular de CART que usaremos es conocida como Recursive Partitioning and Regression Trees (RPART).

Los modelos Random Forest están formados por un conjunto de árboles de decisión individuales, cada uno entrenado con una muestra ligeramente distinta de los datos de entrenamiento generada mediante bootstrapping. La predicción de una nueva observación se obtiene agregando las predicciones de todos los árboles individuales que forman el modelo.

Los modelos Gradient Boosting están formados por un conjunto de árboles de decisión individuales, entrenados de forma secuencial, de forma que cada nuevo árbol trata de mejorar los errores de los árboles anteriores. La predicción de una nueva observación se obtiene agregando las predicciones de todos los árboles individuales que forman el modelo.

Los métodos basados en árboles se han convertido en uno de los referentes dentro del ámbito predictivo debido a los buenos resultados que generan en problemas muy diversos. A lo largo de este documento se explora la forma en que se construyen y predicen los árboles de decisión (clasificación y regresión), elementos fundamentales de modelos predictivos más complejos como Random Forest y Gradient Boosting.

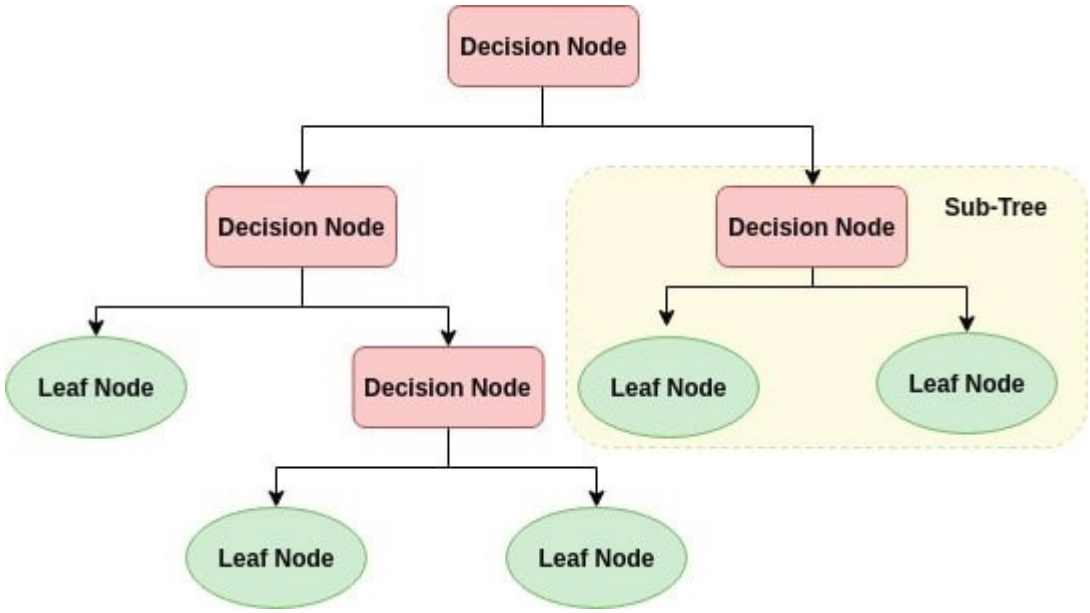
Funcionamiento

El nodo superior en un árbol de decisión se conoce como el nodo raíz y aprende a particionar en función del valor del atributo.

Esta estructura tipo diagrama de flujo lo ayuda a tomar decisiones. Es una visualización como un diagrama de flujo que imita fácilmente el pensamiento a nivel humano. Es por eso que los árboles de decisión son fáciles de entender e interpretar.

Los árboles de decisión clasifican los ejemplos desde la raíz hasta algún nodo hoja este enfoque se llama Enfoque de arriba hacia abajo (Top-Down).

Cada nodo en el árbol actúa como un caso de prueba para algún atributo, y cada borde que desciende de ese nodo corresponde a una de las posibles respuestas al caso de prueba. Este proceso es recursivo y se repite para cada subárbol en los nuevos nodos.

