Fundamentos de la Ciencia de Datos. Prueba de Laboratorio 2

Ana Cortés Cercadillo[1], Carlos Javier Hellín Asensio[2], Daniel Ferreiro Rodríguez[3], and Francisco Calles Esteban[4]

University of Alcalá, Ctra. Madrid-Barcelona km 33
28805 Alcalá de Henares, Madrid, Spain

[1]a.cortesc@edu.uah.es, [2]carlos.hellin@edu.uah.es, [3]daniel.ferreiror@edu.uah.es, [4]francisco.calles@edu.uah.es

January 11, 2022

Abstract

En este documento se encontrará un análisis de detección de datos anómalos y un análisis de clasificación supervisada y no supervisada, aplicando todos los conceptos teóricos vistos en cada lección. Primero se realizan estos análisis de forma guiada y finalmente, con los conceptos teóricos adquiridos, se plantean diferentes formas de afrontar los análisis, como programando los algoritmos y aportando mejoras a LearnClust.

Keywords: Ciencia de datos, Análisis, Outliers, Clasificación no Supervisada, Clasificación Supervisada, R, LearnClust

Contents

1 Introducción

El documento está dividido en *Primera parte* y *Segunda parte*.

En la *Primera parte* se recibió una guía tutorizada por parte del profesor para que viéramos cómo aplicar en *R* los conceptos teóricos vistos en clase. Nuestra labor consistió en explicar con lujos de detalles el funcionamiento de las funciones enseñadas y los pasos seguidos en los apartados de *Análisis de detección de datos anómalos* (Caja y Bigotes, Desviación Típica, Regresión), *Análisis de clasificación no supervisada* (K-means y Clustering Jerarquico Aglomerativo) y *Análisis de clasificación supervisada* (árboles de decisión y Regresión).

En la *Segunda parte* se ha optado por programar la inmensa mayoría de las funciones utilizadas y las que no, se han empleado de paquetes como: *cluster*, *rpart*, *stats* y *LearnClust*.

Se ha probado todas y cada una de las funciones del paquete *LearnClust* detectando pequeños errores y se han planteado pequeñas mejoras detalladas en el apartado *Clustering Jerarquico Aglomerativo*.

2 Primera parte

2.1 Análisis de detección de datos anómalos

2.1.1 Caja y Bigotes

Se empieza con el análisis de detección de datos anómalos y en este caso concreto con el método de Caja y Bigotes para realizar el análisis de outliers.

Se introduce los datos en R mediante una matriz cuyo nombre de variable será *muestra* con los datos del enunciado.

```
(muestra=t(matrix(c(3,2,3.5,12,4.7,4.1,5.2,4.9,7.1,6.1,6.2,5.2,14,5.3),
2,7,dimnames=list(c("r","d")))))
```

```
      r     d
[1,]  3.0  2.0
[2,]  3.5 12.0
[3,]  4.7  4.1
[4,]  5.2  4.9
[5,]  7.1  6.1
[6,]  6.2  5.2
[7,] 14.0  5.3
```

Lo que hace esta instrucción es darle al valor muestra la transpuesta de la matriz con la función *t()* y se utiliza *matrix()* (que viene en el paquete base[4], por lo tanto no es necesario cargar nada, a diferencia, por ejemplo, de usar *Matrix()*[2]) para definir en su interior la propia matriz de dos columnas y siete filas. Con *dirnames* se le da los nombres que, para la resistencia es *r* y la densidad es *d*. Además, una novedad es el uso de paréntesis al principio y al final, con ello se obtiene el resultado a la vez que se guarda en la variable como se ve puede comprobar anteriormente.

Ahora se va a convertir la muestra de una matriz a un *data.frame*.

```
(muestra=data.frame(muestra))
     r    d
1  3.0  2.0
2  3.5 12.0
3  4.7  4.1
4  5.2  4.9
5  7.1  6.1
6  6.2  5.2
7 14.0  5.3
```

Aunque es parecida con la matriz, la diferencia fundamental es que las columnas tienen que ser todas del mismo tipo de datos, mientras que si hay varias características y con distintos tipos de variables (cualitativa y cuantitativa) es mejor utilizar un *data.frame*.

Se calculan ahora los outliers usando la función *boxplot()*:

```
(boxplot(muestra$r, range=1.5, plot=FALSE))
$stats
     [,1]
[1,] 3.00
[2,] 4.10
[3,] 5.20
[4,] 6.65
[5,] 7.10

$n
[1] 7

$conf
         [,1]
[1,] 3.677181
[2,] 6.722819

$out
[1] 14

$group
[1] 1

$names
[1] "1"
```

boxplot()[5] es una función pensada para dibujar una gráfica, pero en este caso no se hace pasando el argumento $plot = FALSE$ porque solo se busca la información que permite construir el diagrama de Caja y Bigotes. Se hará el análisis con la resistencia, por lo tanto, se pasa *muestra$r*. Por último, en el segundo atributo está el $range = 1.5$ que es el grado de outlier siendo el mismo que se ha visto en clase de teoría. Al observar los resultados se encuentra lo siguiente:

- *$stats*: son valores estadísticos relacionado con lo que se está haciendo. Primero es el valor mínimo (3.00), luego el valor del primer cuartil (4.10), del segundo cuartil (5.20), tercer cuartil (6.65) y máximo valor en la variable resistencia (7.10).
- *$n*: es el número de datos (7).
- *$conf*: los límites del intervalo de confianza que obtiene R si utiliza la función *boxplot()*. (entre 3.67 y 6.72)
- *$out*: el valor que considera outlier y está fuera de rango. (14)

Hasta ahora todo se ha hecho diferente a como se hace en clase de teoría, pero ahora se explica de esa forma, usando primero los cuartiles.

Se calcula el primer cuartil usando la función *quantile()* como ya se vio en la PL1:

```
(cuar1r=quantile(muestra$r,0.25))
25%
4.1
```

Y el tercer cuartil:

```
(cuar3r=quantile(muestra$r,0.75))
 75%
6.65
```

A continuación se calcula el intervalo de valores con la siguiente ecuación vista en teoría.

$$(Q_1 - d(Q_3 - Q_1), Q_3 + d(Q_3 - Q_1))$$

```
(int=c(cuar1r-1.5*(cuar3r-cuar1r), cuar3r+1.5*(cuar3r-cuar1r)))
   25%    75%
 0.275 10.475
```

Sale de 0.27 a 10.47 y es diferente a lo visto en teoría porque se utiliza la siguiente ecuación para calcular los cuartiles siendo distinta a lo que se usa en R.

- Si $nc \notin \mathbb{N} : \tilde{x}_c = x_{[nc+1]}$ [nc] parte entera de nc
- Si $nc \in \mathbb{N} : \tilde{x}_c = \frac{x_{nc}+x_{nc+1}}{2}$

Viendo los datos del enunciado, se ve que el valor 14 es el que está fuera del rango, por lo tanto es el outlier.

```
(3,2,3.5,12,4.7,4.1,5.2,4.9,7.1,6.1,6.2,5.2,14,5.3)
```

Pero esta comprobación se puede hacer con pocos datos, si tenemos más datos lo mejor es automatizarlo para saber qué valores están dentro del intervalo de valores y para eso hay que ir dato a dato. Esto se hace con un bucle y un *if*:

```
for(i in 1:length(muestra$r))
{if(muestra$r[i]<int[1] || muestra$r[i]>int[2])
{print("el suceso");print(i);print(muestra$r[i]);
print("es un suceso anómalo o outlier")}}
[1] "el suceso"
[1] 7
[1] 14
[1] "es un suceso anómalo o outlier"
```

El bucle empieza con $i = 1$ hasta la longitud de la muestra y en su interior el *if* para comprobar si el valor i de la muestra es menor que el valor inferior del intervalo o es mayor que el valor superior, entonces es un outlier.

2.1.2 Desviación Típica

Para realizar el análisis de outliers mediante desviación típica no va a ser necesario instalar y cargar ningún paquete en R. Siguiendo con la muestra del anterior análisis, se ejecuta la siguiente instrucción para calcular el intervalo de valores con densidad:

```
(int=c(mean(muestra$d) - 2*sd(muestra$d),
mean(muestra$d) + 2*sd(muestra$d)))
[1] -0.5146825 11.8289682
```

El intervalo que sale es de -0.51 a 11.82 que no es el mismo que daba en teoría. Esto es porque la desviación típica en R lo hace con el denominador $n-1$ y no n.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

Para corregir esto hay que obtener la desviación típica de la siguiente manera:

```
sdd = sqrt(var(muestra$d)*((length(muestra$d)-1)/length(muestra$d)))
sdd
[1] 2.857
```

Lo que hace es la raíz cuadrada de la varianza de la densidad y se multiplica por $n-1$ dividido por n.

Una vez con esto, ya se puede corregir el intervalo usando el *sdd* calculado.

```
(int=c(mean(muestra$d) - 2*sdd, mean(muestra$d) + 2*sdd))
[1] -0.05685714 11.37114285
```

Que es la misma ecuación que se utiliza en teoría.

$$(\bar{x}_a - dS_a, \bar{x}_a + dS_a)$$

Ahora se puede comprobar como antes, qué valores son outliers, teniendo en cuenta que ahora son los valores de densidad de la muestra.

```
for(i in 1:length(muestra$d))
{if(muestra$d[i]<int[1] || muestra$d[i]>int[2])
{print("el suceso");print(i);print(muestra$d[i]);
print("es un suceso anómalo o outlier")}}
[1] "el suceso"
[1] 2
[1] 12
[1] "es un suceso anómalo o outlier"
```

2.1.3 Regresión

La regresión se va a calcular utilizando un *script*. En el RGui se va a Archivo → Nuevo Script tal como se muestra en la Figura 1.

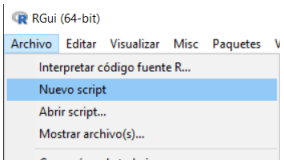


Figure 1: Creación de un nuevo script.

Y se abrirá la siguiente ventana, que se trata del editor de *scripts*.

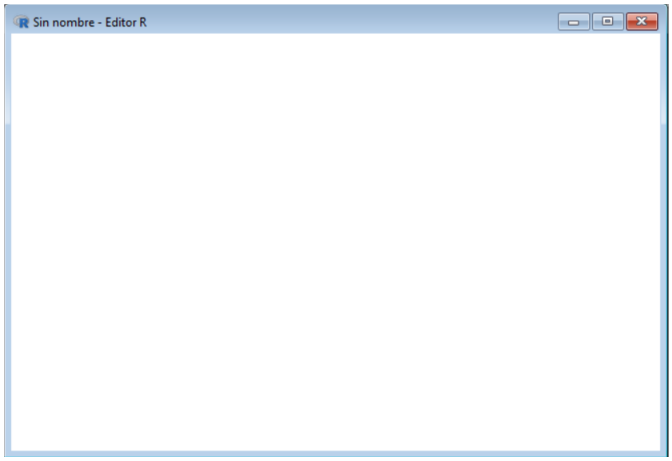


Figure 2: Editor de scripts.

Será necesario obtener la regresión de la densidad en función de la resistencia y, para ello, se pone todas las instrucciones en el *script* de forma que quede como en la Figura 3 y luego se explicarán una a una.

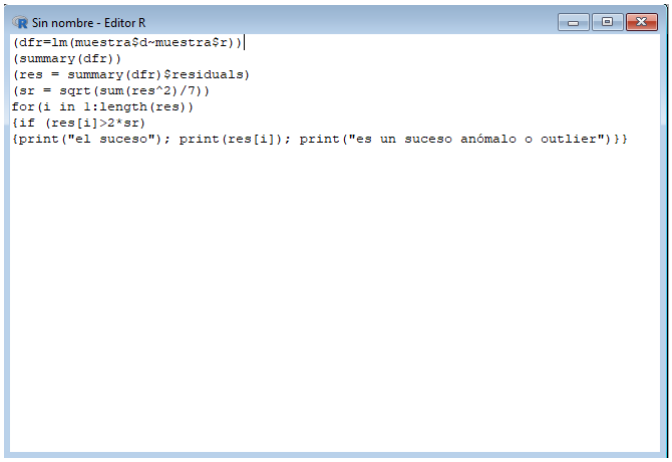


Figure 3: Todas las instrucciones en el script.

Colocando el puntero del ratón en la primera línea, se ejecuta en "Editar → Correr línea o seleccionar" para ir haciendo la ejecución de línea a línea.

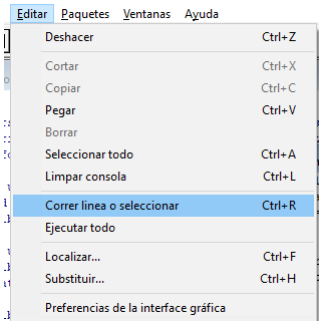


Figure 4: Ejecutar línea a línea en un script.

