Fundamentos de la Ciencia de Datos
Práctica 3

Fernández Díaz, Daniel
Cano Díaz, Francisco
Fernández Hernández, Alberto

5 de noviembre del 2019

Índice

1. Apartado 1

La primera parte de la práctica consistirá en la realización de dos ejercicios en clase con ayuda del profesor en el que se va a realizar un **análisis de clasificación** de Datos con `R` aplicando todos los conceptos vistos en el tema.

1.1. Apartado 1.1: Árboles de decisión de Hunt

El primer ejercicio consitirá en el desarrollo de un árbol de decisión mediante el **algoritmo de Hunt**, utilizando la siguiente muestra de datos con calificiones en **teoría**, **laboratorio** y **prácticas**, almacenadas el fichero *calificaciones.txt*:

Teoria, Laboratorio, Prácticas, Calificación Global

1. {A, A, B, Ap}
2. {A, B, D, Ss}
3. {D, D, C, Ss}
4. {D, D, A, Ss}
5. {B, C, B, Ss}
6. {C, B, B, Ap}
7. {B, B, A, Ap}
8. {C, D, C, Ss}
9. {B, A, C, Ss}

En primer lugar, leemos el fichero *calificaciones.txt* mediante el comando ***read.table***:

```
> calificaciones <- read.table("calificaciones.txt")
> #Convertimos el dato a formato dataframe
> muestra <- data.frame(calificaciones)
> muestra

  Teoria Lab Prac Calif_glob
1      A   A    B         Ap
2      A   B    D         Ss
3      D   D    C         Ss
4      D   D    A         Ss
5      B   C    B         Ss
6      C   B    B         Ap
7      B   B    A         Ap
8      C   D    C         Ss
9      B   A    C         Ss
```

Una vez obtengamos nuestro *dataframe*, utilizaremos la función ***rpart*** (disponible en la librería *rpart*) para la creación del árbol de decisión. Además, aplicaremos el *Gini* como medida de impureza para el cálculo de la ganancia de información, por defecto en *rpart*:

```
> if(!require(rpart)){
+         install.packages("rpart")
+         library(rpart)
+ }
> clasificacion <- rpart(Calif_glob ~., data = muestra, method = "class", minsplit = 1)
```

Los argumentos de la función ***rpart*** que hemos utilizado son:

- **Formula**: indicamos la fórmula para la clasificación. En nuestro caso, establecemos *Calif_glob* como clasificador, mientras que el resto de columnas serán los valores utilizados para clasificar (indicado con un punto).
- **data**: la muestra a clasificar.
- **method**: en nuestro caso, elegimos el método de **clasificación**.
- **minsplit**: indica el número mínimo de observaciones que deben existir en un nodo.

Una vez elaborado el árbol de decisión, vamos a visualizarlo:

```
> clasificacion

n= 9

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 9 3 Ss (0.3333333 0.6666667)
  2) Lab=A,B 5 2 Ap (0.6000000 0.4000000)
    4) Prac=A,B 3 0 Ap (1.0000000 0.0000000) *
    5) Prac=C,D 2 0 Ss (0.0000000 1.0000000) *
  3) Lab=C,D 4 0 Ss (0.0000000 1.0000000) *
```

Una vez obtenido el árbol nos disponemos a mostrarlo. Para ello utilizaremos el comando ***rpart.plot*** de la librería *rpart.plot*:

```
> if(!require(rpart.plot)){
+         install.packages("rpart.plot")
+         library(rpart.plot)
+ }
```

```
> rpart.plot(clasificacion,type=5)
```

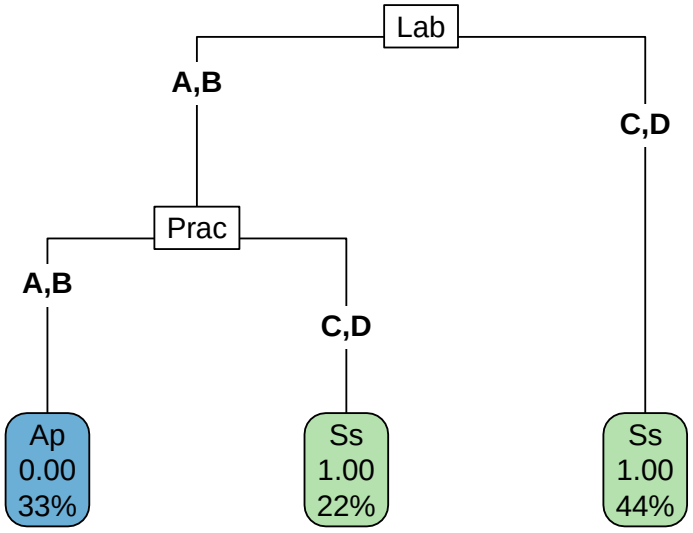


Figura 1: Árbol de decisión para la muestra de Califiaciones

Como podemos ver en el árbol de descisión tenemos un clasificador en el cual partimos de la nota de **Laboratorio**. Si esta nota es una C o D podemos decir que la **Calificación Global** es Suspenso pero si es A o B tendremos que mirar la nota de **Prácticas**, la cual, si es una A o B es Aprobado mientras que si es una C o D es Suspenso.

1.2. Apartado 1.2: Regresión lineal

En este apartado se realizará un **análisis de regresión lineal** utilizando la muestra formada por los radios y densidades de 4 planetas interiores:

Planeta, Radio, Densidad

1. {Mercurio, 2.4, 6.4}
2. {Venus, 6.1, 5.2}
3. {Tierra, 6.4, 5.5}
4. {Marte, 3.4, 3.9}

En primer lugar, realizamos la lectura del fichero ***planetas.txt*** donde se encuentra la muestra:

```
> planetas <- read.table("planetas.txt")
> #Convertimos el dato a dataframe
> muestra_planetas <- data.frame(planetas)
> muestra_planetas

           R   D
Mercurio 2.4 5.4
Venus    6.1 5.2
Tierra   6.4 5.5
Marte    3.4 3.9
```

Una vez tenemos el *dataframe*, mediante la función ***lm*** vamos a crear un **modelo de regresión lineal**, con el que estableceremos una fórmula para el cálculo de la **densidad** en función del **radio** de los planetas (indicado de la siguiente manera: D ~ R):

```
> regresion <- lm(D ~ R, data = muestra_planetas)
> #y = a + bx
> #Coeficientes a b
> #Intercept --> a
> #R --> b
> regresion

Call:
lm(formula = D ~ R, data = muestra_planetas)

Coefficients:
(Intercept)            R
     4.3624       0.1394
```

Como podemos comprobar, obtenemos la siguiente función de regresión: $y = 0.1394x + 4.3624$.

Una vez obtenida la función de regresión, nos disponemos a mostrarla gráficamenente el **modelo de regresión lineal**, es decir, representaremos el **diagrama de dispersión** junto con la **recta de ajuste**. Para ello utilizaremos el comando ***plot*** y ***abline***:

```
>  plot(muestra_planetas$R,muestra_planetas$D,xlab='Radio Ecuatorial',
+       ylab='Densidad', main = "Modelo Regresión Lineal: Planetas Interiores")
>  abline(regresion,col='red')
```

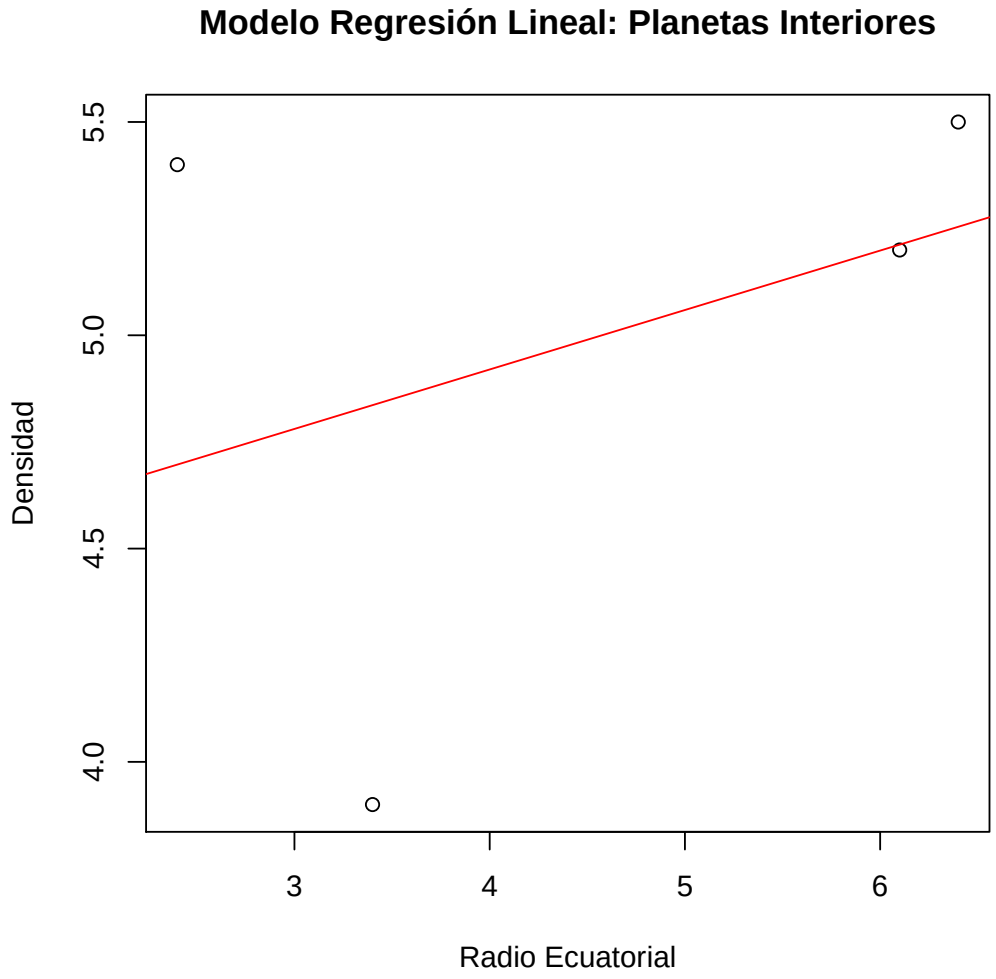


Figura 2: Modelo Regresión Lineal: Planetas Interiores

Además, si ejecutamos el comando ***summary*** podremos ver, entre otros campos, el **error estándar residual** de la función:

```
> summary(regresion)

Call:
lm(formula = D ~ R, data = muestra_planetas)

Residuals:
Mercurio    Venus   Tierra    Marte
 0.70312 -0.01253  0.24566 -0.93624

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.3624     1.2050   3.620   0.0685 .
R             0.1394     0.2466   0.565   0.6289
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.846 on 2 degrees of freedom
Multiple R-squared:  0.1377,        Adjusted R-squared:  -0.2935
F-statistic: 0.3193 on 1 and 2 DF,  p-value: 0.6289
```

Con la representación gráfica y los valores que nos muestra *summary* podemos ver que es un **modelo de regresión lineal mal ajustado** ya que la recta no se ajusta correctamente a los valores y estos a su vez se encuentran bastante lejos de la recta. Esto se debe principalmente a la poca cantidad de datos de los que disponemos.
Si entramos en más detalle en el comando *summary* podemos observar el error estándar residual de cada uno de los puntos al valor correspondiente en la recta (Mercurio: 0.70312; Venus: -0.01253; Tierra: 0.24566; Marte: -0.93624). Todos estos errores son grandes salvo el de Venus, ya que como vemos en la gráfica esta muy cerca de la recta de ajuste. Por último destacar el error estándar residual total que es 0.846, lo que nos confirma el mal ajuste de la recta.