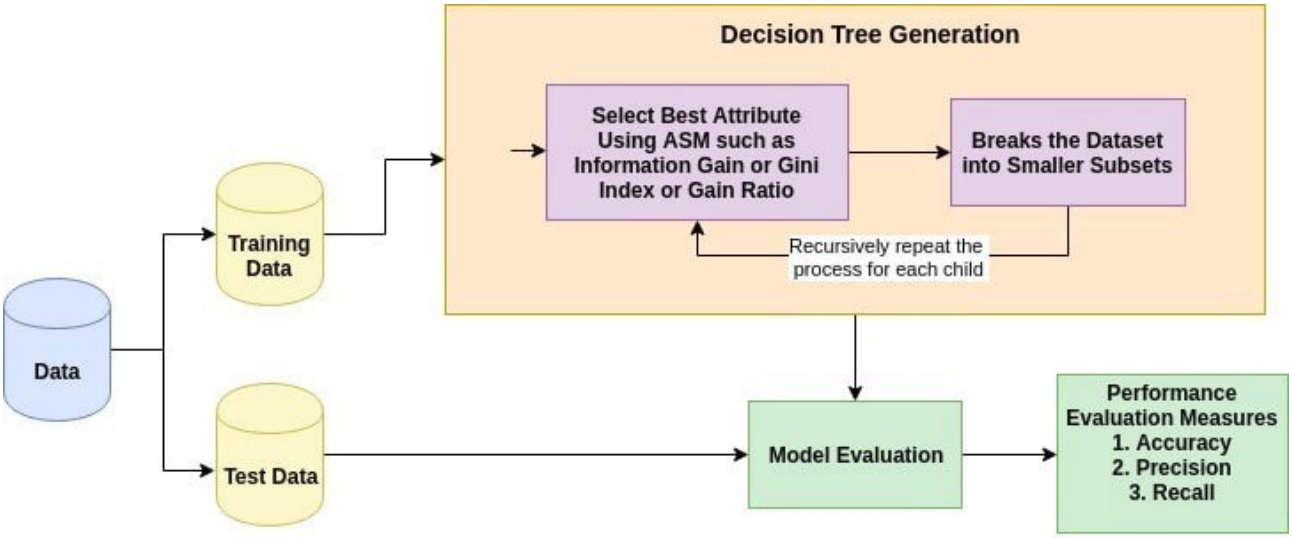


Algoritmo de árbol de decisión

La idea básica detrás de cualquier algoritmo de árbol de decisión es el siguiente:

1. Seleccionar el mejor atributo utilizando Medidas de selección de atributos (ASM) para dividir los atributos.
2. Hacer que ese atributo sea un nodo de decisión y dividir el conjunto de datos en subconjuntos más pequeños.
3. Realizar recursivamente este proceso para cada subconjunto resultante hasta que una de las condiciones coincida:

- Todas las tuplas pertenecen al mismo valor de atributo.
- No quedan más atributos.
- No hay más instancias.



Medidas de selección de atributos

La medida de selección de atributos (ASM) es una heurística para seleccionar el criterio de división que divide los datos de la mejor manera posible. También se conoce como reglas de división porque nos ayuda a determinar puntos de interrupción para tuplas en un nodo dado.

ASM proporciona un rango para cada atributo (variable) al explicar el conjunto de datos dado. El atributo de mejor puntuación se seleccionará como un atributo de división.

Ganancia de información. La ganancia de información es una propiedad estadística que mide qué tan bien un atributo dado separa los ejemplos de entrenamiento de acuerdo con sus clases objetivo.

Entropía: Mide la impureza del conjunto de entrada (entrenamiento). En teoría de la información, se refiere a la impureza en un grupo de ejemplos. La ganancia de información es una disminución de la entropía.

Gini: Enuncia que si seleccionamos dos elementos de una población al azar, entonces deben ser de la misma clase y la probabilidad de esto es 1 si la población es pura.