Usando la función *lm()*[6] para almacenar el resultado en la variable *dfr* (Densidad en Función de Resistencia), por lo tanto se usa la densidad como la variable de salida y la resistencia la de entrada. Así que se pone la muestra de d y r con el símbolo ˜ entre ellas.

```
(dfr=lm(muestra$d~muestra$r))

Call:
lm(formula = muestra$d ~ muestra$r)

Coefficients:
(Intercept)    muestra$r
    6.01445     -0.05723
```

Y lo que sale como resultado, es la ecuación de la recta que tiene una variable $a = 6.01$ y una variable $b = -0.057$.

$$a = \bar{y} - b\bar{x}\ ; b = \frac{S_{xy}}{{S_x}^2}$$

$$y_{ci} = 6.01 - 0.057x_i$$

Para obtener los residuos se usa la función *summary()*[4], que es la parte que interesa conocer.

```
(summary(dfr))

Call:
lm(formula = muestra$d ~ muestra$r)

Residuals:
      1       2       3       4       5       6       7
-3.84275  6.18587 -1.64545 -0.81683  0.49192 -0.45960  0.08684

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.01445    2.64632   2.273   0.0722 .
muestra$r   -0.05723    0.37148  -0.154   0.8836
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.372 on 5 degrees of freedom
Multiple R-squared:  0.004725,  Adjusted R-squared:  -0.1943
F-statistic: 0.02374 on 1 and 5 DF,  p-value: 0.8836
```

Esta información de los residuos es exactamente la diferencia que hay entre una densidad observada y una densidad calculada a través de la ecuación. Cada resistencia será multiplicada por -0.057 y se le sumará 6.01 y saldrá la densidad correspondiente a la primera resistencia.

$$y_i - y_{ci}$$

Para extraer el vector de los residuos se usa la siguiente instrucción:

```
(res = summary(dfr)$residuals)
         1          2          3          4          5          6
-3.8427477  6.1858698 -1.6454482 -0.8168308  0.4919157 -0.4595958
         7
 0.0868370
```

De esta forma, de toda la información disponible se obtiene solo los residuos y se almacena en la variable *res*, ya que R permite extraer información de las salidas complejas y asignarla a una variable.

Ahora en la siguiente instrucción lo que hace es sumar los residuos al cuadrado con la variable *res* que se ha obtenido en la anterior instrucción, y se divide por el número de datos para todo ello hacer la raíz cuadrada.

```
(sr = sqrt(sum(res^2)/7))
[1] 2.850242
```

Se hace igual que la ecuación vista en teoría.

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

Para saber qué valores son outliers, en este caso de la regresión, no hay un intervalo sino que aquellos valores cuyo residuo asociado sea mayor que el grado outlier por el error estándar de los residuos, entonces el suceso se considera un outlier.

$$|y_i - y_{ci}| > dS_r$$

Y de forma automatizada se obtiene los outliers con el *for* y el *if*.

```
for(i in 1:length(res))
{if (res[i]>2*sr)
{print("el suceso"); print(res[i]);
print("es un suceso anómalo o outlier")}}
[1] "el suceso"
     2
6.18587
[1] "es un suceso anómalo o outlier"
```

2.1.4 K-vecinos (*k-nearest neighbors*)

Los datos que se introducirán a continuación pertenecen a 5 calificaciones de estudiantes:

1. 4, 4; 2. 4, 3; 3. 5, 5; 4. 1, 1; 5. 5, 4

Donde las características de las calificaciones son: Teoría, Laboratorio) y estas se introducirán en una matriz (2x5).

```
> (muestra=matrix(c(4,4,4,3,5,5,1,1,5,4),2,5))
     [,1] [,2] [,3] [,4] [,5]
[1,]    4    4    5    1    5
[2,]    4    3    5    1    4
```

Para poder calcular distancias se tendrá que transponer la matriz y para ello se ha utilizado la función *t()*.

```
> (muestra=t(muestra))
     [,1] [,2]
[1,]    4    4
[2,]    4    3
[3,]    5    5
[4,]    1    1
[5,]    5    4
```

El grado de outliers será 2.5 y la $K = 3$, los valores empleados en teoría. La función *dist()*[6] calcula las distancias euclideas entre todos los puntos.

```
> dist(muestra)
         1        2        3        4
2 1.000000
3 1.414214 2.236068
4 4.242641 3.605551 5.656854
5 1.000000 1.414214 1.000000 5.000000
```

La matriz triangular obtenida en el apartado anterior no vale porque tiene algunas posiciones que están sin rellenar. La función *as.matrix()*[4] convierta la matriz triangular que se trabajando en una matriz completa de (5x5).

```
> (distancias=as.matrix(dist(muestra)))
         1        2        3        4        5
1 0.000000 1.000000 1.414214 4.242641 1.000000
2 1.000000 0.000000 2.236068 3.605551 1.414214
3 1.414214 2.236068 0.000000 5.656854 1.000000
4 4.242641 3.605551 5.656854 0.000000 5.000000
5 1.000000 1.414214 1.000000 5.000000 0.000000
```

La matriz distancias, a veces, puede no funcionar y para asegurarse que todo vaya bien se definirá la dimensión de matriz (5x5).

Si se fija en la columna 1, por ejemplo, los valores no están desordenados. Para conseguir los K-vecinos más próximos a un punto, las distancias a los demás puntos estarán ordenadas de menor a mayor.

```
> (distancias=matrix(distancias,5,5))
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.000000 1.000000 1.414214 4.242641 1.000000
[2,] 1.000000 0.000000 2.236068 3.605551 1.414214
[3,] 1.414214 2.236068 0.000000 5.656854 1.000000
[4,] 4.242641 3.605551 5.656854 0.000000 5.000000
[5,] 1.000000 1.414214 1.000000 5.000000 0.000000
```

La siguiente línea consigue ordenar, con la función *sort()*[4], los valores de las columnas para conseguir los K-vecinos más próximos a los puntos que enumeran las columnas. Las columnas ordenas estarán contenidas en la matriz *distanciasordenadas*.

Recordatorio: El *;* se emplea para separar dos líneas que van hacer ejecutadas de forma secuencial.

```
> for(i in 1:5){distancias[,i]=sort(distancias[,i])};
(distanciasordenadas=distancias)
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.000000 0.000000 0.000000 0.000000 0.000000
[2,] 1.000000 1.000000 1.000000 3.605551 1.000000
[3,] 1.000000 1.414214 1.414214 4.242641 1.000000
[4,] 1.414214 2.236068 2.236068 5.000000 1.414214
[5,] 4.242641 3.605551 5.656854 5.656854 5.000000
```

Se comprueba si hay algún valor outlier en la matriz *distanciasordenadas* mirando los K-vecinos más próximos. Todos los valores de la primera fila de la matriz *distanciasordenadas* son ceros porque expresan las distancias de estos a sí mismos. El nuevo K será $K + 1$, en este caso $K = 4$ $(3 + 1)$, para ignorar la primera fila.

El siguiente código recorre la matriz *distanciasordenadas* mostrando el/los valor/es y el mensaje "es un suceso anómalo o outlier" si el $K + 1$ vecino más próximo es/son mayor/es que el grado de outlier (2.5).

```
> for(i in 1:5){if(distanciasordenadas[4,i]>2.5){print(i);
print("es un suceso anómalo o outlier")}}
[1] 4
[1] "es un suceso anómalo o outlier"
```

En el código siguiente se ordena las columnas de menor a mayor de la matriz *distancias* y se determina el/los valor/es outlier/s que tiene la matriz. Se gana en optimización al fusionar el código de los dos últimos apartados. El suceso anómalo de esta matriz para un $d = 2.5$ es el valor 4.

```
> for(i in 1:5){distancias[,i]=sort(distancias[,i])
+ if (distancias[4,i]>2.5){print(i)
+ print("es un suceso anómalo o outlier")}}
[1] 4
[1] "es un suceso anómalo o outlier"
```

La función *dist()* calcula por defecto las distancias euclideas, pero se requiere para este caso que sean calculadas por el método *Manhattan*.

Longitud Manhattan: La distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas.

La matriz distanciasM tiene las distancias entre los puntos de la muestra utilizando el método *Manhattan*. La función *dist* recibe como parámetro el método *manhattan* el que calculará las distancias entre los valores almacenados en la variable *muestra*. A continuación, se observa que la diagonal de la matriz es de ceros porque es la distancia de cada punto a si mismo.

Comprobación del resultado de la matriz tomando como ejemplo el cálculo de las distancias entre el p1 con el p2 y el p4 con el p5. Distancia del p1 (4,4) con el p2 (4,3) es igual a 1 porque $|4 - 4| + |4 - 3| = 1$. Distancia del p4 (1,1) con el p5 (5,4) es igual a 7 porque $|1 - 5| + |1 - 4| = 7$.

```
> (distanciasM=as.matrix(dist(muestra,method="manhattan")))
  1 2 3 4 5
1 0 1 2 6 1
2 1 0 3 5 2
3 2 3 0 8 1
4 6 5 8 0 7
5 1 2 1 7 0
```

Para calcular la N (factor más diferencial) se tiene que contar tantos vecinos más próximos como valor de la K haya (mayor distancia de los puntos más próximos). Hasta este momento $N = K$. A continuación, se debe comprobar si las siguientes distancias más próximas son iguales o no a la mayor distancia determinada antes. Posibilidades:

- Igual: se le suma 1 al valor actual de la N, $N = N + 1$, y se sigue comprobando las siguientes distancias más próximas.
- Distinto: no hace falta seguir comprobando las siguientes distancias más próximas y el valor de la N es el actual.

Ejemplo: Si se tuviera que $K = 2$, el vecino más próximo está a una distancia 1 y el segundo más próximo a una distancia de 5. La mayor distancia es 5 y el valor

actual de la $N = 2$. Si la distancia del siguiente vecino es 5, se hace un $N + 1$ entonces $N = 3$. Si la distancia del posterior más próximo es 6, no se sigue comprobando y $N = 3$.

2.2 Análisis de clasificación no supervisada

2.2.1 K-medias (*K-means*)

En este apartado se va a realizar un análisis de clasificación no supervisada con una muestra de ejemplo utilizando la técnica de clusterización de *K-medias* o *K-means*.

Para el análisis se utiliza un conjunto de datos formado por ocho calificaciones separadas en notas de teoría y laboratorio cuyo rango se encuentra entre 0 y 5. Este conjunto se introduce en R guardándolo en una matriz de dos filas y ocho columnas mediante el uso de la función *matrix()* de la siguiente manera:

```
> (m<-matrix(c(4,4,3,5,1,2,5,5,0,1,2,2,4,5,2,1),2,8))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    4    3    1    5    0    2    4    2
[2,]    4    5    2    5    1    2    5    1
```

Dicha función como se explica anteriormente viene en el paquete base por lo que no es necesario cargar ningún paquete.

A la matriz de los datos se la transpone utilizando la función *t()*.

```
> (m<-t(m))
     [,1] [,2]
[1,]    4    4
[2,]    3    5
[3,]    1    2
[4,]    5    5
[5,]    0    1
[6,]    2    2
[7,]    4    5
[8,]    2    1
```

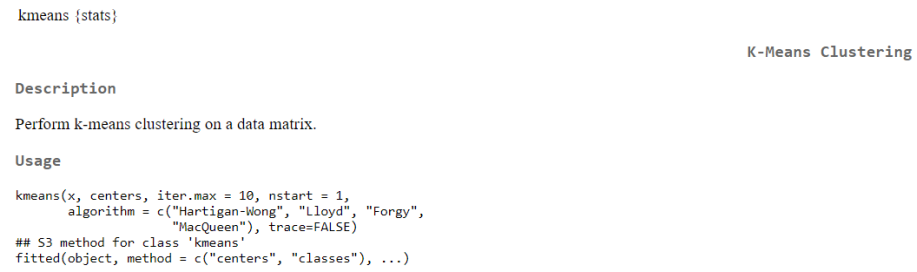
A continuación se toman los centroides como los puntos medios del grupo de sucesos que componen el cluster. En este caso se utilizan dos clusters por lo que se establece C1(0,1) y C2(2,2) y se guardan en una matriz.

```
> (c<-t(matrix(c(0,1,2,2),2,2)))
     [,1] [,2]
[1,]    0    1
[2,]    2    2
```

Tras estructurar el conjunto de datos y definir los centroides iniciales se procede a realizar el algoritmo *k-means*. Para ello R dispone en el paquete *stats*[6] que viene cargado por defecto de la función *kmeans()* tal como se puede observar en la documentación de ésta (Figura 5) después de ejecutar el comando *help(kmeans)*.

Esta función lo que hace es realizar una clusterización (agrupamiento) en una matriz de datos pasada como argumento utilizando la técnica de k-medias. Otros argumentos de los que dispone la función son el número o matriz de centroides *centers*, el número máximo de iteraciones que se van a realizar *iter.max*, el algoritmo a utilizar siendo el de *Hartigan-Wong* por defecto *algorithm*, o el método *method*.

En este caso se aplicará el k-means al conjunto de de datos de las calificaciones, con la matriz de los centroides iniciales y con cuatro iteraciones como máximo.



kmeans {stats}

K-Means Clustering

Description

Perform k-means clustering on a data matrix.