2. Apartado 2

En este apartado realizaremos de nuevo varios **análisis de clasificación** pero en este caso utilizaremos nuevas muestras y herramientas.

2.1. Apartado 2.1: Árboles de decisión de Hunt

Al igual que hicimos en el Apartado 1.1 desarrollaremos el árbol de decisión mediante el **Algoritmo de Hunt**, pero en este caso utilizaremos la siguiente muestra:

TipoCarnet, NúmeroRuedas, NúmeroPasajeros, TipoVehículo

1. {B, 4, 5, Coche}
2. {A, 2, 2, Moto}
3. {N, 2, 1, Bicicleta}
4. {B, 6, 4, Camion}
5. {B, 4, 6, Coche}
6. {B, 4, 4, Coche}
7. {N, 2, 2, Bicicleta}
8. {B, 2, 1, Moto}
9. {B, 6, 2, Camion}
10. {N, 2, 1, Bicicleta}

En esta muestra, tenemos las características de 10 vehículos de cuatro tipos diferentes. A partir del **TipoCarnet**, **NúmeroRuedas** y **NúmeroPasajeros** obtendremos el **TipoVehículo** que será nuestro suceso clasificador.
Lo primero que debemos que hacer, al igual que antes, es leer el fichero ***vehiculos.txt*** que contiene la muestra:

```
> vehiculos <- read.table("vehiculos.txt")
> vehiculos

   TC NR NP        TP
1   B  4  5     Coche
2   A  2  2      Moto
3   N  2  1 Bicicleta
4   B  6  4    Camion
5   B  4  6     Coche
6   B  4  4     Coche
7   N  2  2 Bicicleta
8   B  2  1      Moto
9   B  6  2    Camion
10  N  2  1 Bicicleta
```

Donde TC corresponde a TipoCarnet, NR a NúmeroRuedas, NP a NúmeroPasajeros y TP a TipoVehículo.

Una vez leída la muestra, aplicaremos el **algoritmo de Hunt** para el desarrollo del **árbol de decisión** mediante las siguientes funciones:

```
> ## Arboles de decision: Algoritmo de Hunt ##
>
> # Funcion principal. Llama a las funciones que construyen el arbol y lo muestran.
> arbol <- function(muestra){
+   clasificacion <- arbol.clasificacion(muestra)
+   arbol.mostrar(clasificacion)
+ }
> # Realiza la construccion del arbol de decision mediante el comando rpart utilizando
> # el Gini para calcular la ganancia de informacion.
> # Además, como parametro le podemos pasar la columna que queramos considerar
> # como suceso clasificador (por defecto, sera la ultima).
> arbol.clasificacion <- function(muestra,posClasificador=length(muestra)){
+   clasificacion <- rpart(paste(colnames(muestra)[posClasificador],"~."),data=muestra,
+                          method="class",minsplit=1,parms=list(split="gini"),model=T)
+ }
> # Muestra el arbol de decision mediante el comando rpart.plot.
> arbol.mostrar <- function(clasificacion){
+    rpart.plot(clasificacion,type=5)
+ }
```

```
> ## Ejecucion ##
> arbol(vehiculos)
```

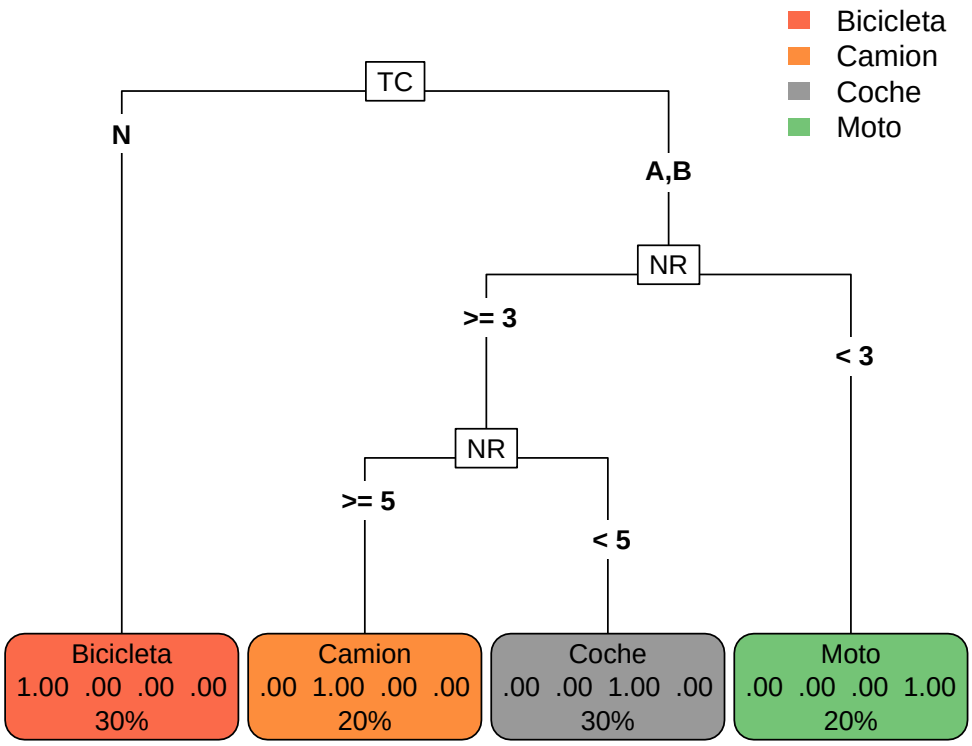
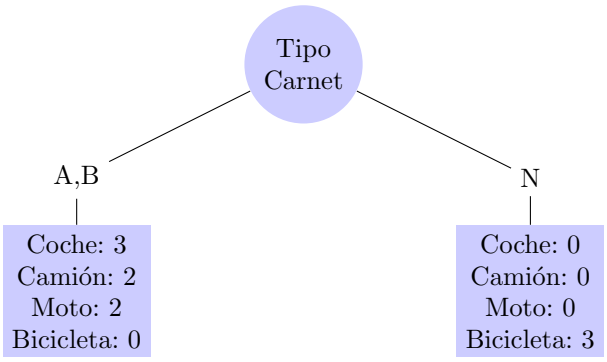


Figura 3: Árbol de decisión para la muestra de Vehículos

Como podemos ver en el árbol de descisión tenemos un clasificador en el cual partimos de la característica **TipoCarnet**. Si esa característica toma como valor una N podemos decir que el **TipoVehículo** es una Bicicleta pero si es A o B tendremos que mirar la característica **NúmeroRuedas**, la cual, si es menor que 3 es una Moto mientras que si es mayor o igual que 3 volvemos a mirar la misma caracteristica, de tal forma que si es mayor o igual que 5 es un Camión y si es menor que 5 es un Coche.

Ahora bien, ¿por qué tenemmos como **nodo raíz** a **TipoCarnet**? Para contestar a la pregunta debemos calcular la **ganancia de información** de los tres posibles casos:

1. Ganancia de información basada en el Gini del nodo **TipoCarnet**



Impurezas:

$$Gini(padre) = 1 - \sum_{i=0}^{c-1}(f_i(padre))^2 = 1 - \left(\left(\frac{3}{10}\right)^2 + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^2 + \left(\frac{3}{10}\right)^2\right) = 0{,}74$$

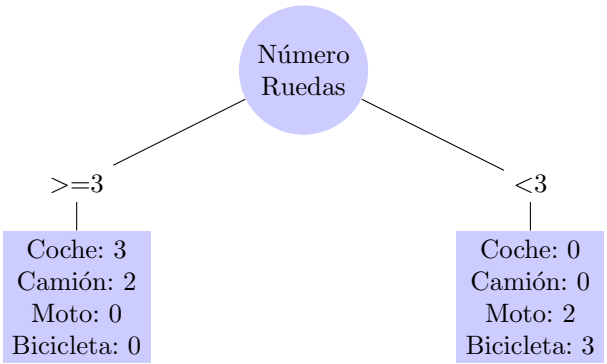
$$Gini(hijo1) = 1 - \sum_{i=0}^{c-1}(f_i(hijo1))^2 = 1 - \left(\left(\frac{3}{7}\right)^2 + \left(\frac{2}{7}\right)^2 + \left(\frac{2}{7}\right)^2 + \left(\frac{0}{7}\right)^2\right) = 0{,}65$$

$$Gini(hijo2) = 1 - \sum_{i=0}^{c-1}(f_i(hijo2))^2 = 1 - \left(\left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{3}{3}\right)^2\right) = 0$$

La última impureza sale 0 ya que hemos clasificado las bicicletas. Ganancia de información:

$$A_I = I_{padre} - \sum_{i=1}^{n} \frac{N(n_i)}{N} * I_{n_i} = 0{,}74 - ((\frac{7}{10} * 0{,}65) + (\frac{3}{10} * 0)) = 0{,}285$$

2. Ganancia de información basada en el Gini del nodo **NúmeroRuedas**



Impurezas:

$$Gini(padre) = 1 - \sum_{i=0}^{c-1}(f_i(padre))^2 = 1 - \left(\left(\frac{3}{10}\right)^2 + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^2 + \left(\frac{3}{10}\right)^2\right) = 0{,}74$$

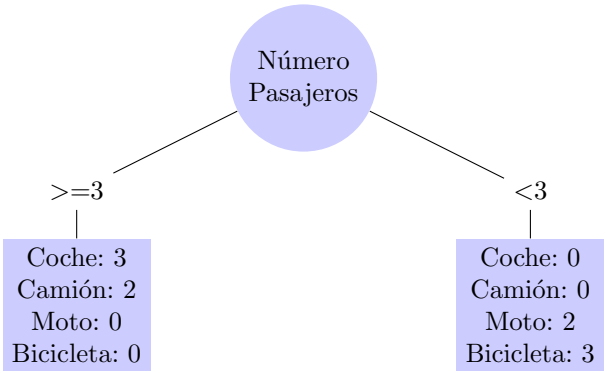
$$Gini(hijo1) = 1 - \sum_{i=0}^{c-1}(f_i(hijo1))^2 = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{0}{5}\right)^2 + \left(\frac{0}{5}\right)^2\right) = 0{,}48$$

$$Gini(hijo2) = 1 - \sum_{i=0}^{c-1}(f_i(hijo2))^2 = 1 - \left(\left(\frac{0}{5}\right)^2 + \left(\frac{0}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{3}{5}\right)^2\right) = 0{,}48$$

Ganancia de información:

$$A_I = I_{padre} - \sum_{i=1}^{n} \frac{N(n_i)}{N} * I_{n_i} = 0{,}74 - ((\frac{5}{10} * 0{,}48) + (\frac{5}{10} * 0{,}48)) = 0{,}26$$

3. Ganancia de información basada en el Gini del nodo **NúmeroPasajeros**

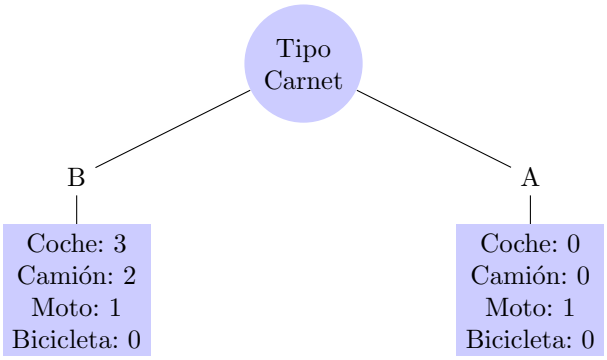


Al tener los mismos datos que el anterior tendremos una ganancia de información de 0.26.