Ventajas de arboles de decisión

- Los árboles son fáciles de interpretar aun cuando las relaciones entre predictores son complejas.
- Los modelos basados en un solo árbol (no es el caso de Random Forest y Boosting) se pueden representar gráficamente aun cuando el número de predictores es mayor de 3.
- Al tratarse de métodos no paramétricos, no es necesario que se cumpla ningún tipo de distribución específica.

- No se ven muy influenciados por outliers.
- Son muy útiles en la exploración de datos, permiten identificar de forma rápida y eficiente las variables (predictores) más importantes.
- Son capaces de seleccionar predictores de forma automática.
- Pueden aplicarse a problemas de regresión y clasificación.

Desventajas de arboles de decisión

- La capacidad predictiva de los modelos basados en un único árbol es bastante inferior a la conseguida con otros modelos. Esto es debido a su tendencia al overfitting y alta varianza.
- Son sensibles a datos de entrenamiento desbalanceados.
- Cuando tratan con predictores continuos, pierden parte de su información al categorizarlos en el momento de la división de los nodos.
- No son capaces de extrapolar fuera del rango de los predictores observado en los datos de entrenamiento.
- La creación de las ramificaciones de los árboles se consigue mediante el algoritmo de recursive binary splitting. Este algoritmo identifica y evalúa las posibles divisiones de cada predictor acorde a una determinada medida (RSS, Gini, entropía…). Los predictores continuos tienen mayor probabilidad de contener, solo por azar, algún punto de corte óptimo, por lo que suelen verse favorecidos en la creación de los árboles.

Ejemplos en R:

En esta practica trabajaremos sobre el dataset Dry Bean Dataset que se encuentra en el repositorio UCI[1] para clasificar tipos de frijoles. Utilizaremos el paquete rpart de R-cran que incorpora el uso de arboles de decisión.
Características del dataset:
Consta de siete tipos diferentes de frijoles secos, tomando en cuenta las características como forma, forma, tipo y estructura por la situación del mercado. Las imágenes de los frijoles obtenidas, 13611, por el sistema de visión por computadora fueron sometidas a etapas de segmentación y extracción de 16 características: 12 de dimensiones y 4 de formas de los granos. Las clases de frijoles son: Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira

Para analizar y comprender el funcionamiento de los métodos de clasificación reduciremos el dataset original a un subconjunto más pequeño que consta dos clases (Bombay y Sira) y 3 atributos (Área, Perímetro y Solidez).

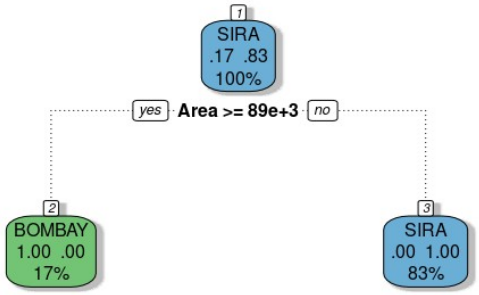
```
library(rpart)
library(caret)

miniDryBean <- read.csv("MiniDryBean.csv")
miniDryBean[,4] <- as.factor(miniDryBean[,4])

indexData <- createFolds(t(miniDryBean[, "TipoFrijol"]), k = 5)

miniDryBeanTest <- miniDryBean[indexData[[3]], ]
```

1 https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset

```
miniDryBeanTrain <- miniDryBean[setdiff(seq(1:dim(miniDryBean)[1]),
indexData[[3]]), ]
```