Usage

```
kmeans(x, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                     "MacQueen"), trace=FALSE)
## S3 method for class 'kmeans'
fitted(object, method = c("centers", "classes"), ...)
```

Figure 5: Función *kmeans()* en el paquete *stats*.

```
> (clasificacionns=kmeans(m,c,4))
K-means clustering with 2 clusters of sizes 4, 4

Cluster means:
  [,1] [,2]
1 1.25 1.50
2 4.00 4.75

Clustering vector:
[1] 2 2 1 2 1 1 2 1

Within cluster sum of squares by cluster:
[1] 3.75 2.75
 (between_SS / total_SS =  84.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss"      "betweenss"    "size"         "iter"
[9] "ifault"
```

El valor de los centroides finales de los clusters corresponde al dado en *Cluster means*, éste es C1(1.25, 1.5) y C2(4, 4.75). Así como el vector de agrupamiento resultante *Clustering vector* que indica los puntos asignados al centroide más cercano de los clusters siendo "1" al primer cluster y "2" al segundo.

A continuación se unen el vector agrupado y la matriz de muestra inicial mediante la función *cbind()*[4]. Esta función pertenece al paquete base de R y lo que hace es tomar una secuencia de vectores, matrices o *data frames* y los combina por columnas o filas, respectivamente. La matriz combinada será la siguiente.

```
> (m=cbind(clasificacionns$cluster, m))
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    2    3    5
[3,]    1    1    2
[4,]    2    5    5
[5,]    1    0    1
[6,]    1    2    2
[7,]    2    4    5
[8,]    1    2    1
```

Después de esa matriz se obtiene el subconjunto formado por los datos cuyo vector agrupado (primera columna) valga "1" y se guarda en una variable.

Para ello se utiliza la función *subset()*[4] del paquete base que lo que hace es devolver un subconjunto de vectores, matrices o *data frames* que cumplen una condición dada. El primer argumento es el objeto del que se va a extraer un subconjunto y el segundo la expresión lógica de la condición que se tiene que cumplir para definir el subconjunto.

En este caso el objeto es es la matriz combinada y la condición que los elementos de la primera columna valgan "1".

```
> (mc1=subset(m,m[,1]==1))
     [,1] [,2] [,3]
[1,]    1    1    2
[2,]    1    0    1
[3,]    1    2    2
[4,]    1    2    1
```

Se hace lo mismo para los datos en los que el vector agrupado valga "2" guardándolo en otra variable.

```
> (mc2=subset(m,m[,1]==2))
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    2    3    5
[3,]    2    5    5
[4,]    2    4    5
```

Por último se extraen los datos de los dos clusters o agrupaciones eliminando la columna del vector agrupado (la primera).

Datos del primer cluster:

```
> (mc1=mc1[,-1])
     [,1] [,2]
[1,]    1    2
[2,]    0    1
[3,]    2    2
[4,]    2    1
```

Datos del segundo cluster:

```
> (mc2=mc2[,-1])
     [,1] [,2]
[1,]    4    4
[2,]    3    5
[3,]    5    5
[4,]    4    5
```

2.2.2 Clustering jerárquico aglomerativo

Para trabajar con la clusterización jerárquica aglomerativa en R se va a usar el paquete *LearnClust*[1]. Este paquete está desarrollado por un antiguo alumno de la Universidad de Alcalá.

Primero se realizará la instalación del paquete de forma local para conocer más en profundidad todo lo relacionado con el mismo, por lo tanto, hay que acceder a la página web del CRAN (`https://cran.r-project.org/`).

En la anterior práctica ya se usó esta página web por lo que ya resulta más familiar. Se accede a la sección *Packages* del menú situado a la izquierda de la pantalla (Figura 6). Ahí se busca el paquete en la tabla de paquetes listados por nombre (Figura 7). Una vez encontrado, se pulsa en el nombre del paquete para entrar en la página web específica del paquete (Figura 8).

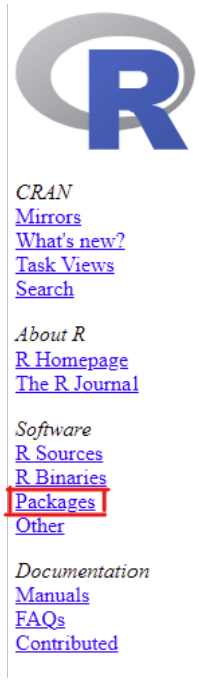


Figure 6: Menú CRAN.

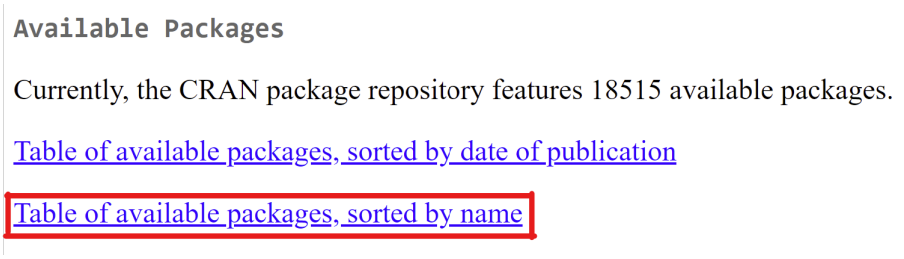


Figure 7: Acceder al listado de paquetes.

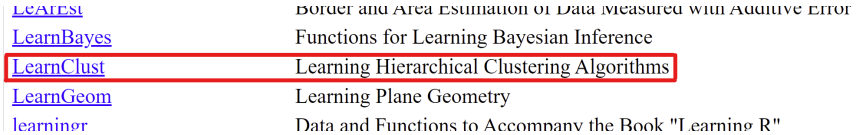
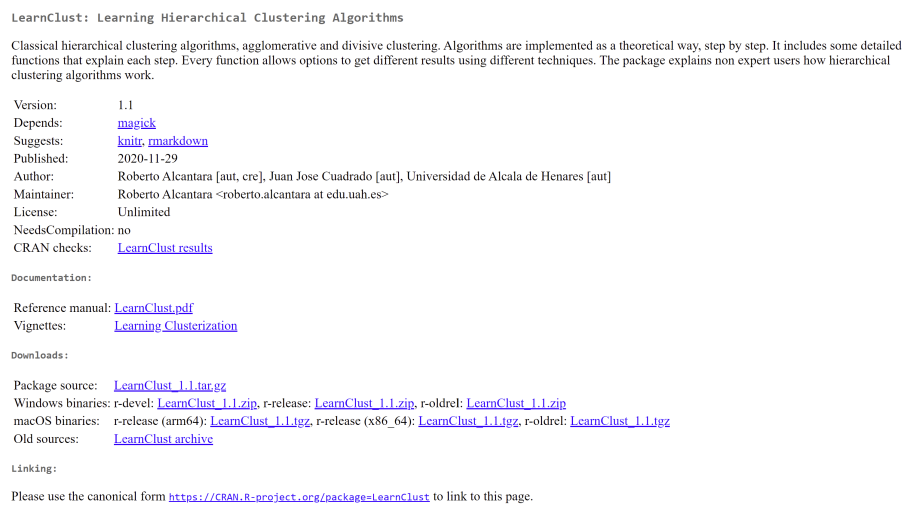


Figure 8: Paquete *LearnClust*.

Cuando hay que analizar, ver con detalle y conocer un paquete en profundidad lo que se debe hacer es ir a la página del paquete para ver todas sus características (Figura 9).



LearnClust: Learning Hierarchical Clustering Algorithms

Classical hierarchical clustering algorithms, agglomerative and divisive clustering. Algorithms are implemented as a theoretical way, step by step. It includes some detailed functions that explain each step. Every function allows options to get different results using different techniques. The package explains non expert users how hierarchical clustering algorithms work.

Version:	1.1
Depends:	magick
Suggests:	knitr, rmarkdown
Published:	2020-11-29
Author:	Roberto Alcantara [aut, cre], Juan Jose Cuadrado [aut], Universidad de Alcala de Henares [aut]
Maintainer:	Roberto Alcantara <roberto.alcantara at edu.uah.es>
License:	Unlimited
NeedsCompilation:	no
CRAN checks:	LearnClust results

Documentation:

Reference manual:	LearnClust.pdf
Vignettes:	Learning Clusterization

Downloads:

Package source:	LearnClust_1.1.tar.gz
Windows binaries:	r-devel: LearnClust_1.1.zip, r-release: LearnClust_1.1.zip, r-oldrel: LearnClust_1.1.zip
macOS binaries:	r-release (arm64): LearnClust_1.1.tgz, r-release (x86_64): LearnClust_1.1.tgz, r-oldrel: LearnClust_1.1.tgz
Old sources:	LearnClust archive

Linking:

Please use the canonical form https://CRAN.R-project.org/package=LearnClust to link to this page.

Figure 9: Página web del paquete *LearnClust*.

Primero hay que ver el título, en este caso, se llama *LearnClust* con la extensión *Learning Hierarchical Clustering Algorithms*, que lo que quiere decir es que permite el aprendizaje de la clusterización jerárquica aglomerativa.

Luego hay una descripción del paquete que dice que lo que va hacer es la aplicación de algoritmos clásicos de clasificación jerárquica y que los algoritmos estarán implementados de forma teórica paso por paso, lo que incluye las funciones detalladas de cada paso. Cada función tiene diferentes opciones que permiten aplicar diferentes técnicas, es decir, que permitirá utilizar diferentes definiciones de la medida en las distancias (euclídea, manhattan, etc.) y permite también utilizar diferentes algoritmos de proximidad (mínimo, máximo, average). Además el paquete explica a los usuarios no expertos como el algoritmo de clusterización funciona, por lo tanto, no solamente es que va a clusterizar los datos, si no que va a explicar como funciona el algoritmo.

A continuación están las características del paquete, primero la versión que es la 1.1, por lo que tiene una corrección de la versión 1; las dependencias, que depende del paquete *magick*, lo cual quiere decir que hay que tener cargado este paquete para que pueda funcionar el paquete *LearnClust*; los paquetes sugeridos, que son el *knitr* y el *rmarkdown*; en que fecha se publicó, es la fecha de la última publicación; el autor y la persona que se ocupa de mantener el paquete; la licencia es ilimitada y no necesita compilación. Lo siguiente son las pruebas del paquete y la documentación.

Hay dos tipos de documentación:

1. **Manual de referencia:** explica qué funciones tiene el paquete y cómo operan, qué funcionalidad tienen cada una de ellas. En el caso del paquete *LearnClust* es un pdf.
2. **Viñetas:** pequeño curso que ayuda a introducirse a cómo funciona el paquete. En este caso es un html que luego se abrirá en el navegador.

Es recomendable descargarse los dos tipos de documentación para consultarlo de forma local y analizar correctamente el paquete.

Para finalizar está la parte de descargas donde se encuentra el código fuente del paquete con la extensión .tar.gz, muy útil para trabajar sobre él; los *binaries* de *Win-*

dows y de *mac* de los que hay que descargarse el de tipo *release* que corresponda con el sistema operativo que se esté usando y el archivo de las fuentes antiguas. También aparece cómo referenciar la página web en el apartado *Linking*.

Para instalar el paquete *LearnClust* desde local, se necesita descargar el *r-release* de los *binaries* del sistema operativo correspondiente y guardar el zip *LearnClust_1.1.zip* en un directorio. No hay que descomprimirlo. En R se usa la instrucción *setwd()* para cambiar el directorio de trabajo al directorio donde ha sido guardado el zip del paquete. También es útil el uso de *getwd()* para conocer la ruta del directorio de trabajo actual. Luego se usa la instrucción *install.packages()* con el nombre del zip para realizar la instalación en R.

```
> install.packages("LearnClust_1.1.zip")
Installing package into ‘C:/Users/atien/OneDrive - Universidad de
Alcala/Documents/R/win-library/4.1’
(as ‘lib’ is unspecified)
inferring ’repos = NULL’ from ’pkgs’
package ‘LearnClust’ successfully unpacked and MD5 sums checked
```

Ahora hay que cargarlo en el sistema. Para eso se introduce la instrucción *library()* y el nombre del paquete.

```
> library("LearnClust")
Error: package ‘magick’ required by ‘LearnClust’ could not be found
Además: Warning message:
package ‘LearnClust’ was built under R version 4.1.2
```

Sale este mensaje de error que dice que no está el paquete *magick* y que es requerido. Antes, cuando se ha visto en el CRAN la página del paquete decía que hay una dependencia con este paquete por lo que tiene que estar instalado. No lo ha instalado porque como la instalación que se está realizando es la manual, no lo ha instalado automáticamente. Si se opta por una instalación automática, se instalan también todas las dependencias. Como el paquete *magick* no interesa tanto como el paquete *LearnClust*, del que es interesante conocer y descargarse el manual, las viñetas y demás, se puede descargar de forma más automática sin descargarlo en local. Se instala sin problemas de la siguiente forma y se selecciona el espejo (*mirror*) que se desee.

```
> install.packages("magick")
Installing package into ‘C:/Users/atien/OneDrive - Universidad de
Alcala/Documents/R/win-library/4.1’
(as ‘lib’ is unspecified)
--- Please select a CRAN mirror for use in this session ---
probando la URL
’https://cran.rediris.es/bin/windows/contrib/4.1/magick_2.7.3.zip’
Content type ’application/zip’ length 33810271 bytes (32.2 MB)
downloaded 32.2 MB

package ‘magick’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\atien\AppData\Local\Temp\RtmpAprs7c\downloaded_packages
```

Ahora sí, se vuelve a usar la función *library()* y se carga el paquete sin problemas.

```
> library("LearnClust")
Loading required package: magick
Linking to ImageMagick 6.9.12.3
Enabled features: cairo, freetype, fftw, ghostscript, heic, lcms, pango,
```

```
raw, rsvg, webp
Disabled features: fontconfig, x11
Warning messages:
1: package 'LearnClust' was built under R version 4.1.2
2: package 'magick' was built under R version 4.1.2
```

Para comprobar que el paquete está cargado en la sesión de R en la que se ha realizado la instrucción anterior, se puede usar *search()*.