Efectivamente, **TipoCarnet** es el nodo raíz de nuestro árbol de decisión ya que es el que **mayor** ganancia de información presenta. Veamos de la misma manera la elección de los nodos intermedios:

Para ello debemos tener en cuenta que los vehículos de tipo Bicicleta ya están clasificados y por lo tanto no los tendremos en cuenta en los siguientes pasos (quedando 7 vehículos). Calculamos la **ganancia de información** de cada uno de los posibles nodos:

1. Ganancia de información basada en el Gini del nodo **TipoCarnet**



Impurezas:

$$Gini(padre) = 1 - \sum_{i=0}^{c-1}(f_i(padre))^2 = 1 - \left(\left(\frac{3}{7}\right)^2 + \left(\frac{2}{7}\right)^2 + \left(\frac{2}{7}\right)^2 + \left(\frac{0}{7}\right)^2\right) = 0{,}65$$

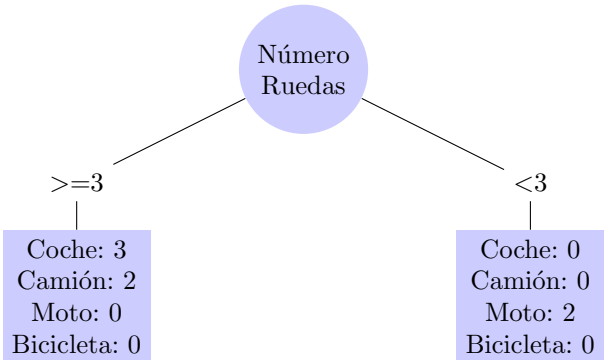
$$Gini(hijo1) = 1 - \sum_{i=0}^{c-1}(f_i(hijo1))^2 = 1 - \left(\left(\frac{3}{6}\right)^2 + \left(\frac{2}{6}\right)^2 + \left(\frac{1}{6}\right)^2 + \left(\frac{0}{6}\right)^2\right) = 0{,}61$$

$$Gini(hijo2) = 1 - \sum_{i=0}^{c-1}(f_i(hijo2))^2 = 1 - \left(\left(\frac{0}{1}\right)^2 + \left(\frac{0}{1}\right)^2 + \left(\frac{1}{1}\right)^2 + \left(\frac{0}{1}\right)^2\right) = 0$$

La última impureza sale 0 ya que hemos clasificado una de las motos. Ganancia de información:

$$A_I = I_{padre} - \sum_{i=1}^{n} \frac{N(n_i)}{N} * I_{n_i} = 0{,}65 - ((\frac{6}{7} * 0{,}61) + (\frac{1}{7} * 0)) = 0{,}13$$

2. Ganancia de información basada en el Gini del nodo **NúmeroRuedas**



Impurezas:

$$Gini(padre) = 1 - \sum_{i=0}^{c-1}(f_i(padre))^2 = 1 - \left(\left(\frac{3}{7}\right)^2 + \left(\frac{2}{7}\right)^2 + \left(\frac{2}{7}\right)^2 + \left(\frac{0}{7}\right)^2\right) = 0{,}65$$

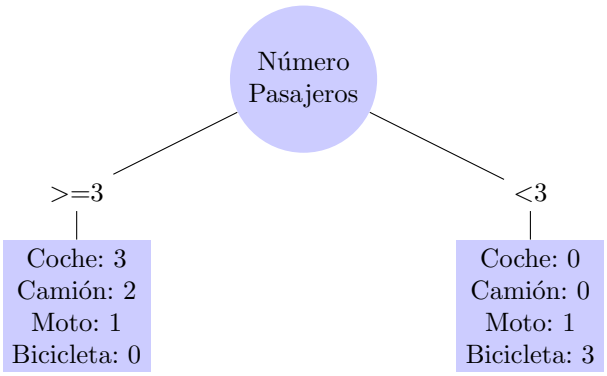
$$Gini(hijo1) = 1 - \sum_{i=0}^{c-1}(f_i(hijo1))^2 = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{0}{5}\right)^2 + \left(\frac{0}{5}\right)^2\right) = 0{,}48$$

$$Gini(hijo2) = 1 - \sum_{i=0}^{c-1}(f_i(hijo2))^2 = 1 - \left(\left(\frac{0}{2}\right)^2 + \left(\frac{0}{2}\right)^2 + \left(\frac{2}{2}\right)^2 + \left(\frac{0}{2}\right)^2\right) = 0$$

La última impureza sale 0 ya que hemos clasificado las motos. Ganancia de información:

$$A_I = I_{padre} - \sum_{i=1}^{n} \frac{N(n_i)}{N} * I_{n_i} = 0{,}65 - ((\frac{5}{7} * 0{,}48) + (\frac{2}{7} * 0)) = 0{,}3$$

3. Ganancia de información basada en el Gini del nodo **NúmeroPasajeros**

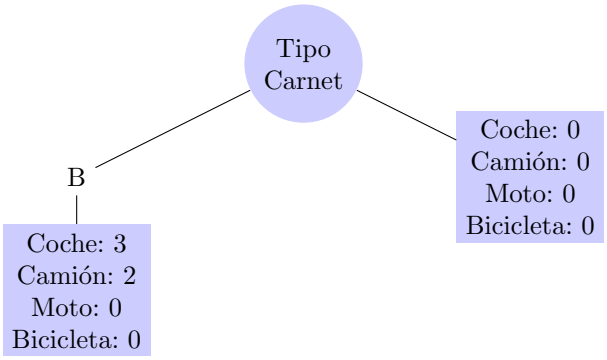


Al tener los mismos datos que el TipoVehículo tendremos una ganancia de información de 0.13.

Efectivamente, **NúmeroRuedas** es el nodo intermedio de nuestro árbol de decisión ya que es el que **mayor** ganancia de información presenta. Veamos de la misma manera la elección para el siguiente nodo intermedio:

Para ello debemos tener en cuenta que los vehículos de tipo Bicicleta y Moto ya están clasificados y por lo tanto no los tendremos en cuenta en los siguientes pasos (quedando 5 vehículos). Calculamos la **ganancia de información** de cada uno de los posibles nodos:

1. Ganancia de información basada en el Gini del nodo **TipoCarnet**



   Impurezas:

$$Gini(padre) = 1 - \sum_{i=0}^{c-1}(f_i(padre))^2 = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{0}{5}\right)^2 + \left(\frac{0}{5}\right)^2\right) = 0{,}48$$

$$Gini(hijo1) = 1 - \sum_{i=0}^{c-1}(f_i(hijo1))^2 = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{0}{5}\right)^2 + \left(\frac{0}{5}\right)^2\right) = 0{,}48$$

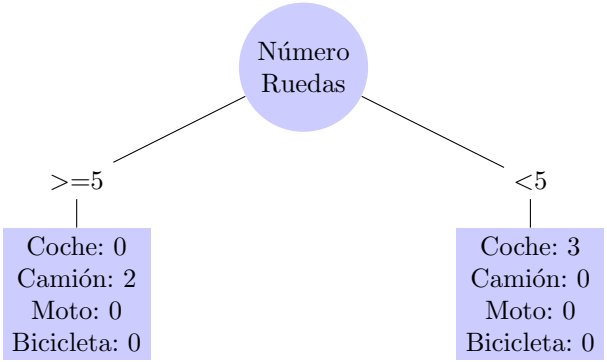
$$Gini(hijo2) = 1 - \sum_{i=0}^{c-1}(f_i(hijo2))^2 = 1 - \left(\left(\frac{0}{0}\right)^2 + \left(\frac{0}{0}\right)^2 + \left(\frac{0}{0}\right)^2 + \left(\frac{0}{0}\right)^2\right) = 0$$

   La última impureza sale 0 ya que no tenemos más datos a clasificar. Ganancia de información:

$$A_I = I_{padre} - \sum_{i=1}^{n} \frac{N(n_i)}{N} * I_{n_i} = 0{,}48 - ((\frac{5}{5} * 0{,}48) + (\frac{0}{5} * 0)) = 0$$

   Nos sale cero ya que realmente no estamos clasificando.

2. Ganancia de información basada en el Gini del nodo **NúmeroRuedas**



Impurezas:

$$Gini(padre) = 1 - \sum_{i=0}^{c-1}(f_i(padre))^2 = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{0}{5}\right)^2 + \left(\frac{0}{5}\right)^2\right) = 0{,}48$$

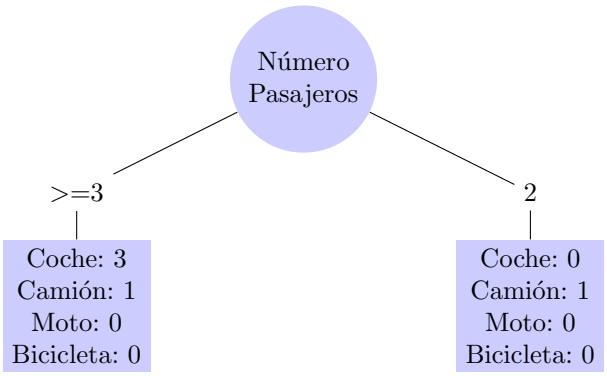
$$Gini(hijo1) = 1 - \sum_{i=0}^{c-1}(f_i(hijo1))^2 = 1 - \left(\left(\frac{2}{2}\right)^2 + \left(\frac{0}{2}\right)^2 + \left(\frac{0}{2}\right)^2 + \left(\frac{0}{2}\right)^2\right) = 0$$

$$Gini(hijo2) = 1 - \sum_{i=0}^{c-1}(f_i(hijo2))^2 = 1 - \left(\left(\frac{3}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2\right) = 0$$

Las impurezas de los hijos salen 0 ya que ambas clasifican. Ganancia de información:

$$A_I = I_{padre} - \sum_{i=1}^{n} \frac{N(n_i)}{N} * I_{n_i} = 0{,}48 - ((\frac{2}{5} * 0) + (\frac{3}{5} * 0)) = 0{,}48$$

3. Ganancia de información basada en el Gini del nodo **NúmeroPasajeros**



Impurezas:

$$Gini(padre) = 1 - \sum_{i=0}^{c-1}(f_i(padre))^2 = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{0}{5}\right)^2 + \left(\frac{0}{5}\right)^2\right) = 0{,}48$$