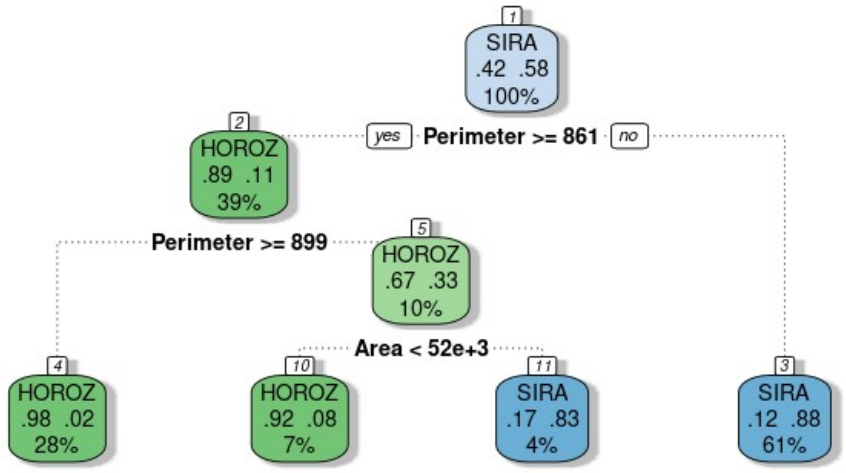
Entrenamiento

```
fit3 <- rpart(TipoFrijol~Area+Perimeter+Solidity, data = miniDryBeanTrain,
parms = list(split = 'information'))

summary(fit3)
Call:
rpart(formula = TipoFrijol ~ Area + Perimeter + Solidity, data =
miniDryBeanTrain,
    parms = list(split = "information"))
  n= 2526

    CP nsplit rel error xerror       xstd
1 1.00      0         1      1 0.04468178
2 0.01      1         0      0 0.00000000

Variable importance
     Area Perimeter  Solidity
       49        49         2

Node number 1: 2526 observations,    complexity param=1
  predicted class=SIRA    expected loss=0.165479  P(node) =1
    class counts:   418  2108
   probabilities: 0.165 0.835
  left son=2 (418 obs) right son=3 (2108 obs)
  Primary splits:
      Area      < 88808     to the right, improve=1133.27600, (0 missing)
      Perimeter < 1125.104  to the right, improve=1133.27600, (0 missing)
      Solidity  < 0.9802664 to the left,  improve=  31.03195, (0 missing)
  Surrogate splits:
      Perimeter < 1125.104  to the right, agree=1.00, adj=1.000, (0 split)
      Solidity  < 0.9746305 to the left,  agree=0.84, adj=0.033, (0 split)

Node number 2: 418 observations
  predicted class=BOMBAY  expected loss=0  P(node) =0.165479
    class counts:   418     0
   probabilities: 1.000 0.000

Node number 3: 2108 observations
  predicted class=SIRA    expected loss=0  P(node) =0.834521
    class counts:     0  2108
   probabilities: 0.000 1.000

library(rattle)
library(rpart.plot)

# plot mi modelo reducido
fancyRpartPlot(fit3, caption = NULL)
```



Ahora vamos a trabajar con otro subconjunto que constara con dos clases (Horoz y Sira) y los mismo 3 atributos (Área, Perímetro y Solidez).

```
miniDryBean2 <- read.csv("MiniDryBean2.csv")
miniDryBean2[,4] <- as.factor(miniDryBean2[,4])

indexData <- createFolds(t(miniDryBean2[, "TipoFrijol"]), k = 5)

miniDryBeanTest2 <- miniDryBean2[indexData[[3]], ]

miniDryBeanTrain2 <- miniDryBean2[setdiff(seq(1:dim(miniDryBean)[1]),
indexData[[3]]), ]

fit4 <- rpart(TipoFrijol~Area+Perimeter+Solidity, data = miniDryBeanTrain2,
parms = list(split = 'information'))

fancyRpartPlot(fit4, caption = NULL)
```



Ejercicio:

Analizar los modelos generados considerando:

1. Otros 3 atributos distintos del dataset original
2. No considerar el atributo Área en el modelo generado con el miniDryBeanTrain