Una vez instalado y cargado el paquete, se hace uso de algunas de sus funciones. Primero se crea una matriz de 2 filas y 6 columnas con datos de 6 pares de valores y se guarda en una variable de nombre "m".

```
> m=matrix(c(0.89,2.94,4.36,5.21,3.75,1.12,6.25,3.14,4.1,1.8,3.9,4.27),
2,6)
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] 0.89 4.36 3.75 6.25  4.1 3.90
[2,] 2.94 5.21 1.12 3.14  1.8 4.27
```

Ahora hay que trasponer esta matriz para tenerla preparada para trabajar con ella. Para ello se usa la función *t()*.

```
> m=t(m)
> m
     [,1] [,2]
[1,] 0.89 2.94
[2,] 4.36 5.21
[3,] 3.75 1.12
[4,] 6.25 3.14
[5,] 4.10 1.80
[6,] 3.90 4.27
```

Es el momento de abrir el pdf del manual de *LearnClust* para conocer sus funciones. Primero se va a usar la función *agglomerativeHC*, por lo tanto, se busca en el pdf su información (Figura 10 y 11).

La función que se va a introducir es *agglomerativeHC()* que devolverá la clusterización jerárquica aglomerativa. Tiene tres atributos fundamentales:

1. Datos sobre los que se va a clusterizar que va a ser la matriz "m".
2. El tipo de distancia que se va a utilizar, en este caso la euclídea "EUC".
3. Proximidad que se va a usar, en este caso la definición de proximidad mínima "MIN".

La ejecución quedaría de la siguiente forma, la salida en la consola de R y la Figura 12:

```
> agglomerativeHC(m,'EUC','MIN')
$dendrogram
Number of objects: 6


$clusters
$clusters[[1]]
    X1   X2
1 0.89 2.94

$clusters[[2]]
    X1   X2
```

`agglomerativeHC`	*To execute agglomerative hierarchical clusterization algorithm by distance and approach.*

Description

To execute complete agglomerative hierarchical clusterization algorithm choosing distance and approach type.

Usage

```
agglomerativeHC(data, distance, approach)
```

Arguments

`data`	could be a numeric vector, a matrix or a numeric data frame. It will be transformed into matrix and list to be used.
`distance`	is a string. It chooses the distance to use.
`approach`	is a string. It chooses the approach to use.

Details

This function is the main part of the agglomerative hierarchical clusterization method. It executes the theoretical algorithm step by step.

1 - The function transforms data in useful object to be used.

2 - It creates the clusters.

3 - It calculates a matrix distances with the clusters created applying distance and approach given.

4 - It chooses the distance value and gets the clusters.

5 - It groups the clusters in a new one and updates clusters list.

6 - It repeats these steps until an unique cluster exists.

Figure 10: Función *agglomerativeHC*.

Value

R object with a dendrogram, the grouped clusters and the list with every cluster.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
a <- c(1,2,1,3,1,4,1,5,1,6)

matrixA <- matrix(a,ncol=2)

dataFrameA <- data.frame(matrixA)

agglomerativeHC(a,'EUC','MAX')

agglomerativeHC(matrixA,'MAN','AVG')

agglomerativeHC(dataFrameA,'CAN','MIN')
```

Figure 11: Función *agglomerativeHC*.

```
1 4.36 5.21

$clusters[[3]]
    X1   X2
1 3.75 1.12

$clusters[[4]]
    X1   X2
1 6.25 3.14

$clusters[[5]]
   X1  X2
1 4.1 1.8

$clusters[[6]]
   X1   X2
1 3.9 4.27

$clusters[[7]]
    X1   X2
1 3.75 1.12
2 4.10 1.80

$clusters[[8]]
    X1   X2
1 4.36 5.21
2 3.90 4.27

$clusters[[9]]
    X1   X2
1 3.75 1.12
2 4.10 1.80
3 4.36 5.21
4 3.90 4.27

$clusters[[10]]
    X1   X2
1 6.25 3.14
2 3.75 1.12
3 4.10 1.80
4 4.36 5.21
5 3.90 4.27

$clusters[[11]]
    X1   X2
1 0.89 2.94
2 6.25 3.14
3 3.75 1.12
4 4.10 1.80
5 4.36 5.21
6 3.90 4.27


$groupedClusters
```

```
  cluster1 cluster2
1        3        5
2        2        6
3        7        8
4        4        9
5        1       10
```

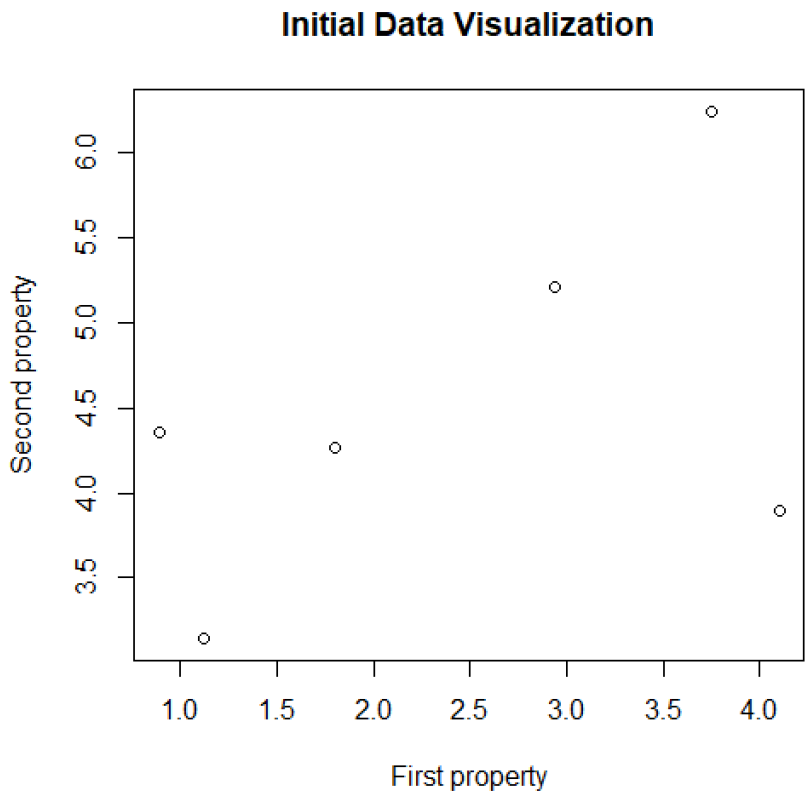


Figure 12: Gráfica de salida.

Al analizar esta salida, se observa que se han generado hasta 11 clusters. Los 6 primeros clusters que aparecen son los clusters originales, ya que en la primera iteración todos los puntos individuales se consideran clusters. Por lo tanto, lo que está haciendo la solución de la función del paquete es considerar cada punto un cluster y enumerarlos del 1 al 6. Por ejemplo, el cluster 6 es el punto (3.9, 4.27), el cluster 5 es el punto (4.1, 1.8) y así hasta el cluster 1 son todos los clusters, es decir, la primera iteración cada punto es un cluster.

El cluster número 7 es el primero que calcula el sistema que tiene como valores los puntos (3.75, 1.12) y (4.10, 1.80). Lo que se hace en este cluster es unir estos dos puntos, los correspondientes a los clusters 3 y 5 que es lo que corresponde en la resolución con la definición de proximidad "MIN".

En el cluster 8 ocurre lo mismo que en el anterior cluster. Se juntan los puntos (4.36, 5.21) y (3.90, 4.27), es decir, el cluster 2 con el cluster 6.

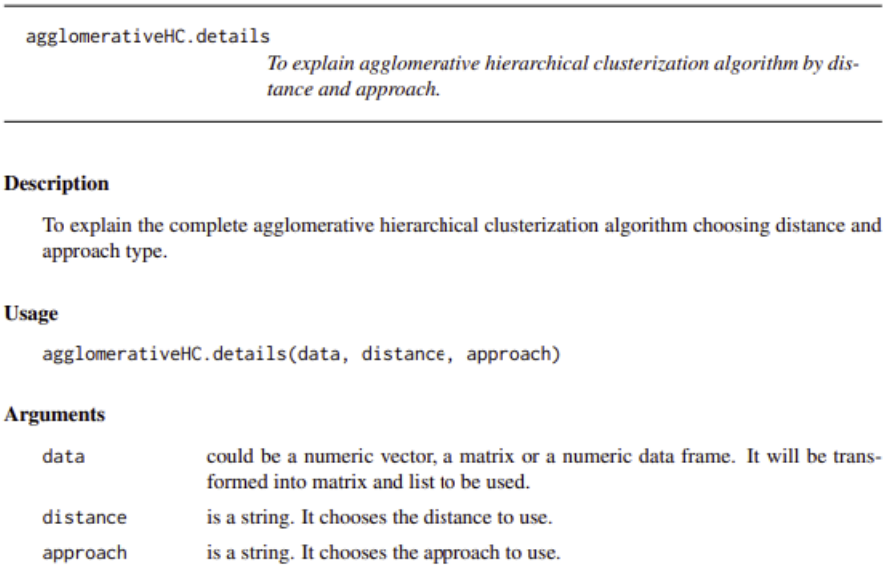
El cluster 9 lo que hace es unir el cluster 7 y el cluster 8, es decir, los puntos (3.75, 1.12), (4.10, 1.80), (4.36, 5.21) y (3.90, 4.27).

A continuación va el siguiente cluster, el número 10, y fijándose se ve como está el cluster 9, con sus puntos (3.75, 1.12), (4.10, 1.80), (4.36, 5.21) y (3.90, 4.27), y un primer punto al principio, el (6.25, 3.14), que es el cluster 4.

Y por último, el undécimo cluster une el cluster 1 al cluster 10, es decir, el cluster 1 a lo anterior y obtiene un único y último cluster.

A continuación de esto, está la última solución que se propone que son los *grouped-Clusters*. Aquí se hace un resumen de lo que ha ocurrido en las iteraciones realizadas diciendo que ha unido en cada una de ellas. Por ejemplo, la primera fila dice que en la iteración 1 se han unido los clusters 3 y 5, la siguiente se unen los clusters 2 y 6, luego el 7 y el 8, en la iteración 4 los clusters 4 y 9 y en la última iteración, la 5, se han unido los clusters 1 y 10. Esta parte dice como se ha realizado el algoritmo secuencialmente.

A partir de aquí, surge una reflexión que es inmediata. El paquete está claro que ha dado la solución; al menos para la definición de proximidad "MIN" ha dado la solución correcta, pero el paquete se llama *LearnClust*, aprender a clusterizar, y esta solución no enseña nada y es muy poco intuitiva. Devuelve el resultado pero no explica como se ha hecho, por lo que no se puede aprender a clusterizar jerárquicamente con esto. No se sabe de donde salen esos números ni cómo o por qué ha unido los clusters. Lo visto hasta ahora solo corresponde a la parte *Clust* del nombre del paquete. El paquete tiene que tener algo más que haga referencia a la parte de *Learn*. Para ello, se puede ver que en el pdf del manual existe la misma función que se acaba de emplear, *agglomerativeHC*, pero con una extensión, *.details* (Figura 13 y 14).



`agglomerativeHC.details`

To explain agglomerative hierarchical clusterization algorithm by distance and approach.

Description

To explain the complete agglomerative hierarchical clusterization algorithm choosing distance and approach type.

Usage

```
agglomerativeHC.details(data, distance, approach)
```

Arguments

`data`	could be a numeric vector, a matrix or a numeric data frame. It will be transformed into matrix and list to be used.
`distance`	is a string. It chooses the distance to use.
`approach`	is a string. It chooses the approach to use.

Figure 13: Función *agglomerativeHC.details*.

Lo que hace esta función es lo mismo que se acaba de ver pero dando más detalles de cómo está resolviendo lo que está resolviendo. Se introduce en la consola de R esta nueva función y se analiza su salida por consola y la Figura 12.