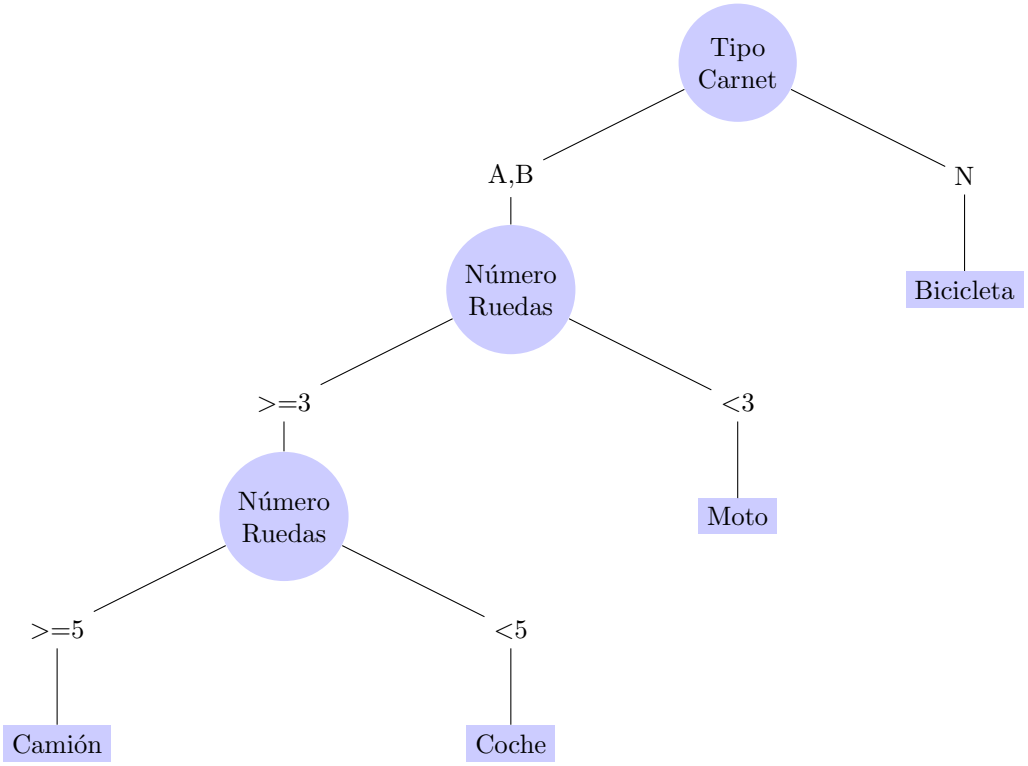
$$Gini(hijo1) = 1 - \sum_{i=0}^{c-1}(f_i(hijo1))^2 = 1 - \left(\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2 + \left(\frac{0}{4}\right)^2 + \left(\frac{0}{4}\right)^2\right) = 0{,}375$$

$$Gini(hijo2) = 1 - \sum_{i=0}^{c-1}(f_i(hijo2))^2 = 1 - \left(\left(\frac{0}{1}\right)^2 + \left(\frac{1}{1}\right)^2 + \left(\frac{0}{1}\right)^2 + \left(\frac{0}{1}\right)^2\right) = 0$$

La última impureza sale 0 ya que hemos clasificado uno de los camiones. Ganancia de información:

$$A_I = I_{padre} - \sum_{i=1}^{n} \frac{N(n_i)}{N} * I_{n_i} = 0{,}48 - ((\frac{4}{5} * 0{,}375) + (\frac{1}{5} * 0)) = 0{,}18$$

Efectivamente, **NúmeroRuedas** es el siguiente nodo intermedio de nuestro árbol de decisión ya que es el que **mayor** ganancia de información presenta. Además, estamos ante un nodo hoja ya que hemos clasificado lo que nos quedaba de la muestra. Por lo tanto el árbol final nos queda:



Como podemos observar obtenemos el mismo árbol de decisión que nos construía *rpart* aplicando la ganancia de información con el Gini.

2.2. Apartado 2.2: Regresión lineal

Al igual que hicimos en el Apartado 1.2 realizaremos un **análisis de regresión lineal**, pero en este caso utilizaremos 4 muestras:

- **Muestra 1**: {10, 8.04; 8, 6.95; 13, 7.58; 9, 8.81; 11, 8.33; 14, 9.96; 6, 7.24; 4, 4.26; 12, 10.84; 7, 4.82; 5, 5.68}
- **Muestra 2**: {10, 9.14; 8, 8.14; 13, 8.74; 9, 8.77; 11, 9.26; 14, 8.1; 6, 6.13; 4, 3.1; 12, 9.13; 7, 7.26; 5, 4.74}
- **Muestra 3**: {10, 7.46; 8, 6.77; 13, 12.74; 9, 7.11; 11, 7.81; 14, 8.84; 6, 6.08; 4, 5.39; 12, 8.15; 7, 6.42; 5, 5.73}
- **Muestra 4**: {8, 6.58; 8, 5.76; 8, 7.71; 8, 8.84; 8, 8.47; 8, 7.04; 8, 5.25; 19, 12.5; 8, 5.56; 8, 7.91; 8, 6.89}

En primer lugar, realizamos la lectura de las 4 muestras mediante la laectura del fichero *muestra4.txt* mediante la siguiente función, la cual devuelve un listado de los cuatro *dataframes*:

```
> # Funcion que lee las muestras contenidas en un fichero .txt
> # Para ello le pasamos la ruta del fichero, la separacion entre
> # los datos, la separacion entre las muestras (salto) y si tiene
> # cabecera o no.
> muestra.leer <- function(ruta,sep=" ",salto="",header=FALSE){
+   # Creamos la conexion y leemos las lineas del fichero.
+         con <- file(ruta, "r", blocking = FALSE)
+         datos <- readLines(con)
+         close(con)
+
+         # Establecemos las variables para controlar el programa
+         # y la variable dataframes donde guardaremos cada una de
+         # las muestras obtenidas.
+         tabla = NULL
+         contador = 1
+         dataframes = list()
+
+         # Leemos cada una de las lineas obtenidas anteriormente
+         # y las escribimos en un archivo temporal. Cuando detectamos
+         # el fin de la muestra, realizamos un read.table de ese
+         # archivo temporal.
+         for(linea in datos){
+                 if(linea==salto){
+                         write(tabla,file="data.temp")
+                         dataframes = append(dataframes,list(read.table("data.temp")))
+                         file.remove("data.temp")
+                         tabla = NULL
+                 } else{
+                         tabla = c(tabla,linea)
+                 }
+         }
+         if(!is.null(tabla)){
+                 write(tabla,file="data.temp")
+                 dataframes = append(dataframes,list(read.table("data.temp")))
+                 file.remove("data.temp")
+         }
```

```
+
+         dataframes
+ }
> # Lectura del fichero muestra4.txt
> muestras <- muestra.leer("muestra4.txt")
> muestras

[[1]]
    X     Y
1  10  8.04
2   8  6.95
3  13  7.58
4   9  8.81
5  11  8.33
6  14  9.96
7   6  7.24
8   4  4.26
9  12 10.84
10  7  4.82
11  5  5.68

[[2]]
    X    Y
1  10 9.14
2   8 8.14
3  13 8.74
4   9 8.77
5  11 9.26
6  14 8.10
7   6 6.13
8   4 3.10
9  12 9.13
10  7 7.26
11  5 4.74

[[3]]
    X     Y
1  10  7.46
2   8  6.77
3  13 12.74
4   9  7.11
5  11  7.81
6  14  8.84
7   6  6.08
8   4  5.39
9  12  8.15
10  7  6.42
11  5  5.73

[[4]]
```

```
    X     Y
1   8  6.58
2   8  5.76
3   8  7.71
4   8  8.84
5   8  8.47
6   8  7.04
7   8  5.25
8  19 12.50
9   8  5.56
10  8  7.91
11  8  6.89
```

A continuación, realizamos el cálculo de la media tanto para los valores de $\bar{x}$ como para los valores de $\bar{y}$:
Primera muestra

```
Media de x:  9

Media de y:  7.500909
```

Segunda muestra

```
Media de x:  9

Media de y:  7.500909
```

Tercera muestra

```
Media de x:  9

Media de y:  7.5
```

Cuarta muestra

```
Media de x:  9

Media de y:  7.500909
```

Como podmeos comprobar, las medias para las cuatro muestras son prácticamente idénticas. Una vez calculadas las medias, vamos a analizar el grado de dependencia entre la variable dependiente (Y) y la variable independiente (X), a través del cálculo de la **Covarianza** (S_{xy}):

$$S_{xy} = \frac{\sum_{i=1}^{n} x_i y_i}{n} - \bar{x}\bar{y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n}$$

Para realizar el cálculo de la **Covarianza**, creamos una función denominada *covarianza*, en la que resolvemos la ecuación anterior para cada una de las muestras:

```
> # Covarianza
> covarianza <- function(x,y){
+   (sum(x*y)/length(x))-(mean(x)*mean(y))
+ }
```

Primera muestra

```
Covarianza primera muestra:  5.000909
```

Segunda muestra

```
Covarianza segunda muestra:  5
```

Tercera muestra

```
Covarianza tercera muestra:  4.997273
```

Cuarta muestra

```
Covarianza cuarta muestra:  4.999091
```

Al igual que ocurría con la media, los valores de covarianza son prácticamente iguales. Por otro lado, los valores obtenidos son positivos, lo que e traduce en una dependencia positiva entre ambas variables.
Dado que la covarianza resulta muy difícil de poder analizar, debemos utilizar otra medida de cálculo de la dependencia, concretamente la **Correlación**:

$$r_{xy} = \frac{S_{xy}}{S_x S_y}$$

Donde S_x y S_y son las **Desviaciones típicas de X e Y**, respectivamente.
Para el cálculo de la **Correlación**, crearemos una función denominada *Correlacion*, la cual se apoya en una función para el cálculo de las **Desviaciones típicas**:

```
> # Varianza iterativa
> varianza <- function(vector){
+   media = mean(vector)
+   suma=0
+   for(i in 1:length(vector)){
+     suma = suma + (as.numeric(vector[i])-media)^2
+   }
+   suma/length(vector)
+ }
> # Desviacion tipica iterativa
> desviacion.tipica <- function(vector){
+   sqrt(varianza(vector))
+ }
> # Correlacion
> correlacion <- function(x,y){
+   covarianza(x,y)/(desviacion.tipica(x)*desviacion.tipica(y))
+ }
```

Primera muestra

```
Correlacion primera muestra:  0.8164205
```

Segunda muestra

```
Correlacion segunda muestra:  0.8162365
```

Tercera muestra

```
Correlacion tercera muestra:  0.8162867
```

Cuarta muestra

```
Correlacion cuarta muestra:  0.8165214
```

Tal y como pudimos comprobar en el cálculo de la **Covarianza**, los valores de **Correlación** obtenidos son cercanos a 1, aunque con un valor bajo. Los valores están relacionados linealmente, pero comprobemoslo relaizando la regresión y sus posteriores análisis.
Una vez obtenidos los valores de correlación y desviación típica, calculamos la ecuación de la recta para cada muestra:

$$y = bx + a$$
$$\text{Donde } b = \frac{S_{xy}}{S_x^2} \text{ y } a = \bar{y} - b\bar{x}$$

Para realizar el cálculo de la ecuación, creamos una función para el cálculo de cada término y, finalmente, almacenamos en un *dataframe* los términos de la ecuación:

```
> # Funcion de regresion
> # Encargada de obtener a y b
> regresion <- function(x,y){
+   b <- regresion.b(x,y)
+   a <- regresion.a(x,y,b)
+   data.frame(a,b)
+ }
> # Obtiene b
> regresion.b <- function(x,y){
+   covarianza(x,y)/(desviacion.tipica(x)^2)
+ }
> # Obtiene a
> regresion.a <- function(x,y,b){
+   mean(y)-(b*mean(x))
+ }
```

Primera muestra

```
Ecuacion primera muestra

Valor a:  3.000091

Valor b:  0.5000909
```

Segunda muestra

```
Ecuacion segunda muestra

Valor a:  3.000909

Valor b:  0.5
```

Tercera muestra

```
Ecuacion segunda muestra

Valor a:  3.002455

Valor b:  0.4997273
```

Cuarta muestra

```
Ecuacion segunda muestra

Valor a:  3.001727

Valor b:  0.4999091
```