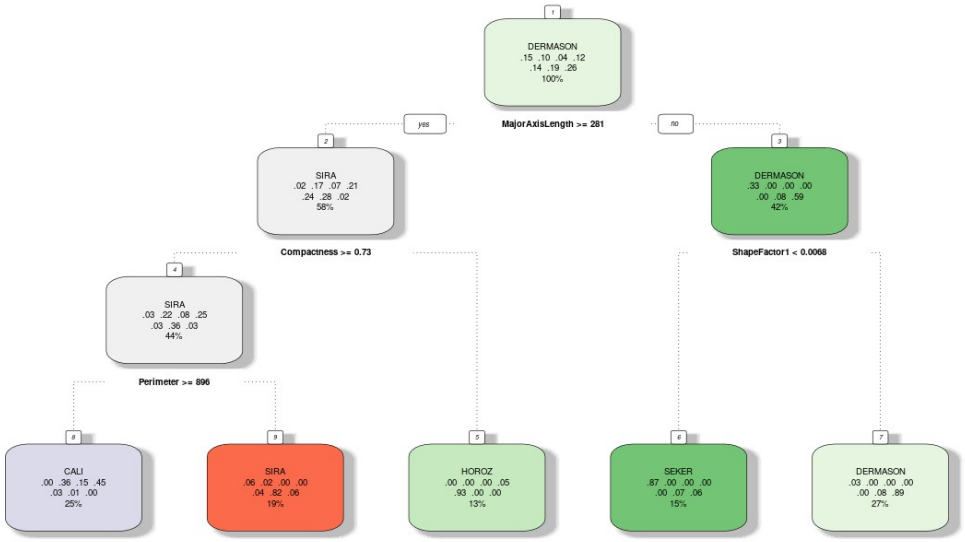
Ahora vamos analizar la performance de nuestro modelo sobre el conjunto de test

Predicción

```
predictDryBean <- predict(fit3, miniDryBeanTest[,-4], type = 'class')

table_mat <- table(t(miniDryBeanTest[,4]), predictDryBean)

accuracy_Test <- sum(diag(table_mat[,c(3,6)])) / sum(table_mat[,c(3,6)])
```

Por último los arboles generados pueden ser podados mediante la función prune.

```
fit1Pruned <- prune(fit1, cp=0.1)

fancyRpartPlot(fit1Pruned, caption = NULL)
```



Ejercicios:

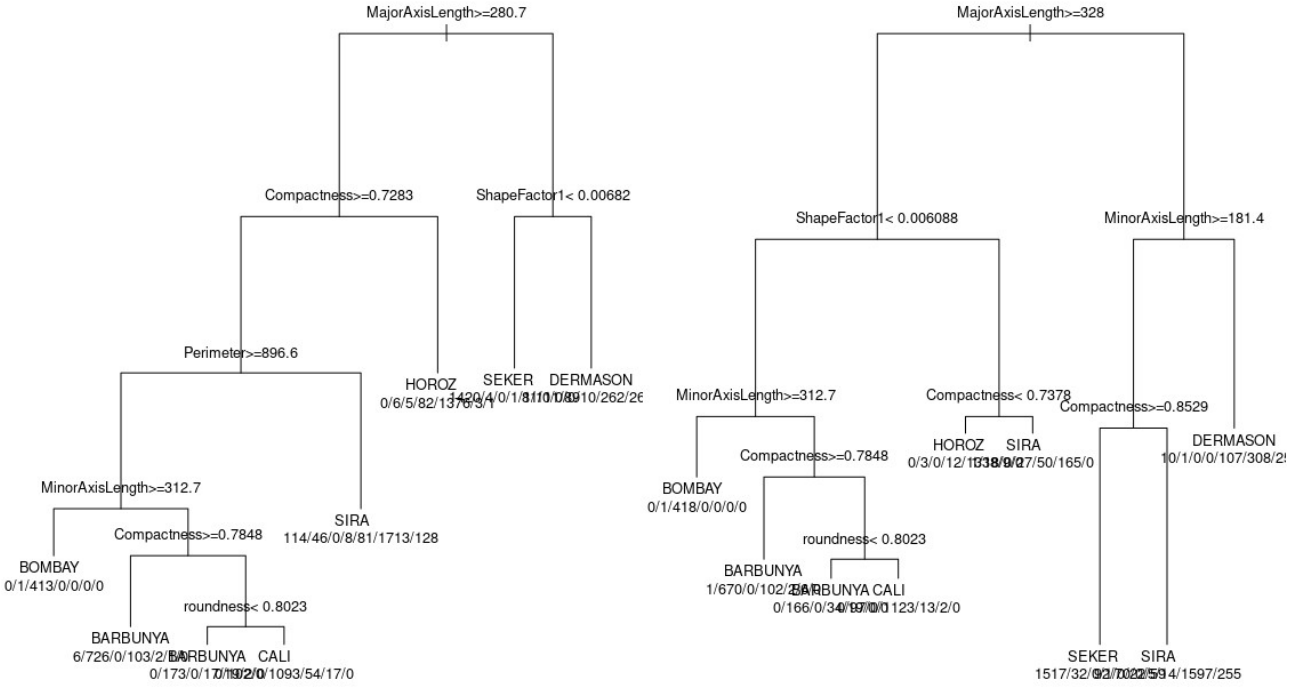
1. Analizar la salida de la siguiente instrucción:
   `predict(fit3, miniDryBeanTest[,-4], type = 'prob')`