```
> agglomerativeHC.details(m,'EUC','MIN')
  Agglomerative hierarchical clustering is a classification technique
  that initializes
 a cluster for each data.
```

Details

This function is the main part of the agglomerative hierarchical clusterization method. It explains the theoretical algorithm step by step.

1 - The function transforms data into useful object to be used.

2 - It creates the clusters.

3 - It calculates a matrix distance with the clusters created by applying the distance and the approach given.

4 - It chooses the distance value and gets the clusters.

5 - It groups the clusters in a new one and updates clusters list.

6 - It repeats these steps until an unique cluster exists.

Value

agglomerative algorithm explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
a <- c(1,2,1,3,1,4,1,5,1,6)

matrixA <- matrix(a,ncol=2)

dataFrameA <- data.frame(matrixA)

agglomerativeHC.details(a,'EUC','MAX')

agglomerativeHC.details(matrixA,'MAN','AVG')

agglomerativeHC.details(dataFrameA,'CAN','MIN')
```

Figure 14: Función *agglomerativeHC.details*.

```
  It calculates the distance between datas depending on the approach type
  given and


 it creates a new cluster joining the most similar clusters until
 getting only one.

  'toList' creates a list initializing datas by creating clusters with
  each one


  These are the clusters with only one element:

[[1]]
     [,1] [,2] [,3]
[1,] 0.89 2.94    1

[[2]]
     [,1] [,2] [,3]
[1,] 4.36 5.21    1

[[3]]
     [,1] [,2] [,3]
[1,] 3.75 1.12    1

[[4]]
     [,1] [,2] [,3]
[1,] 6.25 3.14    1

[[5]]
     [,1] [,2] [,3]
[1,]  4.1  1.8    1

[[6]]
     [,1] [,2] [,3]
[1,]  3.9 4.27    1


  In each step:
   - It calculates a matrix distance between active clusters depending
   on the approach and distance type.

   - It gets the minimum distance value from the matrix.

   - It creates a new cluster joining the minimum distance clusters.

   - It repeats these steps while final clusters do not include all
   datas.


-----------------------------------------------------------------
```

```
STEP => 1


 Matrix Distance (distance type = EUC, approach type = MIN):

         [,1]     [,2]      [,3]     [,4]      [,5]     [,6]
[1,] 0.000000 4.146541 3.3899853 5.363730 3.4064204 3.290745
[2,] 4.146541 0.000000 4.1352388 2.803034 3.4198977 1.046518
[3,] 3.389985 4.135239 0.0000000 3.214094 0.7647876 3.153569
[4,] 5.363730 2.803034 3.2140940 0.000000 2.5333969 2.607566
[5,] 3.406420 3.419898 0.7647876 2.533397 0.0000000 2.478084
[6,] 3.290745 1.046518 3.1535694 2.607566 2.4780839 0.000000

 The minimum distance is: 0.764787552199956


 The closest clusters are: 3, 5


 The grouped clusters are added to the solution.


 Grouping clusters 3 and cluster 5, it is created a new cluster:

    X1   X2
1 3.75 1.12
2 4.10 1.80

 The new cluster is added to the solution.


----------------------------------------------------------------------

STEP => 2


 Matrix Distance (distance type = EUC, approach type = MIN):

         [,1]     [,2] [,3]     [,4] [,5]     [,6]     [,7]
[1,] 0.000000 4.146541    0 5.363730    0 3.290745 3.389985
[2,] 4.146541 0.000000    0 2.803034    0 1.046518 3.419898
[3,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[4,] 5.363730 2.803034    0 0.000000    0 2.607566 2.533397
[5,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[6,] 3.290745 1.046518    0 2.607566    0 0.000000 2.478084
[7,] 3.389985 3.419898    0 2.533397    0 2.478084 0.000000

 The minimum distance is: 1.04651803615609


 The closest clusters are: 2, 6


 The grouped clusters are added to the solution.
```

```
 Grouping clusters 2 and cluster 6, it is created a new cluster:

    X1   X2
1 4.36 5.21
2 3.90 4.27

 The new cluster is added to the solution.


-----------------------------------------------------------------

STEP => 3


 Matrix Distance (distance type = EUC, approach type = MIN):

         [,1] [,2] [,3]     [,4] [,5] [,6]     [,7]     [,8]
[1,] 0.000000    0    0 5.363730    0    0 3.389985 3.290745
[2,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[3,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[4,] 5.363730    0    0 0.000000    0    0 2.533397 2.607566
[5,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[6,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[7,] 3.389985    0    0 2.533397    0    0 0.000000 2.478084
[8,] 3.290745    0    0 2.607566    0    0 2.478084 0.000000

 The minimum distance is: 2.47808393723861


 The closest clusters are: 7, 8


 The grouped clusters are added to the solution.


 Grouping clusters 7 and cluster 8, it is created a new cluster:

    X1   X2
1 3.75 1.12
2 4.10 1.80
3 4.36 5.21
4 3.90 4.27

 The new cluster is added to the solution.


-----------------------------------------------------------------

STEP => 4


 Matrix Distance (distance type = EUC, approach type = MIN):
```

```
          [,1] [,2] [,3]     [,4] [,5] [,6] [,7] [,8]     [,9]
 [1,] 0.000000    0    0 5.363730    0    0    0    0 3.290745
 [2,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [3,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [4,] 5.363730    0    0 0.000000    0    0    0    0 2.533397
 [5,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [6,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [7,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [8,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [9,] 3.290745    0    0 2.533397    0    0    0    0 0.000000

 The minimum distance is: 2.53339692902632


 The closest clusters are: 4, 9


 The grouped clusters are added to the solution.


 Grouping clusters 4 and cluster 9, it is created a new cluster:

    X1   X2
1 6.25 3.14
2 3.75 1.12
3 4.10 1.80
4 4.36 5.21
5 3.90 4.27

 The new cluster is added to the solution.


---------------------------------------------------------------------

STEP => 5


 Matrix Distance (distance type = EUC, approach type = MIN):

          [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]     [,10]
 [1,] 0.000000    0    0    0    0    0    0    0    0 3.290745
 [2,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [3,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [4,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [5,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [6,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [7,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [8,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [9,] 0.000000    0    0    0    0    0    0    0    0 0.000000
[10,] 3.290745    0    0    0    0    0    0    0    0 0.000000

 The minimum distance is: 3.29074459659208
```

```
The closest clusters are: 1, 10


The grouped clusters are added to the solution.


Grouping clusters 1 and cluster 10, it is created a new cluster:

   X1   X2
1 0.89 2.94
2 6.25 3.14
3 3.75 1.12
4 4.10 1.80
5 4.36 5.21
6 3.90 4.27

The new cluster is added to the solution.


This loop has been repeated until the last cluster contained every
single clusters.
```

Como se observa, se ha obtenido una solución mucho más larga, por lo que se analizará paso a paso.

Primero explica que se va a realizar una clusterización jerárquica aglomerativa y lo que va a calcular. Esto es simplemente una pequeña introducción.

A continuación, aparece una línea explicativa que pone que los siguientes serán los clusters con un solo elemento, es decir, los puntos que son clusters y después se muestran.

A partir de aquí, empieza a explicar como realiza la clusterización. En cada paso calcula la matriz de distancia entre los clusters anteriores dependiendo de la proximidad y el tipo de distancia que se haya elegido, se coge la distancia mínima de la matriz, se crea un cluster uniendo los clusters que estén a la distancia mínima y se repite esto hasta que haya un cluster final que incluya todos los anteriores.

Ahora empieza a aplicar el algoritmo con el paso 1. Se calcula la matriz de distancia con el tipo de distancia euclidiana y el tipo de proximidad va a ser "MIN". Luego lo que dice es que la distancia mínima es 0.764787552199956 y, en consecuencia, los clusters más cercanos que hay son el 3 y el 5, por lo tanto, se agrupan los clusters para la solución. Agrupando el cluster 3 y el cluster 5 se crea el nuevo cluster que es el que se proponía antes como el cluster 7 que es el formado por los puntos (3.75, 1.12) y (4.10, 1.80). Este nuevo cluster es añadido a la solución y cómo no hay un solo cluster se realiza otro paso más.

En el paso 2, se hace lo mismo, la matriz de distancias; se obtiene el valor mínimo 1.04651803615609; los clusters más cercanos son el 2 y el 6; se agrupan y se añaden a la solución; y se crea el nuevo cluster.

Se pasa al paso 3, al 4 y al 5 de la misma forma y así se consigue lo que se obtuvo anteriormente pero además explicando lo que se hace en cada uno de los casos. Finaliza con un mensaje de que se ha repetido las iteraciones hasta que se ha obtenido un solo cluster.

Ahora se va a realizar la ejecución de la clusterización aglomerativa usando igualmente la distancia euclídea pero la definición de distancia "MAX". Solamente se cambia el argumento "MIN" por "MAX". La gráfica de salida es la misma que en los casos anteriores ya que se siguen usando los mismos datos (Figura 12).

```
> agglomerativeHC(m,'EUC','MAX')
$dendrogram
Number of objects: 6


$clusters
$clusters[[1]]
    X1   X2
1 0.89 2.94

$clusters[[2]]
    X1   X2
1 4.36 5.21

$clusters[[3]]
    X1   X2
1 3.75 1.12

$clusters[[4]]
    X1   X2
1 6.25 3.14

$clusters[[5]]
   X1  X2
1 4.1 1.8

$clusters[[6]]
   X1   X2
1 3.9 4.27

$clusters[[7]]
    X1   X2
1 3.75 1.12
2 4.10 1.80

$clusters[[8]]
    X1   X2
1 4.36 5.21
2 3.90 4.27

$clusters[[9]]
    X1   X2
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27

$clusters[[10]]
    X1   X2
1 0.89 2.94
2 3.75 1.12
3 4.10 1.80

$clusters[[11]]
    X1   X2
```

```
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27
4 0.89 2.94
5 3.75 1.12
6 4.10 1.80


$groupedClusters
  cluster1 cluster2
1        3        5
2        2        6
3        4        8
4        1        7
5        9       10
```

Calcula la clusterización con el algoritmo "MAX". Los primeros 6 clusters son iguales y luego empieza a hacer las modificaciones correspondientes. Se ejecuta también con la extensión *.details* para conocer más a fondo el funcionamiento del algoritmo.

```
> agglomerativeHC.details(m,'EUC','MAX')
  Agglomerative hierarchical clustering is a classification technique
  that initializes a cluster for each data.


  It calculates the distance between datas depending on the approach
  type given and

 it creates a new cluster joining the most similar clusters until getting
 only one.

  'toList' creates a list initializing datas by creating clusters with
  each one


  These are the clusters with only one element:

[[1]]
     [,1] [,2] [,3]
[1,] 0.89 2.94    1

[[2]]
     [,1] [,2] [,3]
[1,] 4.36 5.21    1

[[3]]
     [,1] [,2] [,3]
[1,] 3.75 1.12    1

[[4]]
     [,1] [,2] [,3]
[1,] 6.25 3.14    1

[[5]]
```

```
     [,1] [,2] [,3]
[1,]  4.1  1.8    1

[[6]]
     [,1] [,2] [,3]
[1,]  3.9 4.27    1


  In each step:
   - It calculates a matrix distance between active clusters depending
   on the approach and distance type.

   - It gets the minimum distance value from the matrix.

   - It creates a new cluster joining the minimum distance clusters.

   - It repeats these steps while final clusters do not include all
   datas.


-----------------------------------------------------------------

STEP => 1


 Matrix Distance (distance type = EUC, approach type = MAX):

         [,1]     [,2]      [,3]     [,4]      [,5]     [,6]
[1,] 0.000000 4.146541 3.3899853 5.363730 3.4064204 3.290745
[2,] 4.146541 0.000000 4.1352388 2.803034 3.4198977 1.046518
[3,] 3.389985 4.135239 0.0000000 3.214094 0.7647876 3.153569
[4,] 5.363730 2.803034 3.2140940 0.000000 2.5333969 2.607566
[5,] 3.406420 3.419898 0.7647876 2.533397 0.0000000 2.478084
[6,] 3.290745 1.046518 3.1535694 2.607566 2.4780839 0.000000

 The minimum distance is: 0.764787552199956


 The closest clusters are: 3, 5


 The grouped clusters are added to the solution.


 Grouping clusters 3 and cluster 5, it is created a new cluster:

    X1   X2
1 3.75 1.12
2 4.10 1.80

 The new cluster is added to the solution.


-----------------------------------------------------------------
```

```
STEP => 2


 Matrix Distance (distance type = EUC, approach type = MAX):

         [,1]     [,2] [,3]     [,4] [,5]     [,6]     [,7]
[1,] 0.000000 4.146541    0 5.363730    0 3.290745 3.406420
[2,] 4.146541 0.000000    0 2.803034    0 1.046518 4.135239
[3,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[4,] 5.363730 2.803034    0 0.000000    0 2.607566 3.214094
[5,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[6,] 3.290745 1.046518    0 2.607566    0 0.000000 3.153569
[7,] 3.406420 4.135239    0 3.214094    0 3.153569 0.000000

 The minimum distance is: 1.04651803615609


 The closest clusters are: 2, 6


 The grouped clusters are added to the solution.


 Grouping clusters 2 and cluster 6, it is created a new cluster:

   X1   X2
1 4.36 5.21
2 3.90 4.27

 The new cluster is added to the solution.


------------------------------------------------------------------

STEP => 3


 Matrix Distance (distance type = EUC, approach type = MAX):

         [,1] [,2] [,3]     [,4] [,5] [,6]     [,7]     [,8]
[1,] 0.000000    0    0 5.363730    0    0 3.406420 4.146541
[2,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[3,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[4,] 5.363730    0    0 0.000000    0    0 3.214094 2.803034
[5,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[6,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[7,] 3.406420    0    0 3.214094    0    0 0.000000 4.135239
[8,] 4.146541    0    0 2.803034    0    0 4.135239 0.000000

 The minimum distance is: 2.80303407043154


 The closest clusters are: 4, 8
```

```
 The grouped clusters are added to the solution.


 Grouping clusters 4 and cluster 8, it is created a new cluster:

   X1   X2
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27

 The new cluster is added to the solution.


------------------------------------------------------------------

STEP => 4


 Matrix Distance (distance type = EUC, approach type = MAX):

        [,1] [,2] [,3] [,4] [,5] [,6]     [,7] [,8]     [,9]
 [1,] 0.00000    0    0    0    0    0 3.406420    0 5.363730
 [2,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [3,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [4,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [5,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [6,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [7,] 3.40642    0    0    0    0    0 0.000000    0 4.135239
 [8,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [9,] 5.36373    0    0    0    0    0 4.135239    0 0.000000

 The minimum distance is: 3.40642040858142


 The closest clusters are: 1, 7


 The grouped clusters are added to the solution.


 Grouping clusters 1 and cluster 7, it is created a new cluster:

   X1   X2
1 0.89 2.94
2 3.75 1.12
3 4.10 1.80

 The new cluster is added to the solution.


------------------------------------------------------------------
```

```
STEP => 5


 Matrix Distance (distance type = EUC, approach type = MAX):

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]    [,9]   [,10]
 [1,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [2,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [3,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [4,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [5,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [6,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [7,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [8,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [9,]    0    0    0    0    0    0    0    0 0.00000 5.36373
[10,]    0    0    0    0    0    0    0    0 5.36373 0.00000

 The minimum distance is: 5.36373004540683


 The closest clusters are: 9, 10


 The grouped clusters are added to the solution.


 Grouping clusters 9 and cluster 10, it is created a new cluster:

   X1   X2
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27
4 0.89 2.94
5 3.75 1.12
6 4.10 1.80

 The new cluster is added to the solution.


 This loop has been repeated until the last cluster contained every
 single clusters.
```