Como podemos comprobar, para las cuatro muestras tenemos prácticamente la misma **ecuación de la recta**:

$$y = 0{,}5x + 3$$

Una vez obtenida la ecuación de la función de regresión, debemos analizar *cómo de buena es nuestra recta*, es decir, **analizar la relación de dispersión entre los valores de y calculados con respecto a la media, así como los valores de y iniciales con respecto a la media**, conocido como **análisis *ANOVA***:
En primer lugar, calculamos la diferencia entre los **valores de y' calculados y la media de y**, conocido como dipsersión SSR de los y calculados:

$$SSR = \sum_{i=1}^{n} (y'_i - \bar{y})^2$$

En segundo lugar, calculamos la diferencia entre los **valores de y originales y la media de y**, conocido como dipsersión SSy de los y reales:

$$SSR = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Para el análisis *ANOVA* de las muestra, utilizaremos una función denominada *anova* con la que llamaremos a una misma función para el cálculo tanto SSR como SSy, denominada *anova.ss*:

```
> # Obtiene los valores de y para unos valores de x
> # de la funcion de regresion
> regresion.ycalculada <- function(x,regresion){
+   regresion$a + x*regresion$b
+ }
> anova <- function(x,y,regresion){
+   ssr <- anova.ss(regresion.ycalculada(x,regresion),mean(y))
+   ssy <- anova.ss(y,mean(y))
+   r2 <- ssr/ssy
+   data.frame(ssr,ssy,r2)
+ }
> # SSR y SSy dependiendo de la y (observada o calculada)
> anova.ss <- function(y,media){
+   sum((y-media)^2)
+ }
```

Primera muestra

```
ANOVA primera muestra

SSR:  27.51

SSy:  41.27269

r2:  0.6665425
```

Segunda muestra

```
ANOVA segunda muestra

SSR:  27.5

SSy:  41.27629

r2:  0.666242
```

Tercera muestra

```
ANOVA tercera muestra

SSR:  27.47001

SSy:  41.2262

r2:  0.666324
```

Cuarta muestra

```
ANOVA cuarta muestra

SSR:  27.49

SSy:  41.23249

r2:  0.6667073
```

En el caso de las muestras anteriores, la correlación cuadrada es muy baja. Esto nos indica que nuestro ajuste no es demasiado bueno.
Representaremos ahora gráficamente tanto los diagramas de dispersión como las 4 rectas de regresión obtenidas en cada una de las muestras. Para ello, utilizaremos la siguiente función:

```
> # Funcion encargada de realizar las 4 regresiones lineales y
> # mostrarlas en una misma ventana (diagrama de dispersion y recta
> # de ajuste).
> mostrar.regresion <- function(ruta,sep=" ",salto="",header=FALSE){
+   dataframe <- muestra.leer(ruta,sep,salto,header)
+   par(mfrow=c(2,2))
+   for (i in 1:length(dataframe)){
+     data <- dataframe[[i]]
+     regresion <- lm(Y~X,data = data)
+     main <- paste("Muestra ",i)
+     plot(data$X,data$Y,xlim=c(0,20),ylim=c(0,14),xlab='x', ylab='y', main = main)
+     abline(regresion,col='red')
+   }
+ }
```

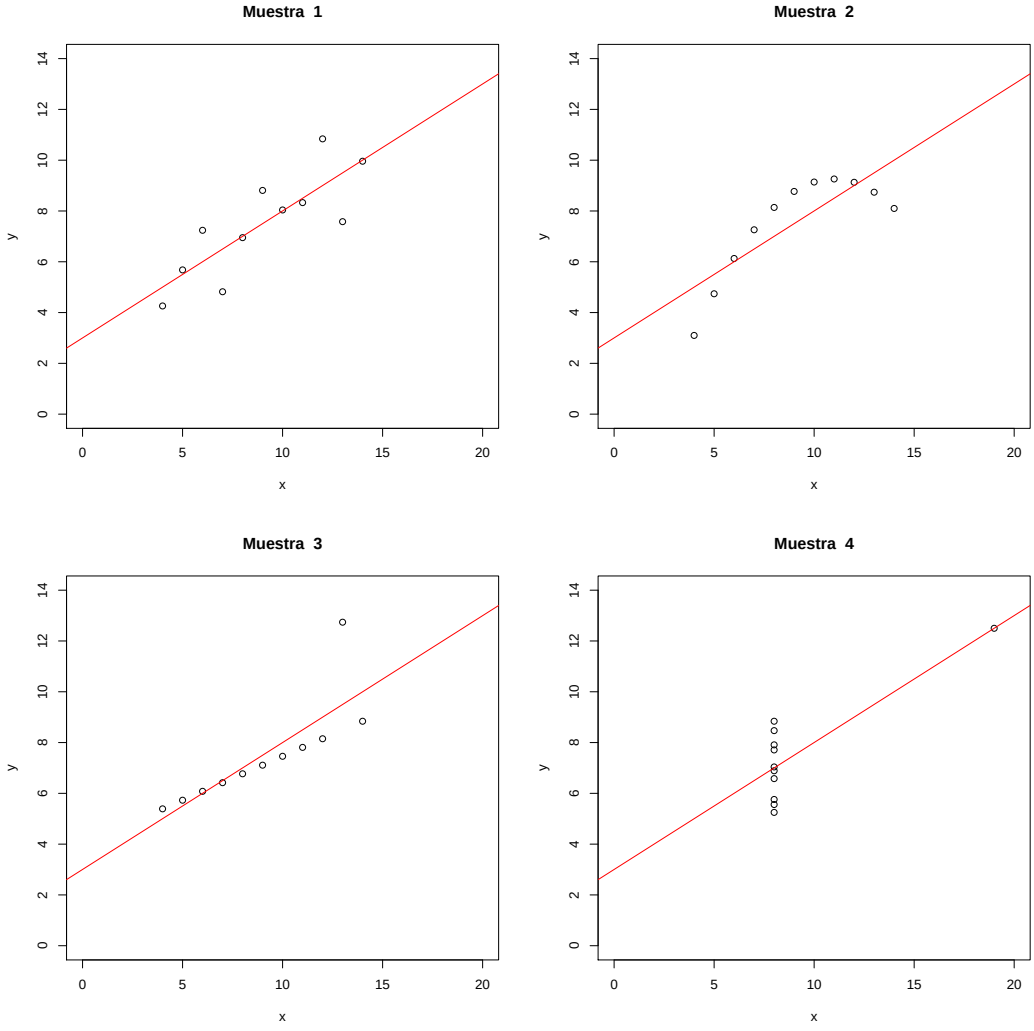


Figura 4: Funciones de regresión

Si observamos las gráficas resultantes, podemos ver que:

- En la **muestra 1**, la relación entre los puntos podría ser lineal, pero están demasiado dispersos, por lo que su correlación no es demasiado alta (apenas un 0.8) y su coeficiente de correlación cuadrada es bastante bajo.

- En la **muestra 2**, la relación entre los puntos no es lineal en absoluto. Por ello su correlación es de apenas un 0.8 y su correlación cuadrada es tan baja. Sería necesario otro tipo de ajuste.

- En la **muestra 3**, se ve claramente que todos los datos mantienen una relación lineal casi perfecta excepto por un punto. A causa de este la correlación y la correlación cuadrada son bajas. En este caso lo mejor sería suprimir dicho punto de nuestro ajuste (habría que ver las características del problema).

- En la **muestra 4** ocurre algo similar a la muestra 3. Todos los datos excepto 1 podrían ajustarse de forma casi perfecta a una recta, su relación es lineal. La razón por la que su correlación y correlación cuadrada son tan bajas como las anteriores es la presencia de dicho valor extremo. Una vez más, lo mejor sería intentar descartarlo a la hora de realizar el ajuste.

Por último, realizaremos un análisis del **error estándar de la estimación o desviación típica residual**, calculando el error estándar obtenido de los residuos, o diferencia entre los valores de y observados y los valores de y calculados empleando la función de regresión:

$$S_r = \sqrt{\frac{\sum_{i=1}^{n}(y_i - y_i')^2}{n}}$$

Para el cálculo del **Error estándar** creamos una función a la que llamaremos *error.estandar*, en la que implementamos la ecuación anterior:

```
> # Error estandar
> error.estandar <- function(y,regresion){
+   sqrt(sum((y-regresion.ycalculada(y,regresion))^2)/length(y))
+ }
```

Primera muestra

```
Sr:  1.224621
```

Segunda muestra

```
Sr:  1.224711
```

Tercera muestra

```
Sr:  1.224691
```

Cuarta muestra

```
Sr:  1.22436
```

Los resultados obtenidos (valores de error lejanos de cero), implican **un mal ajuste de la función**. Efectivamente, como vimos en el análisis ANOVA, las respectivas rectas no ajustan muy bien los datos por las razones expuestas. Si representamos ahora gráficamente las rectas paralelas a la de ajuste a distancias **Sr** y **2Sr** respectivamente, podemos demostrar cómo el 95 % de los puntos están situados en la región comprendida entre dos líneas paralelas a la recta de regresión separadas 4 S_r veces, 2 veces a cada lado de la recta, mientras que el 66 % de los datos están separados 2 S_r veces, una a cada lado de la recta. Para demostrarlo, utilizaremos la siguiente función:

```
> # Funcion encargada de mostrar graficamente el error
> # estandar junto con las rectas del 95% y 66%.
> error.estandar.plot <- function(x,y,regresion){
+   sr <- error.estandar(y,regresion)
+   plot(x,y,xlim=c(0,ceiling(max(x))),ylim=c(floor(regresion$a),ceiling(max(y))),xlab='x', ylab='y', ma
+   abline(regresion$a,regresion$b,col='red')
+
+   abline(regresion$a+sr,regresion$b,col='green')
+   abline(regresion$a-sr,regresion$b,col='green')
+
+   abline(regresion$a+2*sr,regresion$b,col='blue')
+   abline(regresion$a-2*sr,regresion$b,col='blue')
+   legend(x="topleft",legend=c("Regresión","66%","95%"),fill=c("red","green","blue"))
+ }
```

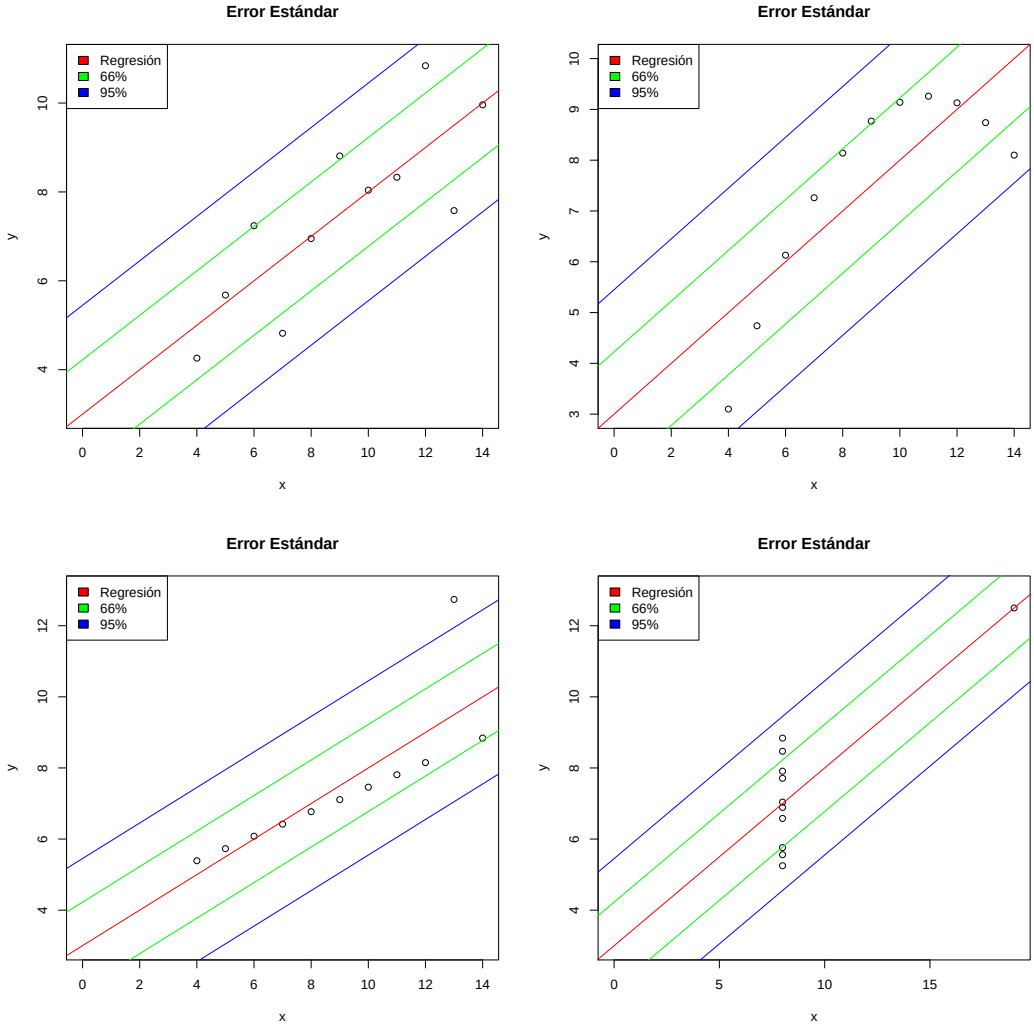


Figura 5: Error estándar (66 y 95 %)