2. Analizar que sucede si cambiamos los valores por defecto de los parámetros a maxdepth = 1, minsplit = 2 y minbucket = 1 en la generación del modelo.

Ahora trabajaremos con el dataset completo que consta de 16 atributos para clasificar las 7 clases de frijoles

```
library(readxl)
```

```
Dry_Bean_Dataset <- read_excel("Dry_Bean_Dataset.xlsx", sheet =
"Dry_Beans_Dataset")

Dry_Bean_Dataset[, 17] <- apply(Dry_Bean_Dataset[, 17], MARGIN = 1,
as.factor)
colnames(Dry_Bean_Dataset)[17]<-"TipoFrijol"

indexData <- createFolds(t(Dry_Bean_Dataset[, "TipoFrijol"]), k = 5)

dryBeanTest <- Dry_Bean_Dataset[indexData[[3]], ]

dryBeanTrain <- Dry_Bean_Dataset[setdiff(seq(1:dim(Dry_Bean_Dataset)[1]),
indexData[[3]]), ]

fit1 <- rpart(TipoFrijol~., data=dryBeanTrain, parms=list(split = 'gini'))

fit2 <- rpart(TipoFrijol~., data=dryBeanTrain, parms=list(split = '
information'))

par(mfrow = c(1,2), mar = rep(0.1, 4))
plot(fit1, margin = 0.05); text(fit1, use.n = TRUE, cex = 0.8)
plot(fit2, margin = 0.05); text(fit2, use.n = TRUE, cex = 0.8)
```



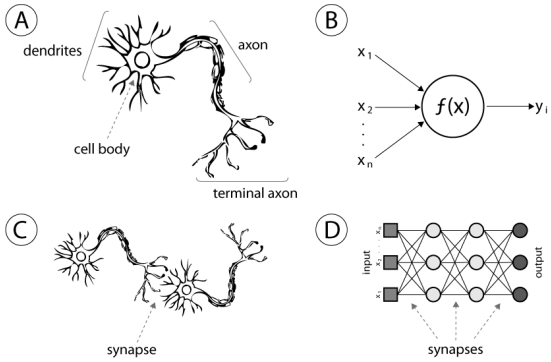
Redes Neuronales Artificiales

Paradigma de aprendizaje automatizado inspirado en sistemas nerviosos biológicos, constituidos por una serie de nodos (neuronas) y una serie de conexiones entre los nodos (sinapsis).