En la gráfica de la Figura 12 solo aparecen los puntos sin estar separados por líneas en los clusters que se han creado. Para visualizarlo de alguna manera, el sistema propone un dendrograma. Se le tiene que asignar un nombre de variable a la solución, en este caso será "cmax". Si se consulta el valor de esta variable, aparece la solución que había antes.

```
> cmax=agglomerativeHC(m,'EUC','MAX')
> cmax
$dendrogram
Number of objects: 6


$clusters
```

```
$clusters[[1]]
    X1   X2
1 0.89 2.94

$clusters[[2]]
    X1   X2
1 4.36 5.21

$clusters[[3]]
    X1   X2
1 3.75 1.12

$clusters[[4]]
    X1   X2
1 6.25 3.14

$clusters[[5]]
   X1  X2
1 4.1 1.8

$clusters[[6]]
   X1   X2
1 3.9 4.27

$clusters[[7]]
    X1   X2
1 3.75 1.12
2 4.10 1.80

$clusters[[8]]
    X1   X2
1 4.36 5.21
2 3.90 4.27

$clusters[[9]]
    X1   X2
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27

$clusters[[10]]
    X1   X2
1 0.89 2.94
2 3.75 1.12
3 4.10 1.80

$clusters[[11]]
    X1   X2
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27
4 0.89 2.94
5 3.75 1.12
6 4.10 1.80
```

```
$groupedClusters
  cluster1 cluster2
1        3        5
2        2        6
3        4        8
4        1        7
5        9       10
```

Al tener asignado ahora un nombre de variable, se puede coger cada uno de los apartados por separado (*$dendrogram*, *$clusters*, *$groupedClusters*...). En este caso se cogerá el dendrograma para poder pintarlo usando la función *plot()*[5] con *cmax$dendrogram*.

```
> plot(cmax$dendrogram)
```

Aparece el dendrograma de la Figura 15.

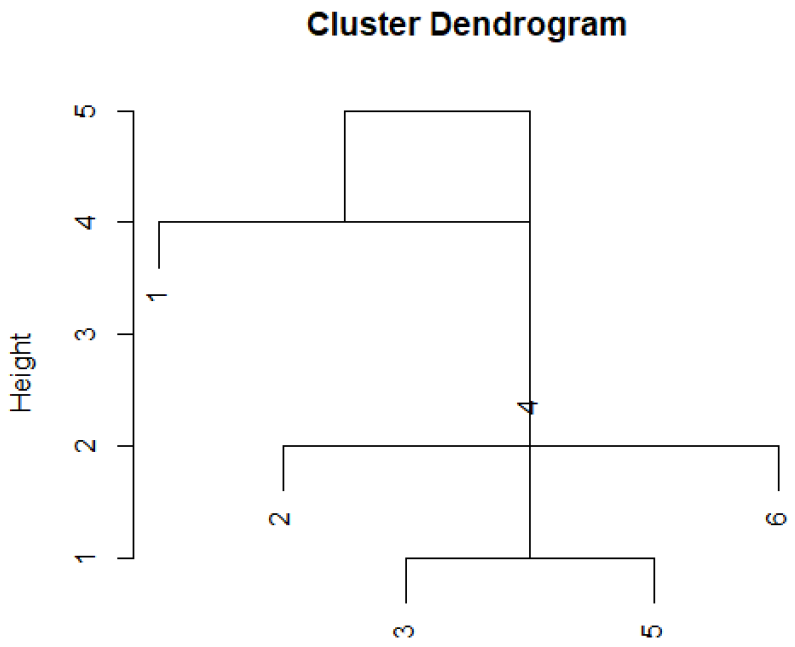


Figure 15: Dendrograma.

2.3 Análisis de clasificación supervisada

En las técnicas de clasificación supervisada a diferencia de la no supervisada se conocen los posibles valores que puede tener el valor desconocido de un conjunto de datos.

2.3.1 Utilizando árboles de decisión

La primera técnica que se va a utilizar es la de los *Arboles de Decision*. Para poder trabajar con ella en R se utiliza el paquete *rpart*[8].

Antes que todo, se debe ir a la página oficial de R para descargar el paquete r-release: $rpart_4.1 - 15.zip$ que se va usar en esta parte de la práctica.

Se va a cargar este paquete utilizando una forma distinta a la empleada hasta ahora. Para ello, se tiene que ir al menú de R, en el apartado *Paquete* y escoger la opción *Install package(s) from local files* como se puede ver en la Figura 16.

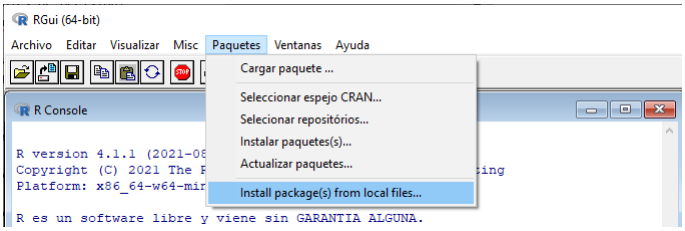


Figure 16: Instalar paquete local en R.

Luego ir a la ruta en donde se tenga guardado el paquete, seleccionarlo y darle *Abrir* para cargarlo.

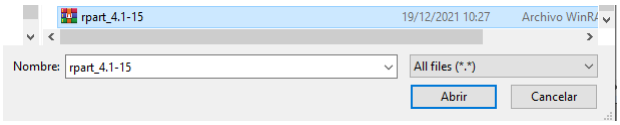


Figure 17: Abrir paquete local.

Saldrá un mensaje en la consola de R diciendo que el paquete *rpart* se desempaquetó correctamente.

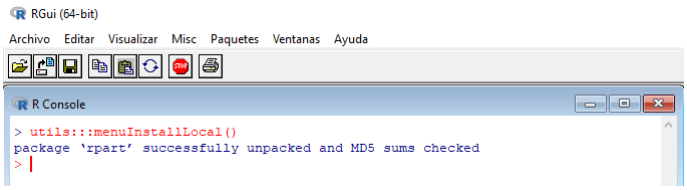


Figure 18: Paquete local instalado correctamente.

Se emplea el comando *library()*[4] añadiendo como parámetro el nombre del paquete que se desea cargar:

```
> library(rpart)

Warning message:
package 'rpart' was built under R version 4.1.2
```

El comando *search()*[4] muestra la lista de los paquetes adjuntos y se puede apreciar el paquete *rpart*:

```
> search()
 [1] ".GlobalEnv"        "package:rpart"     "package:stats"
```

```
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

Se crea un archivo de texto, *clasificaciones.txt* con los datos del ejercicio hecho en clase de teoría.

En una variable llamada *calificaciones* se añadirán los datos que obedecen al mismo nombre.

Las notas de teoría (T), laboratorio (L) y práctica (P) podrán ser desde la A-D, la A es la mayor calificación y la D es la menor. La calificación general (C.G) podrá ser aprobado (Ap) y suspenso (Ss).

```
> calificaciones = read.table("calificaciones.txt")
> calificaciones
  T L P C.G
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
```

Si se emplea el comando *data.frame()*[4] añadiendo la variable *calificaciones* se obtiene el mismo resultado anterior, por lo que se puede prescindir:

```
> muestra = data.frame(calificaciones)
> muestra
.............................
```

La función estrella del paquete es *rpart()* y permitirá hacer una clasificación global de las notas utilizando todos los atributos del archivo *calificaciones.txt*.

Parámetros de la función:

- C.G~. (Calificación Global) que al añadir el punto al final toma a todos los atributos del archivo, equivalente a C.G~T+L+P. Si solo se quieren utilizar algunos atributos, se nombran los que se necesitan separados por el carácter más(+).
- data=calificaciones: La variable *calificaciones* almacenan los datos que se tomarán para trabajar.
- method="class": Se emplea *class* porque se quiere hacer un árbol de clasificación (Algoritmo de Hunt) con datos cualitativos.
- minsplit=1: número mínimo de observaciones que deben existir en un nodo para que se intente una división.

En el ejercicio hecho en clases de teoría se ha escogido el atributo Prácticas y luego Laboratorio, pero en esta solución se ha hecho al revés obteniendo el mismo resultado de clasificación. La función evita Teoría porque no le hace falta para clasificar. Los asteriscos del final significa que son los nodos terminales. A continuación, se muestra la trayectoria seguida para la solución:

```
> clasificacion=rpart(C.G~.,data=calificaciones,method="class",minsplit=1)
> clasificacion
n= 9

node), split, n, loss, yval, (yprob)
```

```
      * denotes terminal node

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

2.3.2 Utilizando regresión

Para la resolución de este apartado se crea un archivo de texto, *planetas.txt* con los datos del ejercicio hecho en clase de teoría.

La variable *muestra2* almacenará los datos que se encuentran en el archivo de los planetas, R:radio y D:densidad, y se muestra su contenido:

```
> muestra2 = read.table("planetas.txt")
> muestra2
           R   D
Mercurio 2.4 5.4
Venus    6.1 5.2
Tierra   6.4 5.5
Marte    3.4 3.9
```

La función *lm()* permitirá hacer la regresión lineal de la variable *muestra2*. Se añade como parámetro D~R que significa que la D:densidad está en función de la R: radio.

```
> regresion=lm(D~R, data=muestra2)
> regresion

Call:
lm(formula = D ~ R, data = muestra2)

Coefficients:
(Intercept)            R
     4.3624       0.1394
```

Nota: La función lineal de la recta de regresión obedece a la forma $y = ax + b$. El coeficiente $a = 4.3624$ y $b = 0.1394$.

Faltaría por determinar si la recta de regresión es buena o mala, ¿se puede desde radio obtener las densidades? En clase de teoría salía que no. Cuando hemos calculado los residuos, el r^2 de la función calculada y la observada salía que era con 0.14 y este valor era muy pequeño.