2.3. Apartado 2.3: Otros algoritmos

Este apartado consistirá en el desarrollo de análisis de clasificación con `R` utilizando diferentes métodos de **clasificación supervisada**:

1. *Random Forest*
2. *Regresión Logística*
3. *Redes Neuronales*

2.3.1. Random Forest

La técnica del **Random Forest** consiste en crear varios árboles de decisión **independientes**, probados sobre conjuntos de datos aleatorios de igual distribución. Supongamos que tenemos el siguiente *dataset*:

edad	ingresos	sexo	casado
35	40006	F	No
50	40000	M	Si
22	70000	M	No
28	50000	F	No
41	90000	F	Si
32	60000	F	Si
21	120000	M	Si
60	70000	F	Si

Durante la **fase de aprendizaje**, se crean múltiples árboles de decisión independientes, a partir de un subconjunto de los datos de entrada ligeramente distintos. Para ello, **seleccionamos aleatoriamente con reemplazamiento un porcentaje de datos de la muestra total**. Comúnmente, se incluye un segundo nivel de aleatoriedad, seleccionando una porción de atributos de forma aleatoria.
Por tanto, cada árbol es construido mediante el siguiente algoritmo:

1. Sea N el número de filas del *dataset* y M el número total de atributos.
2. Sea m el número de atributos de entrada empleado para la construcción de un nodo del árbol de decisión, donde $m \leq M$.
3. Se elige un conjunto de entrenamiento para cada árbol, usando el resto de datos no utilizados para calcular el error.
4. Para cada nodo, elegimos aleatoriamente m filas para realizar la decisión.

Supongamos que creamos un total de tres árboles. Cada uno de ellos utilizará un subconjunto aleatorio de los datos, utilizando en este caso todas las columnas:

Cuadro 1: Árbol 1

Edad	Ingresos	Sexo	Casado
50	40000	M	Si
28	50000	F	No
32	60000	F	Si
21	120000	M	Si

Cuadro 2: Árbol 2

Edad	Ingresos	Sexo	Casado
35	40006	F	No
28	50000	F	No
41	90000	F	Si
21	120000	M	Si

Cuadro 3: Árbol 3

Edad	Ingresos	Sexo	Casado
22	70000	M	No
28	50000	F	No
41	90000	F	Si
60	70000	F	Si

Una vez que tengamos los árboles, la siguiente fase (**clasificación**), evalúa cada árbol de forma independiente. Una vez construidos y optimizados, la predicción final del bosque será una **media de los árboles construidos**. Veamos un ejemplo de ejecución con la **clasificación de la calidad de un diamante** (*Fair, Good, Very Good, Premium, Ideal*). [1]:

- Anchura del diamante (*carat*)
- Calidad del diamante (*cut*)
- Color del diamante (*color*)
- Claridad del diamante (*clarity*)
- Porcentaje de altura del diamante (*depth*)
- Porcentaje de anchura del diamante (*table*)
- Precio del diamante (*price*)
- Altura en mm (x)
- Anchura en mm (y)
- Profundidad en mm (z)

Inicialmente, leemos el fichero *.csv*. Como el fichero original contiene 54.000 filas, vamos a utilizar un subconjunto, con un total de 500 filas, mediante el paquete *dplyr* [2], el cual nos permite realizar consultas a un dataframe similares a una consulta `SQL`:

```
> diamonds <- read.csv("diamonds.csv", sep = ",")
> if(!require(dplyr)){
+   install.packages("dplyr")
+   require(dplyr)
+ }
> #Seleccionamos las 500 primeras filas, similar a:
> #SELECT * FROM diamonds LIMIT 500
> diamonds_reduce <- diamonds %>% head(500)
> diamonds_reduce %>% head(5)

  X carat     cut color clarity depth table price    x    y    z
1 1  0.23   Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2 2  0.21 Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3 3  0.23    Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4 4  0.29 Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5 5  0.31    Good     J     SI2  63.3    58   335 4.34 4.35 2.75
```

[1] https://www.kaggle.com/shivam2503/diamonds/download
[2] https://www.rdocumentation.org/packages/dplyr/versions/0.7.8

Como podemos comprobar a partir de la ejecución anterior, existe una columna que enumera las filas del *dataframe*, por lo que debemos eliminarla. Para ello, indicamos el número de columna a eliminar:

```
> #Eliminamos la primera columna (X)
> diamonds_reduce <- diamonds_reduce[-1]
> diamonds_reduce %>% head(5)
```

```
  carat     cut color clarity depth table price    x    y    z
1  0.23   Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21 Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23    Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29 Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31    Good     J     SI2  63.3    58   335 4.34 4.35 2.75
```

Una vez eliminada la columna, vamos a crear un *randomForest*. Para ello utilizaremos la función *randomForest* disponible en la librería *randomForest* [3]. Por defecto, el número de árboles aleatorios generados son 500, aunque se pueden modificar con el parámetro *ntree*.

```
> if(!require(randomForest)){
+   install.packages("randomForest")
+   require(randomForest)
+ }
> #Elegimos la calidad del diamante (cut)
> #como clasificador
> rf <- randomForest(cut ~.,diamonds_reduce)
```

Para analizar el contenido del árbol, utilizaremos la función *getTree*. Por defecto, de todos los árboles construidos muestra el primero. Si queremos mostrar cualquier otro, añadimos el parámetro k a la función:

```
> #getTree(rf, k = 1), por defecto
> getTree(rf) %>% head(10)
```

```
   left daughter right daughter split var split point status prediction
1              2              3         9       3.915      1          0
2              4              5         9       2.600      1          0
3              6              7         4      63.800      1          0
4              8              9         8       3.845      1          0
5             10             11         1       0.670      1          0
6             12             13         9       4.015      1          0
7              0              0         0       0.000     -1          1
8              0              0         0       0.000     -1          4
9             14             15         2      48.000      1          0
10            16             17         8       4.450      1          0
```

La función *getTree* nos muestra un *dataframe*, cuyo número de filas corresponde con el número de nodos del árbol:

```
> nrow(getTree(rf))
```

```
[1] 255
```

[3] https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

Analicemos en detalle cada una de las filas del *dataframe*:

- *left daughter*: por cada nodo indica en qué fila se encuentra el nodo hijo izquierdo (si el nodo es terminal, *left daughter* = 0).
- *right daughter*: por cada nodo indica en qué fila se encuentra el nodo hijo derecho (si el nodo es terminal, *right daughter* = 0).
- *split var*: indica qué variable se ha empleado para dividir el nodo en sus correspondientes nodos hijos (0 si se trata de un nodo terminal). En el primer nodo, se ha empleado la columna 5, es decir, el atributo *depth*.
- *split point*: indica dónde se ha producido la mejor partición.
- *status*: indica si el nodo es terminal (-1) o no (1).
- *prediction*: indica la predicción para cada nodo, 0 si se trata de un nodo terminal.

Una vez construidos los árboles, vamos a intentar predecir la calidad de 5 muestras del *dataframe* original:

```
> #Seleccionamos filas con diferentes calidades
> diamonds_test <- diamonds[c(501,502,505,511,515),]
> #Eliminamos la columna X que cuenta el numero de fila
> diamonds_test <- diamonds_test[-1]
> diamonds_test
```

```
    carat       cut color clarity depth table price    x    y    z
501  0.71     Ideal     D     SI1  60.2    56  2822 5.86 5.83 3.52
502  0.70   Premium     E     VS2  61.5    59  2822 5.73 5.68 3.51
505  0.70      Good     E     SI1  61.4    64  2822 5.71 5.66 3.49
511  0.79 Very Good     D     SI2  62.8    59  2823 5.86 5.90 3.69
515  0.90      Fair     H     SI2  65.8    54  2823 6.05 5.98 3.96
```

A continuación, eliminamos los valores de la columna *cut*:

```
> diamonds_test$cut <- NA
> diamonds_test
```

```
    carat cut color clarity depth table price    x    y    z
501  0.71  NA     D     SI1  60.2    56  2822 5.86 5.83 3.52
502  0.70  NA     E     VS2  61.5    59  2822 5.73 5.68 3.51
505  0.70  NA     E     SI1  61.4    64  2822 5.71 5.66 3.49
511  0.79  NA     D     SI2  62.8    59  2823 5.86 5.90 3.69
515  0.90  NA     H     SI2  65.8    54  2823 6.05 5.98 3.96
```

Para realizar la predicción, utilizaremos la función *predict*, en el que indicaremos como parámetros el *Random Forest* creado, así como el *dataframe* de prueba:

```
> predict(rf, newdata = diamonds_test)
```

```
    501     502     505     511     515
  Ideal Premium Premium Premium    Fair
Levels: Fair Good Ideal Premium Very Good
```

Como podemos comprobar, ha predicho correctamente tres de las cinco muestras. Si aumentásemos el número de datos de entrenamiento, lo más probable es que el porcentaje de aciertos aumente.
Para mostrar gráficamente un árbol del *Random Forest*, utilizaremos el paquete *reprtree*. Para instalarlo, ejecutamos la siguiente línea de comandos:

```
> if(!require(reprtree)){
+   options(repos='http://cran.rstudio.org')
+   have.packages <- installed.packages()
+   cran.packages <- c('devtools','plotrix','randomForest','tree')
+   to.install <- setdiff(cran.packages, have.packages[,1])
+   if(length(to.install)>0) install.packages(to.install)
+
+     library(devtools)
+   if(!('reprtree' %in% installed.packages())){
+     install_github('araastat/reprtree')
+   }
+   for(p in c(cran.packages, 'reprtree')) eval(substitute(library(pkg), list(pkg=p)))
+   require(reprtree)
+ }
```

Una vez descargada e importada la librería, vamos a representar gráficamente uno de los árboles del *Random Forest*. Dado que el árbol generado anteriormente presenta una gran cantidad de nodos, vamos a representar un árbol utilizando 100 datos de entrenamiento:

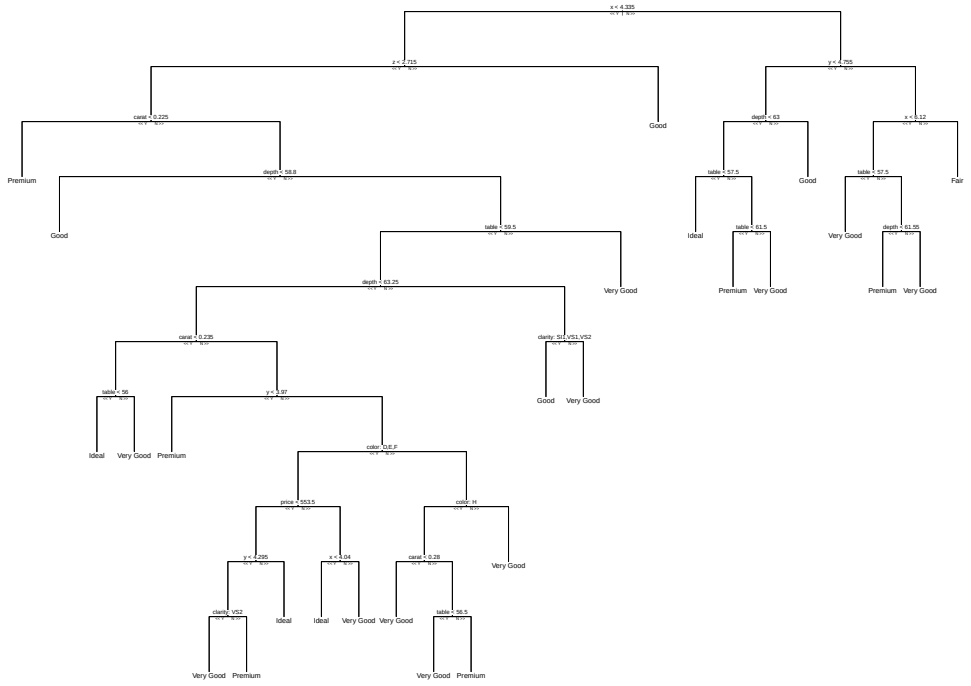


Figura 6: Random Forest

2.3.2. Logistic Regression

Se trata de un tipo específico de **análisis de regresión** con el que se pretende **precedir el resultado de variables categóricas**, es decir, variables que presentan un número limitado de posibles valores (principal diferencia con respecto a la **regresión lineal**), en función del resto de variables independientes.
A modo de ejemplo vamos a **predecir si un pasajero embarcado en el Titanic pudo o no sobrevivir**. [4]
Comenzamos con la fase de entrenamiento, utilizando para ello el dataset *train_titanic.csv*:

```
> train_titanic <- read.csv("train_titanic.csv")
> #Vamos a visualizar los 10 primeros datos
> #mediante la libreria dplyr
> train_titanic %>% head(10)

   PassengerId Survived Pclass
1            1        0      3
2            2        1      1
3            3        1      3
4            4        1      1
5            5        0      3
```

[4] https://www.kaggle.com/jeremyd/titanic-logistic-regression-in-r/notebook

```
6            6        0      3
7            7        0      1
8            8        0      3
9            9        1      3
10          10        1      2
                                                 Name    Sex Age SibSp Parch
1                             Braund, Mr. Owen Harris   male  22     1     0
2  Cumings, Mrs. John Bradley (Florence Briggs Thayer) female  38     1     0
3                              Heikkinen, Miss. Laina female  26     0     0
4        Futrelle, Mrs. Jacques Heath (Lily May Peel) female  35     1     0
5                            Allen, Mr. William Henry   male  35     0     0
6                                    Moran, Mr. James   male  NA     0     0
7                             McCarthy, Mr. Timothy J   male  54     0     0
8                      Palsson, Master. Gosta Leonard   male   2     3     1
9    Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg) female  27     0     2
10                Nasser, Mrs. Nicholas (Adele Achem) female  14     1     0
             Ticket    Fare Cabin Embarked
1         A/5 21171  7.2500              S
2          PC 17599 71.2833   C85        C
3  STON/O2. 3101282  7.9250              S
4            113803 53.1000  C123        S
5            373450  8.0500              S
6            330877  8.4583              Q
7             17463 51.8625   E46        S
8            349909 21.0750              S
9            347742 11.1333              S
10           237736 30.0708              C
```

1. *PassengerId*: id del pasajero.
2. *PClass*: clase en la que viajaba el pasajero (primera, segunda o tercera).
3. *Name*: nombre del pasajero.
4. *Sex*: sexo del pasajero.
5. *Age*: edad del pasajero.
6. *SibSp*: número de hermanos o parejas a bordo.
7. *Parch*: número de padres o hijos a bordo.
8. *Ticket*: id del billete.
9. *Fare*: precio pagado por el billete.
10. *Cabin*: número de la cabina.
11. *Embarked*: puerto donde embarcó el pasajero (C - Cherbourg, S - Southampton, Q - Queenstown).

Una vez cargado el fichero, eliminamos aquellos campos no relevantes para la regresión, tales como *PassengerId*, *Name*, *Ticket*, *Cabin* y *Embarked*:

```
> #Eliminamos las columnas anteriores
> train_titanic <- train_titanic[c(-1,-4,-9,-11,-12)]
```

A continuación, debemos eliminar filas con algún campo a *NA*. Para ello utilizamos la función *na.omit()*:

```
> sum(is.na(train_titanic))

[1] 177

> #Eliminamos las filas con campos a NA
> train_titanic <- na.omit(train_titanic)
> sum(is.na(train_titanic))

[1] 0
```

Una vez eliminadas dichas filas, debemos pasar a formato **numérico** el campo *Sex*: 2 para el hombre y 1 para la mujer. A continuación, visualizamos la matriz de correlación mediante la función *cor*, disponible en el paquete *stats*:

```
> train_titanic$Sex = as.numeric(train_titanic$Sex)
> #Matriz de correlacion
> cor(train_titanic)

            Survived      Pclass         Sex         Age       SibSp
Survived  1.00000000 -0.35965268 -0.53882559 -0.07722109 -0.01735836
Pclass   -0.35965268  1.00000000  0.15546030 -0.36922602  0.06724737
Sex      -0.53882559  0.15546030  1.00000000  0.09325358 -0.10394968
Age      -0.07722109 -0.36922602  0.09325358  1.00000000 -0.30824676
SibSp    -0.01735836  0.06724737 -0.10394968 -0.30824676  1.00000000
Parch     0.09331701  0.02568307 -0.24697204 -0.18911926  0.38381986
Fare      0.26818862 -0.55418247 -0.18499425  0.09606669  0.13832879
               Parch        Fare
Survived  0.09331701  0.26818862
Pclass    0.02568307 -0.55418247
Sex      -0.24697204 -0.18499425
Age      -0.18911926  0.09606669
SibSp     0.38381986  0.13832879
Parch     1.00000000  0.20511888
Fare      0.20511888  1.00000000
```

Como queremos predecir si un pasajero o no sobrevivió, nos fijamos en la primera fila de la matriz. Tal y como podemos observar, el campo Supervivencia depende en mayor medida del camarote en el que se encontrará el pasajero (*Pclass*), el sexo del pasajero y, en menor medida, del precio del billete.
A continuación, creamos nuestro modelo de **regresión logística** mediante la función *glm*, disponible en el paquete *stats*:

```
> #La regresion logistica analiza los datos mediante una distribucion
> #binomial, por lo que debemos indicarlo en el campo family
> titanic_lr <- glm(Survived~., data = train_titanic, family = binomial)
> titanic_lr

Call:  glm(formula = Survived ~ ., family = binomial, data = train_titanic)

Coefficients:
(Intercept)       Pclass          Sex          Age        SibSp        Parch
    8.02385     -1.24225     -2.63484     -0.04395     -0.37575     -0.06194
```

```
       Fare
    0.00216

Degrees of Freedom: 713 Total (i.e. Null);  707 Residual
Null Deviance:            964.5
Residual Deviance: 635.8         AIC: 649.8
```

```
> summary(titanic_lr)

Call:
glm(formula = Survived ~ ., family = binomial, data = train_titanic)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7953  -0.6476  -0.3847   0.6271   2.4433

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  8.023848   0.724047  11.082  < 2e-16 ***
Pclass      -1.242249   0.163191  -7.612 2.69e-14 ***
Sex         -2.634845   0.219609 -11.998  < 2e-16 ***
Age         -0.043953   0.008179  -5.374 7.70e-08 ***
SibSp       -0.375755   0.127361  -2.950  0.00317 **
Parch       -0.061937   0.122925  -0.504  0.61436
Fare         0.002160   0.002493   0.866  0.38627
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 964.52  on 713  degrees of freedom
Residual deviance: 635.81  on 707  degrees of freedom
AIC: 649.81

Number of Fisher Scoring iterations: 5
```

Como podemos comprobar obtenemos la siguiente función de regresión:

$y = -1{,}24225 \times Pclass - 2{,}63484 \times Sex - 0{,}04395 \times Age - 0{,}37575 \times SibSp - 0{,}06194 \times Parch + 0{,}00216 \times Fare$

Una vez terminada la fase de **entrenamiento**, procedemos con la fase de prueba. Para ello utilizaremos un *dataset* aparte, al que hemos llamado *test_titanic.csv*:

```
> #En primer lugar leemos el fichero csv
> test_titanic <- read.csv("test_titanic.csv")
> #Creamos un dataframe donde nos quedaremos con los
> #ids de filas sin valores a NA para mas adelante
> id <- na.omit(test_titanic) %>% select(PassengerId)
> #Eliminamos las columnas que no vayamos a utilizar
> test_titanic <- test_titanic[c(-1,-3,-8,-10,-11)]
> #Eliminamos posibles filas con campos a NA
```

```
> test_titanic <- na.omit(test_titanic)
> #Cambiamos el campo Sex a tipo numeric
> test_titanic$Sex <- as.numeric(test_titanic$Sex)
```

Para realizar la predicción, utilizaremos la función *predict*:

```
> #Response para regresiones de tipo lineal
> predictTest <- predict(titanic_lr, type = "response", newdata = test_titanic)
> #Añadimos una nueva columna al dataframe test_titanic, indicando si el
> #pasajero sobrevivio o no al hundimiento. Para limitar los posibles valores
> #a 0 o 1, si el valor predicho es mayor o igual a 0.5 lo ponemos a 1,
> #mientras que cualquier valor inferior a 0.5 lo limitamos a 0
> test_titanic$Survived <- as.numeric(predictTest >= 0.5)
> #Analizamos el campo Survived
> table(test_titanic$Survived)

  0   1
197 134
```

Como podemos comprobar, según nuestro modelo de predicción, 134 personas sobrevivieron al hundimiento, mientras que 197 no.
Finalmente, vamos a comprobar la precisión del modelo. Para ello, *Kaggle* dispone de un fichero *csv* con el campo *Survived* para los pasajeros del fichero *test_titanic.csv*. Vamos a comprobar el porcentaje de aciertos. Dado que el fichero original contiene filas a *NA*, nos quedaremos con aquellas filas del fichero cuyos ids correspondan con filas con campos completos en *test_titanic.csv*:

```
> real_test_data <- read.csv("real_data.csv")
> real_test_data <- subset(real_test_data, (real_test_data$PassengerId %in% id$PassengerId))
> rbind(table(test_titanic$Survived), table(real_test_data$Survived))

      0   1
[1,] 197 134
[2,] 204 127
```

Como podemos observar, de un total de 204 fallecidos, el sistema ha predicho correctamente 197 (7 falsos positivos), mientras que de un total de 127 supervivientes el sistema ha predicho 134 (7 falsos negativos).