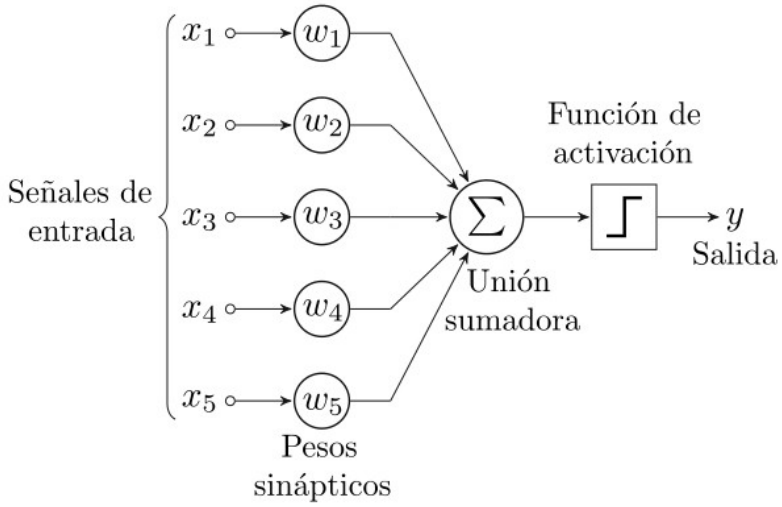
La topología básica consta de:

- Nodos de entrada al sistema, generalmente tantos nodos como variables.
- Una serie de capas ocultas intermedias con un número variable de nodos.
- Nodos de salida, uno por cada respuesta.

Cada nodo de entrada tiene asociado un peso W_i y en cada nodo se aplica una función de activación (opcional) y una de propagación con la suma de las entradas ponderadas por sus pesos.



Aunque hay muchas topologías, la más común es la conocida como perceptrón



Componentes:

- N entradas, x_1,..., x_n
- Cada entrada x tiene un peso asociado w, w_1,..., w_n
- Un nodo de entrada extra llamada bias (intercepto)
- Suma de las entradas ponderada por sus pesos: $y= \sum x * w$
- Función de activación p.e.: $f_a(x) = 1$ si $y>0$, $f_a(x) = -1$ si $x \leq 0$

Ejemplos en R:

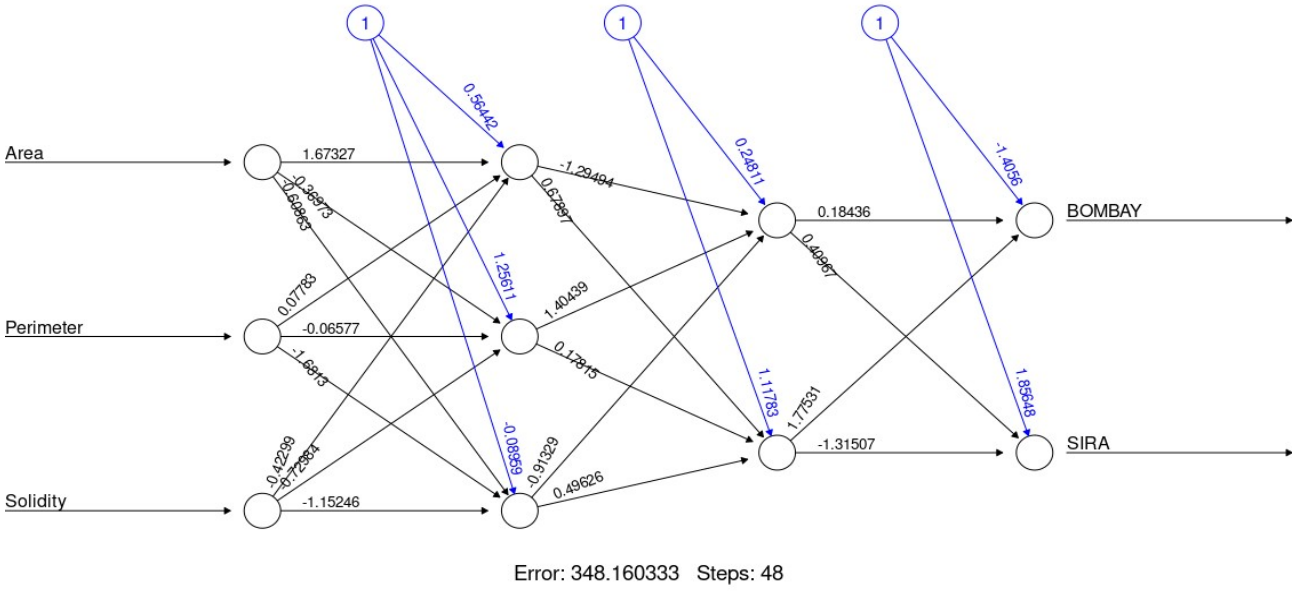
Lo primero que debemos hacer es binarizar las clases objetivo para esto se generan nuevas columnas con valores lógicos (Verdadero o Falso) para pertenencia de cada observación a cada clase.

```
library(neuralnet)

miniDryBeanTrain <- cbind(miniDryBeanTrain, miniDryBeanTrain[,4]=="BOMBAY")
miniDryBeanTrain <- cbind(miniDryBeanTrain, miniDryBeanTrain[,4]=="SIRA")
colnames(miniDryBeanTrain)[5:6] <- c("BOMBAY", "SIRA")

head(miniDryBeanTrain)
```

```
    Area Perimeter  Solidity TipoFrijol BOMBAY  SIRA
1 114004  1279.356 0.9887769     BOMBAY   TRUE FALSE
2 126503  1326.959 0.9866090     BOMBAY   TRUE FALSE
3 128118  1360.135 0.9910578     BOMBAY   TRUE FALSE
4 129409  1348.888 0.9902133     BOMBAY   TRUE FALSE
5 131249  1374.898 0.9880901     BOMBAY   TRUE FALSE
6 131488  1368.233 0.9893382     BOMBAY   TRUE FALSE
```