La función *summary()* pasándole como argumento la variable *regresion* muestra un conjunto de coeficientes que se utilizan cuando se hace un análisis que va desde la muestra hasta la población. El resultado que a nosotros como estudiantes nos interesa es $MultipleR - squared$ igual a 0.1377, si se redondea a dos decimales es 0.14 (valor que se obtenía en clases de teoría).

```
> summary(regresion)

Call:
lm(formula = D ~ R, data = muestra2)

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
```

3 Segunda parte

3.1 Análisis de detección de datos anómalos

Para las técnicas con base estadística se utilizará la muestra de la primera parte de las resistencias y densidades y la segunda muestra de los datos de velocidades y temperaturas del enunciado de la segunda parte.

```
Velocidad,Temperatura
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
```

La muestra de datos se guarda en un documento CSV y se transforma en un *data frame* con el método *read.csv()*[7].

```
> (muestra2 = read.csv("muestra.csv", header = TRUE))
   Velocidad Temperatura
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
```

Los algoritmos de cada uno de las técnicas se programarán manualmente con funciones y se compararán los resultados que muestren con los mostrados con las funciones que proporciona R y los métodos seguidos en la primera parte.

3.1.1 Caja y Bigotes

Para calcular los límites del intervalo para los valores atípicos primero se utiliza la función para calcular los cuantiles que se creó en la práctica anterior.

Para ello se realiza la ecuación matemática que halla el cuantil dado el vector de datos y el número de cuantil que se se quiere obtener. En esta función se utiliza el método *ordenar()* disponible de la anterior práctica.

```
cuantil <- function(x, valor)
{
        xOrdenado <- x[ordenar(x)]

        i <- (valor * longitud(xOrdenado))
        if(es_entero(i)) return((xOrdenado[i] + xOrdenado[i+1])/2)
        else return(xOrdenado[trunc(i)+1])
}
```

Después se crea la función *intervaloCyB()* que recibirá el vector de datos y el grado de outlier y devolverá la fórmula de los límites tal como se explicó en la primera parte.

```
intervaloCyB <- function(x, d)
{
        cuar1r <- cuantil(x, 0.25)
        cuar3r <- cuantil(x, 0.75)

        c(cuar1r - d * (cuar3r - cuar1r), cuar3r + d * (cuar3r - cuar1r))
}
```

Los límites del intervalo con los datos de resistencia de la muestra y grado de outlier 1.5 se muestran a continuación.

```
> intervaloCyB(muestra$r, 1.5)
[1] -1.9 12.5
```

La otra función creada determina cuál es el outlier dados los datos de un tipo de la muestra y los límites del intervalo calculados en la función anterior. Para ello en *mostrarOutlier()* se recorren los datos y se comprueba cuál de ellos se sale del intervalo dado para mostrarlo por pantalla. En esta función se utiliza el método *longitud()* creado en la anterior práctica.

```
mostrarOutlier <- function(x, int)
{
        for (i in 1:longitud(x))
        {
                if (x[i] < int[1] || x[i] > int[2])
                {
                        print(paste0("El suceso ", i, " cuyo valor es ",
                        x[i], " es un suceso anómalo o outlier"))
                }
        }
}
```

Esta función se utiliza en la función principal del algoritmo, *CajayBigotes()*, que recibirá los datos de un tipo de la muestra y el grado de outlier, y calculará los límites del intervalo y mostrará el outlier u outliers correspondientes.

```
CajayBigotes <- function(x, d)
{
    int = intervaloCyB(x, d)
```

```
    mostrarOutlier(x, int)
}
```

Con los datos de temperaturas de *muestra2* y el grado de outlier 1.5, no existe ningún outlier por lo que no se mostrará nada por pantalla.

```
> CajayBigotes(muestra2$Velocidad, 1.5)
```

Con las funciones de la primera parte, se ve que tampoco se muestra ningún outlier:

```
> cuar1r=quantile(muestra2$Velocidad,0.25)
> cuar3r=quantile(muestra2$Velocidad,0.75)
> int=c(cuar1r-1.5*(cuar3r-cuar1r), cuar3r+1.5*(cuar3r-cuar1r))

> for(i in 1:length(muestra2$Velocidad)){
    if(muestra2$Velocidad[i]<int[1] || muestra2$Velocidad[i]>int[2])
        {print("el suceso");
        print(i);
        print(muestra2$Velocidad[i]);print("es un suceso anómalo o outlier")
    }
 }
```

En el caso de los datos de resistencia de la primera muestra y grado de outlier 1.5 el outlier será el siguiente:

```
> CajayBigotes(muestra$r, 1.5)
[1] "El suceso 7 cuyo valor es 14 es un suceso anómalo o outlier"
```

El suceso 7 es un outlier, tal como se mostraba en la primera parte.

3.1.2 Desviación Típica

Para la técnica de desviación típica se crea una función para calcular los límites del intervalo para los valores atípicos recibiendo como parámetros el vector de los datos y el grado de outlier arbitrario. Primero se calcula la media aritmética, luego la desviación típica (ambos con las funciones creadas en la primera práctica) y por último se calculan los límites con esos valores.

```
intervaloDT <- function(x, d)
{
        media <- mediaAritmetica(x)
        sd <- desTip(x)

        c(media - d * sd, media + d * sd)
}
```

Para la muestra de la primera parte se calculan los límites de las densidades con grado de outlier 2.

```
> intervaloDT(muestra$d, 2)
[1] -0.05685714 11.37114285
```

Y para la segunda muestra se calculan los límites de las temperaturas con grado de outlier 2.

```
> intervaloDT(muestra2$Temperatura, 2)
[1]  3.628134 11.371866
```

Para saber el outlier se crea la función principal del algoritmo, *DesviacionTipica()*, donde se calculan los límites del intervalo con la función anterior y de muestra el outlier o los outliers con la función creada en la técnica de Caja y Bigotes.

```
DesviacionTipica <- function(x, d)
{
        int = intervaloDT(x, d)

        mostrarOutlier(x, int)
}
```

Para los datos de temperaturas de la segunda muestra y con grado de outlier 2 el outlier es el siguiente:

```
> DesviacionTipica(muestra2$Temperatura, 2)
[1] "El suceso 3 cuyo valor es 12.74 es un suceso anómalo o outlier"
```

Utilizando las funciones de la primera parte se ve que el outlier coincide.

```
> sdd = sqrt(var(muestra2$Temperatura)*((length(muestra2$Temperatura)-1)/length(muestra2$Temperatura
> int=c(mean(muestra2$Temperatura) - 2*sdd, mean(muestra2$Temperatura) + 2*sdd)
> for(i in 1:length(muestra2$Temperatura)){
    if(muestra2$Temperatura[i]<int[1] || muestra2$Temperatura[i]>int[2])
        {print("el suceso");
        print(i);
        print(muestra2$Temperatura[i]);print("es un suceso anómalo o outlier")
    }
 }

[1] "el suceso"
[1] 3
[1] 12.74
[1] "es un suceso anómalo o outlier"
```

Con los datos de densidades de la primera muestra y grado de outlier 2 el outlier es el siguiente:

```
> DesviacionTipica(muestra$d, 2)
[1] "El suceso 2 cuyo valor es 12 es un suceso anómalo o outlier"
```

El suceso 2 es un outlier, igual que se mostraba con las funciones usadas en la primera parte.

3.1.3 Regresión

En la última técnica con base estadística, la técnica de regresión, se crean funciones para las fórmulas principales del método.

La primeras funciones serán para obtener la regresión lineal, siendo la primera de ellas para calcular Sxy. En ella se pasan como argumentos los vectores de datos para x y para y. Se realiza la media aritmética de ambos datos y se realiza el sumatorio de los conjuntos de datos multiplicados con la función creada en la primera práctica. Por último se realza el cálculo completo con los parámetros anteriores tal como se sigue en la fórmula.

```
Sxy <- function(x, y)
{
        mediax <- mediaAritmetica(x)
        mediay <- mediaAritmetica(y)
```

```
        mul = c()
        n = longitud(x)

        for (i in 1:n)
        {
                mul = c(mul, x[i] * y[i])
        }

        (sumatorio(mul) / n) - (mediax * mediay)
}
```

Para los datos de resistencias y densidades de la primera muestra el valor de Sxy será el siguiente:

```
> Sxy(muestra$r, muestra$d)
[1] -0.6738776
```

La variable b de la recta de regresión se calcula con la siguiente función:

```
bRecta <- function(x, y)
{
        Sxy(x, y) / desTip(x) ^ 2
}
```

Con los datos de resistencia y densidad tendrá el siguiente valor:

```
> (b <- bRecta(muestra$r, muestra$d))
[1] -0.05723497
```

La variable a de la recta de regresión se calcula con la variable b y las medias aritméticas de los datos x e y.

```
aRecta <- function(x, y, b)
{
        mediaAritmetica(y) - b * mediaAritmetica(x)
}
```

Con los datos de resistencia y densidad tendrá el siguiente valor:

```
> (a <- aRecta(muestra$r, muestra$d, b))
[1] 6.014453
```

Después de calcular la recta de regresión lineal habrá que obtener el error estándar de los residuos. Para ello primero hay que obtener la y calculada para todos los puntos de x con la función creada *yCalculada()* siguiente la fórmula correspondiente.

```
yCalculada <- function(x, a, b)
{
        yc = c()

        for (i in 1:longitud(x))
        {
                yc = c(yc, a + b * x[i])
        }

        yc
}
```

Con los datos de resistencias la y calculada es la siguiente:

```
> yCalculada(muestra$r, a, b)
[1] 5.842748 5.814130 5.745448 5.716831 5.608084 5.659596 5.213163
```

Tras tener la y calculada ya se pueden obtener los residuos con la siguiente función:

```
residuos <- function(x, y, a, b)
{
        yc = yCalculada(x, a, b)

        res <- c()
        for (i in 1:longitud(y))
        {
                res <- c(res, y[i] - yc[i])
        }

        res
}
```

Para los datos de resistencias y densidades los residuos serán estos:

```
> (res <- residuos(muestra$r, muestra$d, a, b))
[1] -3.8427477  6.1858698 -1.6454482 -0.8168308  0.4919157 -0.4595958
0.0868370
```

El error estándar será la desviación típica de los residuos.

```
> desTip(res)
[1] 2.850242
```

Finalmente se crea la función principal de la técnica de regresión teniendo como entradas los datos para x e y, y el grado de outlier. En esta se guardan en variables los datos para las variables a y b de la recta de regresión, los residuos y el error estándar. Después se recorren los residuos para determinas los outliers, estos deben cumplir que deben ser mayores que el límite de intervalo (grado de outlier multiplicado por el error estándar) y se mostrarán por pantalla.

```
Regresion <- function(x, y, d)
{
        b <- bRecta(x, y)
        a <- aRecta(x, y, b)
        res <- residuos(x, y, a, b)
        sr <- desTip(res)

        for(i in 1:longitud(res))
        {
                if (res[i] > d * sr)
                {
                        print(paste0("El suceso ", i, " cuyo valor es ",
                        round(res[i], 5), " es un suceso anómalo o outlier"))
                }
        }
}
```

Con los datos de la primera muestra y grado de outlier 2 el outlier encontrado es el siguiente:

```
> Regresion(muestra$r, muestra$d, 2)
[1] "El suceso 2 cuyo valor es 6.18587 es un suceso anómalo o outlier"
```

El suceso 2 es un outlier, tal como se mostró también en las funciones usadas en la primera parte.

Con los datos de la segunda muestra y grado de outlier 2 el outlier encontrado es el siguiente:

```
> Regresion(muestra2$Velocidad, muestra2$Temperatura, 2)
[1] "El suceso 3 cuyo valor es 3.24109 es un suceso anómalo o outlier"
```

Comparándolo con el uso de las funciones de la primer parte, el outlier no coincide (no muestra ninguno).

```
> dfr=lm(muestra2$Velocidad~muestra2$Temperatura)
> res = summary(dfr)$residuals
> sr = sqrt(sum(res^2)/11)
> for(i in 1:length(res))
+ {if (res[i]>2*sr)
+ {print("el suceso"); print(res[i]); print("es un suceso anómalo o outlier")}}
```

3.1.4 K-vecinos (*k-nearest neighbors*)

Para este apartado se usará algunas funciones que se ha programado o se emplearán algunas funciones de otros paquetes para solucionar el ejercicio con los datos de Mujeres y Hombres.

En el siguiente ejercicio se detectarán los datos anómalos utilizando técnicas basadas en la proximidad y en la densidad, estará formado por el número de Mujeres y Hombres inscritos en una serie de cinco seminarios que se han impartido sobre biología. Los datos son: Mujeres, Hombres: 1. 9, 9; 2. 9, 7; 3. 11, 11; 4. 2, 1; 5. 11, 9.