2.3.3. Redes Neuronales (*ANN*)

Las Redes Neuronales (conocidas actualmente en el campo empresarial como *Deep Learning*), son un modelo computacional inspirado en su homólogo biológico: consisten en un conjunto de unidades elementales, denominadas **neuronas**, conectadas entre sí para transmitir información, generando unos valores a la salida de la red. Cada pareja de neuronas está conectada por medio de **enlaces**. Por cada enlace, el valor a la salida de la neurona es multiplicado por un valor denominado **peso**, así como posibles funciones limitadoras o **umbral** que modifiquen el valor a la salida. Generalmente, una red se divide en tres capas:

1. **Capa de entrada**: capa inicial en el que se produce la entrada de los datos.
2. **Capa oculta**: la clave de las redes neuronales es el **aprendizaje automático**, por lo que la idea es reducir el error (en el caso de las redes neuronales una **función de pérdida**) que evalúa en su total la red. Para ello, a lo largo de la capa intermedia se produce la **actualización de los valores de peso**, mediante un proceso conocido como **propagación hacia atrás** o *backpropagation*.
3. **Capa de salida**: capa en la que se produce la salida de las predicciones obtenidas a partir de los datos de entrada.

Para este tipo de clasificación, vamos a utilizar un *dataset* con el **historial clínico de pacientes con cáncer de mama** [5], en el que se identifican dos tipos de diagnósticos:

- Beningno (B): tumor que no se extiende a otras partes del cuerpo, por lo que no presenta graves consecuencias para el organismo.
- Maligno (M): tumor que invade otros tejidos del organismo, extendiéndose a otras partes del cuerpo.

Por tanto, el objetivo consistirá en diseñar una red neuronal que permita **clasificar y predecir el diagnóstico de los pacientes**. En primer lugar, comenzamos con la fase de entrenamiento, leyendo el fichero *csv* con los datos a analizar:

```
>   breast_cancer <- read.csv("wisc_bc_data.csv")
>   #Vamos a visualizar los 5 primeros datos para
>   #ver su contenido, utilizando la libreria dplyr
>   require(dplyr)
>   breast_cancer %>% head(5)
```

```
        id diagnosis radius_mean texture_mean perimeter_mean area_mean
1 87139402         B       12.32        12.39          78.85     464.1
2  8910251         B       10.60        18.95          69.28     346.4
3   905520         B       11.04        16.83          70.92     373.2
4   868871         B       11.28        13.39          73.00     384.8
5  9012568         B       15.19        13.21          97.65     711.8
  smoothness_mean compactness_mean concavity_mean points_mean symmetry_mean
1         0.10280          0.06981        0.03987     0.03700        0.1959
2         0.09688          0.11470        0.06387     0.02642        0.1922
3         0.10770          0.07804        0.03046     0.02480        0.1714
4         0.11640          0.11360        0.04635     0.04796        0.1771
5         0.07963          0.06934        0.03393     0.02657        0.1721
  dimension_mean radius_se texture_se perimeter_se area_se smoothness_se
1        0.05955    0.2360     0.6656        1.670   17.43      0.008045
2        0.06491    0.4505     1.1970        3.430   27.10      0.007470
3        0.06340    0.1967     1.3870        1.342   13.54      0.005158
4        0.06072    0.3384     1.3430        1.851   26.33      0.011270
5        0.05544    0.1783     0.4125        1.338   17.72      0.005012
  compactness_se concavity_se points_se symmetry_se dimension_se radius_worst
1       0.011800      0.01683  0.012410     0.01924     0.002248        13.50
2       0.035810      0.03354  0.013650     0.03504     0.003318        11.88
```

[5] https://www.kaggle.com/anacoder1/wisc-bc-data/download

```
3       0.009355      0.01056  0.007483     0.01718     0.002198        12.41
4       0.034980      0.02187  0.019650     0.01580     0.003442        11.92
5       0.014850      0.01551  0.009155     0.01647     0.001767        16.20
  texture_worst perimeter_worst area_worst smoothness_worst compactness_worst
1         15.64           86.97      549.1           0.1385            0.1266
2         22.94           78.28      424.8           0.1213            0.2515
3         26.44           79.93      471.4           0.1369            0.1482
4         15.77           76.53      434.0           0.1367            0.1822
5         15.73          104.50      819.1           0.1126            0.1737
  concavity_worst points_worst symmetry_worst dimension_worst
1         0.12420      0.09391         0.2827         0.06771
2         0.19160      0.07926         0.2940         0.07587
3         0.10670      0.07431         0.2998         0.07881
4         0.08669      0.08611         0.2102         0.06784
5         0.13620      0.08178         0.2487         0.06766
```

```
>   #Eliminamos la columna id
>   breast_cancer <- breast_cancer[-1]
```

Como podemos comprobar, junto con el diagnóstico final, el *dataset* contiene características morfólogicas del pecho, tales como **perímetro**, **área**, **suavidad**, **concavidad**, **simetría** etc. Una vez extraídos los datos, vamos a dividir el conjunto en dos partes: **datos de entrenamiento** (70 % de los datos originales) y **datos de validación** (30 % restante). Mediante la función *sample* tomamos una muestra del tamaño especificado sobre el *dataframe*, devolviendo como resultado las filas resultantes:

```
> #Datos de entrenamiento 70%
> breast_cancer.Train <- breast_cancer[sample(nrow(breast_cancer), nrow(breast_cancer)*0.70), ]
> #Datos de validacion 30%
> breast_cancer.Val <- breast_cancer[setdiff(1:nrow(breast_cancer), breast_cancer.Train), ]
```

```
> #Veamos alguno de los datos
> #Entrenamiento
> breast_cancer.Train[1:3, ]
```

```
    diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
73          B       9.042        18.90          60.07     244.5         0.09968
89          M      21.160        23.04         137.20    1404.0         0.09428
326         M      18.460        18.52         121.10    1075.0         0.09874
    compactness_mean concavity_mean points_mean symmetry_mean dimension_mean
73            0.1972         0.1975     0.04908        0.2330        0.08743
89            0.1022         0.1097     0.08632        0.1769        0.05278
326           0.1053         0.1335     0.08795        0.2132        0.06022
    radius_se texture_se perimeter_se area_se smoothness_se compactness_se
73     0.4653      1.911        3.769   24.20      0.009845        0.06590
89     0.6917      1.127        4.303   93.99      0.004728        0.01259
326    0.6997      1.475        4.782   80.60      0.006471        0.01649
    concavity_se points_se symmetry_se dimension_se radius_worst texture_worst
73       0.10270   0.02527     0.03491     0.007877        10.06         23.40
89       0.01715   0.01038     0.01083     0.001987        29.17         35.59
326      0.02806   0.01420     0.02370     0.003755        22.93         27.68
    perimeter_worst area_worst smoothness_worst compactness_worst
73            68.62      297.1           0.1221            0.3748
89           188.00     2615.0           0.1401            0.2600
326          152.20     1603.0           0.1398            0.2089
    concavity_worst points_worst symmetry_worst dimension_worst
73           0.4609       0.1145         0.3135         0.10550
89           0.3155       0.2009         0.2822         0.07526
326          0.3157       0.1642         0.3695         0.08579
```

```
>
```

```
> #Validacion
> breast_cancer.Val[1:3, ]
  diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
1         B       12.32        12.39          78.85     464.1         0.10280
2         B       10.60        18.95          69.28     346.4         0.09688
3         B       11.04        16.83          70.92     373.2         0.10770
  compactness_mean concavity_mean points_mean symmetry_mean dimension_mean
1          0.06981        0.03987     0.03700        0.1959        0.05955
2          0.11470        0.06387     0.02642        0.1922        0.06491
3          0.07804        0.03046     0.02480        0.1714        0.06340
  radius_se texture_se perimeter_se area_se smoothness_se compactness_se
1    0.2360     0.6656        1.670   17.43      0.008045       0.011800
2    0.4505     1.1970        3.430   27.10      0.007470       0.035810
3    0.1967     1.3870        1.342   13.54      0.005158       0.009355
  concavity_se points_se symmetry_se dimension_se radius_worst texture_worst
1      0.01683  0.012410     0.01924     0.002248        13.50         15.64
2      0.03354  0.013650     0.03504     0.003318        11.88         22.94
3      0.01056  0.007483     0.01718     0.002198        12.41         26.44
  perimeter_worst area_worst smoothness_worst compactness_worst concavity_worst
1           86.97      549.1           0.1385            0.1266          0.1242
2           78.28      424.8           0.1213            0.2515          0.1916
3           79.93      471.4           0.1369            0.1482          0.1067
  points_worst symmetry_worst dimension_worst
1      0.09391         0.2827         0.06771
2      0.07926         0.2940         0.07587
3      0.07431         0.2998         0.07881
```

A continuación, vamos a crear la red neuronal. Para ello, añadimos el paquete para Redes Neuronales Artificiales (ANN), llamado *neuralnet*:

```
> if (!require(neuralnet)){
+         install.packages("neuralnet")
+         require(neuralnet)
+ }
```

A continuación, realizamos el **proceso de entrenamiento de la red**, mediante la función *neuralnet*:

- **Fórmula**: en nuestro caso será *diagnosis = resto_de_campos.*
- **Datos de entrenamiento**
- **Número de neuronas en la capa oculta** (*size*): en nuestro caso, la establecemos a 10.

```
> nn <- neuralnet(diagnosis ~., data = breast_cancer.Train, hidden = c(10), linear.output = FALSE, threshold = 0.01)
```

Una vez entrenada la red neuronal, vamos a probarla con los datos de validación. Para ello, creamos un nuevo *dataframe* en el que eliminamos la columna *diagnosis*:

```
> breast_cancer.Test2 <- breast_cancer.Val[-1]
```

Una vez eliminado, realizamos la predicción de los datos de validación, utilizando para ello la función *compute*, indicando como parámetros la red neuronal anteriormente entrenada, así como los datos de validación:

```
> nn.results = compute(nn, breast_cancer.Test2)
```

Una vez realizada la predicción, creamos una **matriz de confusión** con la que podremos analizar la precisión de la red neuronal:

```
> results <- data.frame(actual = as.factor(breast_cancer.Val$diagnosis), prediction = nn.results$net.result)
> #Visualizamos las 10 primeras filas
> results[1:10,]
   actual prediction.1 prediction.2
1       B 9.998984e-01 9.187097e-05
2       B 1.000000e+00 2.724018e-33
3       B 1.000000e+00 2.724018e-33
4       B 1.000000e+00 2.724018e-33
5       B 9.998892e-01 9.892338e-05
6       B 1.000000e+00 2.724018e-33
7       B 1.000000e+00 2.129578e-33
8       M 7.040182e-51 1.000000e+00
9       B 1.000000e+00 2.724018e-33
10      B 1.000000e+00 2.724018e-33
```

Analizando la **matriz de confusión**, para el paciente 1 (al que se le ha diagnosticado un tumor benigno), la red neuronal predeciría un tumor de tipo benigno en el 100 % de las ocasiones, mientras que predeciría un tumor de tipo maligno en el 8.026981e-12 de las ocasiones, es decir, prácticamente acertaría con el diagnóstico. Por otro lado, para el paciente 8 (al que se le ha diagnosticado un tumor maligno), la red neuronal predeciría un tumor de tipo benigno en el 5.343009e-09 de las ocasiones, mientras que para el diagnóstico de un tumor maligno acertaría al 100 % prácticamente.