Entrenamiento

```
dryBeanANN <- neuralnet(BOMBAY+SIRA~Area+Perimeter+Solidity,
data=miniDryBeanTrain, hidden = c(3, 2))

plot(dryBeanANN, col.intercept="blue" , rep="best")
```



Predicción

Con los datos de test, miniDryBeanTest,vamos a clasificar estos datos utilizando nuestro modelo generado a través de la función *compute*

```
predictDryBeanANN <- compute(dryBeanANN, miniDryBeanTest[1:3])
```

Sin embargo, la visualización del resultado de nuestra predicción, clasificación, no es tan "amigable" como en los otros algoritmos vistos. Para verlos en forma de tabla y poder comparar que tan buena es nuestra perfomance usaremos el siguiente código.

```
arrayDryBeanANN <- 0
for ( i in 1:dim(predictDryBeanANN$net.result)[1]){
  arrayDryBeanANN[i] <- which.max(predictDryBeanANN$net.result[i, ])
}
arrayDryBeanANN[ arrayDryBeanANN == 1 ] <- "BOMBAY"
arrayDryBeanANN[ arrayDryBeanANN == 2 ] <- "SIRA"

miniDryBeanTest[,4] <- apply(miniDryBeanTest[,4], MARGIN = 1, as.character)
table(Predicción = arrayDryBeanANN, Realidad = t(miniDryBeanTest[,4]))

          Realidad
Predicción BOMBAY SIRA
      SIRA    105  527

confusionMatrix(as.factor(arrayDryBeanANN),
as.factor(apply(miniDryBeanTest[, 4], MARGIN = 1, as.character)))

Confusion Matrix and Statistics

          Reference
Prediction BOMBAY SIRA
    BOMBAY      0    0
    SIRA      105  527

               Accuracy : 0.8339
                 95% CI : (0.8025, 0.8621)
    No Information Rate : 0.8339
    P-Value [Acc > NIR] : 0.526

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.0000
            Specificity : 1.0000
         Pos Pred Value :    NaN
         Neg Pred Value : 0.8339
             Prevalence : 0.1661
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : BOMBAY
```

Ejercicio

1. Analizar la salida de la siguiente instrucción:
   `predictDryBeanANN <- predict(dryBeanANN, miniDryBeanTest[1:3])`

2. Analizar que sucede si cambiamos los valores por defecto de los parámetros hidden, learningrate, threshold, stepmax en la generación del modelo.
3. Realizar la clasificación sobre las 7 clases del dataset y evaluar su perfomance.

Bibliografía

An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics)

Tutorial online: https://www.datacamp.com/community/tutorials/machine-learning-in-r

Machine Learning Essentials de Alboukadel Kassambara