La función *matriz()* sustituye a *matrix()*, convierte el vector en una matriz con tantas filas y columnas como se especifique por parámetro. El primer parámetro es para el vector, el segundo para la cantidad de filas y el tercero para las columnas.

```
matriz <- function(x,fila, colum)
{
        m <- x
        dim(m) <- c(fila,colum)
        m
}

> (muestra2=matriz(c(9,9,9,7,11,11,2,1,11,9),2,5))
     [,1] [,2] [,3] [,4] [,5]
[1,]    9    9   11    2   11
[2,]    9    7   11    1    9
```

La función *aperm()* pertenece al paquete base[4] y transpone la matriz que se le pasa por parámetro igual que la función *t()*.

```
> (muestra2=aperm(muestra2))
     [,1] [,2]
[1,]    9    9
[2,]    9    7
[3,]   11   11
[4,]    2    1
[5,]   11    9
```

La función *daisy()*, perteneciente al paquete llamado *cluster*[3] que previamente se instaló, calcula las distancias euclídeas por pares entre observaciones en el conjunto de datos de la matriz que se le pasa por parámetro. Si por parámetro no se especifica la métrica a emplear, por defecto utilizada la euclídea. Retorna el mismo resultado que

la función *dist()*, pero muestra la métrica usada y el número de objetos que tiene la matriz.

```
> utils:::menuInstallLocal()
package ‘cluster’ successfully unpacked and MD5 sums checked
> library(cluster)
Warning message:
package ‘cluster’ was built under R version 4.1.2
> daisy(muestra2)
Dissimilarities :
          1         2         3         4
2  2.000000
3  2.828427  4.472136
4 10.630146  9.219544 13.453624
5  2.000000  2.828427  2.000000 12.041595

Metric :  euclidean
Number of objects : 5
```

En vez de usar la función novedosa explicada anteriormente, se programa una función que cree la matriz de distancia completa. La función *matrizDistancias()* sustituye a *as.matrix()*, usa la matriz de los puntos para calcular las distancias euclídeas entre ellos. El parámetro es para la matriz que contiene a los puntos. Como función auxiliar a esta, se necesita la función *distanciaEuclidiana()* sustituye a *dist()*. Calcula la distancia euclídea entre los dos puntos pasados por parámetro.

```
distanciaEuclidiana <- function(punto1, punto2)
{
        acumulador <- 0

        for (i in 1:length(punto1))
        {
                acumulador <- acumulador + (punto1[i] - punto2[i]) ^ 2
        }

        sqrt(acumulador)
}

matrizDistancias <- function(m)
{
  distancias <- c()
  numPuntos <- dim(m)[1]
  for (punto1 in 1:numPuntos)
  {
    for (punto2 in 1:numPuntos)
    {
      if (punto1 == punto2)
      {
        distancias <- append(distancias, 0)
      }
      else
      {
        distancia <- distanciaEuclidiana(m[punto1,], m[punto2,])
        distancias <- append(distancias, distancia)
      }
    }
```

```
  }
  matriz(distancias, numPuntos, numPuntos)
}
```

```
> matrizDistancias(muestra2)
          [,1]     [,2]      [,3]      [,4]      [,5]
[1,]  0.000000 2.000000  2.828427 10.630146  2.000000
[2,]  2.000000 0.000000  4.472136  9.219544  2.828427
[3,]  2.828427 4.472136  0.000000 13.453624  2.000000
[4,] 10.630146 9.219544 13.453624  0.000000 12.041595
[5,]  2.000000 2.828427  2.000000 12.041595  0.000000
```

La función *ordenar()* programada en la PL1, se reutilizará para ordenar los valores de las columnas de la matriz. El algoritmo de ordenamiento es *la ordenación de la burbuja*.

```
ordenar <- function(x)
{
        idx <- 1:longitud(x)
        for(i in 1:(longitud(x)-1))
        {
                for(j in 1:(longitud(x)-1))
                {
                        if(x[j+1] <= x[j]){
                                aux <- x[j+1]
                                x[j+1] <- x[j]
                                x[j] <- aux

                                auxIdx <- idx[j+1]
                                idx[j+1] <- idx[j]
                                idx[j] <- auxIdx
                        }
                }
        }
        return(idx)
}
```

La función *ordenarColMatriz()* ordena de menor a mayor los valores de las columnas de la matriz pasada por parámetro.

```
ordenarColMatriz <- function(x)
{
        for(i in 1:ncol(x))
        {
                x[,i]=x[ordenar(x[,i]),i]
        };
        x
}
```

```
> (distanciasordenadas2=ordenarColMatriz(distancias2))
          [,1]     [,2]      [,3]      [,4]      [,5]
[1,]  0.000000 0.000000  0.000000  0.000000  0.000000
[2,]  2.000000 2.000000  2.000000  9.219544  2.000000
[3,]  2.000000 2.828427  2.828427 10.630146  2.000000
[4,]  2.828427 4.472136  4.472136 12.041595  2.828427
[5,] 10.630146 9.219544 13.453624 13.453624 12.041595
```

La función *sucAnomalo()* comprueba si hay algún valor outlier en la matriz x mirando los K-vecinos más próximos. Todos los valores de la primera fila de la matriz x son ceros porque expresan las distancias de estos a sí mismos. El nuevo K será $K+1$, en este caso $K = 4$ $(3+1)$, para ignorar la primera fila. Se recorre la matriz x mostrando el/los valor/es y el mensaje "es un suceso anómalo o outlier" si el $K+1$ vecino más próximo es/son mayor/es que el grado de outlier d.

```
sucAnomalo <- function(x,d,k)
{
        for(i in 1:ncol(x))
        {
                if(x[k+1,i]>d)
                {
                        print(i);
                        print("es un suceso anómalo o outlier")
                }
        }
}
```

Se gana en optimización al fusionar el código de los anteriores apartados.

```
kvecinos <- function(v,d,k)
{
  n = length(v)/2
  m = ordenarColMatriz(matrizDistancias(aperm(matriz(v,2,n))))

  sucAnomalo(m,d,k)
}
```

La función *kvecinos()* recibe el vector con los datos para un $d = 4.5$, elegido arbitrariamente, es el valor 4.

```
> kvecinos(c(9,9,9,7,11,11,2,1,11,9),4.5,3)
[1] 4
[1] "es un suceso anómalo o outlier"
```

A continuación, se va a utilizar las calificaciones de los estudiantes de la *Primera parte* para comprobar que el resultado devuelto por la *kvecinos()* es el mismo.

```
> kvecinos(c(4,4,4,3,5,5,1,1,5,4),2.5,3)
[1] 4
[1] "es un suceso anómalo o outlier"
```

En esta parte se usará la función *daisy()*. Esta calcula por defecto las distancias euclídeas, pero se requiere para este caso que sean calculadas por la métrica *Manhattan*.

Longitud Manhattan: La distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas.

La matriz distanciasM tiene las distancias entre los puntos de la muestra utilizando la métrica *Manhattan*. La función *daisy()* recibe como parámetro la métrica *manhattan* que calculará las distancias entre los valores almacenados en la variable *matriz2*. La distancia de cada punto a si mismo es cero, es por eso que la diagonal de la matriz es de ceros.

Comprobación del resultado de la matriz tomando como ejemplo el cálculo de las distancias entre el p1 con el p2 y el p4 con el p5. Distancia del p1 (4,4) con el p2 (4,3) es igual a 1 porque $|4-4|+|4-3| = 1$. Distancia del p4 (1,1) con el p5 (5,4) es igual a 7 porque $|1-5|+|1-4| = 7$.

```
> (distanciasM=as.matrix(daisy(matriz2,metric="manhattan")))
  1 2 3 4 5
1 0 1 2 6 1
```

```
2 1 0 3 5 2
3 2 3 0 8 1
4 6 5 8 0 7
5 1 2 1 7 0
```

La función *sacarN()* recibe como primer parámetro la matriz a analizar y como segundo el valor de la K. La mayor N posible es igual a la cantidad de filas de la matriz. Si $(k+2)$, El $K+1$ quitaría el valor 0 de la primera fila y sumándole otro 1 es la siguiente posición, es mayor que la última fila de la matriz el valor de la N será *(nrow(x) - 1)*.

El valor de la N es el mismo que de K, pero puede ser superior. Para comprobarlo, se recorre la matriz *distanciasordenadas2* por las columnas actualizando el valor de N si es mayor que la anterior. Se toma el valor de la posición $(k+1)$ (*ultdist*), si este valor es igual a la posición siguiente se le suma 1 a la N y se termina de comprobar hasta que se llegue a la última fila de la matriz o los números (*ultdist* y el valor actual) sean distintos. Se retorna el valor de la N.

```
sacarN <- function(x,K)
{
        if((K+2) > nrow(x)) (nrow(x) - 1)
        else
        {
                N <- K
                for(j in 1:ncol(x))
                {
                        ultdist <- x[(K+1),j]
                        cont <- K
                        for(i in (K+2):nrow(x))
                        {
                                if(round(ultdist,2) == round(x[i,j],2)) cont = cont + 1
                                else break;
                        }
                        N <- max(N,cont)
                }
                N
        }
}
```

La función *sacarN()* recibe como parámetro la matriz *distanciasordenadas2* y $K = 1$.

Se empieza recorriendo la matriz por las columnas. Se hace un $K+1$ para obviar la fila número uno, el valor la posición [2,1] lo consideramos como *ultdist* (última distancia) y se comprueba si es igual a su K-vecinos más próximo, posición [3,1]. Al tener ambas posiciones el mismo valor, 2.0, se le añade 1 al valor del *cont* (contador) que inicialmente era igual K. Ahora el valor del contador es igual a 2. El siguiente K-vecino, posición [4,1], es distinto a *ultdist* y es por eso que no se sigue comprobando en esa columna y se actualiza la N porque el contador es mayor. Ahora el valor de $N = 2$. Se realiza este proceso con las columnas siguientes.

```
> sacarN(distanciasordenadas2,1)
[1] 2
```

3.2 Análisis de clasificación no supervisada

3.2.1 K-medias (*K-means*)

Para el análisis de clasificación no supervisada con el algoritmo *K-means* se va a desarrollar una función que realice dicho algoritmo para devolver los mismos resultados, (*Cluster means*), que la función que ofrece R.

Para su desarrollo se crearán funciones auxiliares que realizarán los pasos de los que se compone en el algoritmo completo.

La primera función auxiliar realiza el cálculo de la distancia euclidiana que se ha explicado en el método de k-vecinos entre dos puntos representados como una matriz. En el caso del algoritmo se utilizará para la distancia entre cada punto de los datos a cada uno de los centroides definidos.

A continuación se realiza la función que asignará los puntos a los clusters. Esta función simula el paso en el que se realiza una matriz con las distancias de cada punto a cada centroide y se asignan los puntos al centroide que estén más cercanos. La función se define de la siguiente forma.

```
k_meansCluster <- function(matriz, centroides)
{
        clasificacion <- c()
        for (i in 1:nrow(matriz))
        {
                punto <- matriz[i,]
                cluster_mejor <- 0
                distancia_mejor <- Inf

                for (j in 1:nrow(centroides))
                {
                        distancia <- distanciaEuclidiana(punto,
                        centroides[j,])
                        if (distancia < distancia_mejor)
                        {
                                distancia_mejor <- distancia
                                cluster_mejor <- j
                        }
                }

                clasificacion <- c(clasificacion, cluster_mejor)
        }

        as.data.frame(clasificacion)
}
```

En dicha función se pasan como argumentos la matriz de los datos (los puntos) y la matriz de los centroides. La matriz de datos se recorre por filas guardando los puntos de cada una de ellas en una variable, tras guardar un punto se calcula la distancia euclidiana de éste con cada uno de los centroides y se comprueba cuál es la menor distancia para saber el cluster que lo cumple. Los clusters que cumplen la condición de menor distancia se guardan en un vector a medida que se van recorriendo los puntos que faltan de la matriz de datos. Al final se convierte en un *data frame* el vector con los clusters.

Se comprueba el correcto funcionamiento de la función con los datos utilizados en la primera parte.

```
> (cluster <- k_meansCluster(m, c))
  clasificacion
```

```
1             2
2             2
3             2
4             2
5             1
6             2
7             2
8             2
```

La siguiente función se utilizará para separar los puntos asignados a cada cluster.

```
k_meansExtraer <- function(matriz, cluster, centroides)
{
        clusters = cbind(cluster, matriz)

        resultado <- c()

        for (i in 1:nrow(centroides))
        {
                cluster = subset(clusters, clusters[,1] == i)
                cluster = cluster[,-1]

                resultado <- c(resultado, list(cluster))
        }

        resultado
}
```

En el caso del ejemplo seguido el primer cluster estará formado solo por el punto 5 y el segundo cluster por los demás.

```
> (clusters <- k_meansExtraer(m, cluster, c))
[[1]]
  1 2
5 0 1

[[2]]
  1 2
1 4 4
2 3 5
3 1 2
4 5 5
6 2 2
7 4 5
8 2 1
```

Tras tener los clusters separados se recalcularán los centroides con los puntos. Para ello se realiza la siguiente función:

```
k_meansRecalcular <- function(clusters)
{
        centroides <- c()

        for (cluster in clusters)
        {
                for (colum in cluster)
                {
```

```
                        centroides <- c(centroides,
                        mediaAritmetica(colum))
                }
        }

        t(matrix(centroides, length(clusters[[1]]), length(clusters)))
}
```

En esta función se recorren los clusters y en cada uno de ellos por cada columna (los elementos de los puntos) se realiza la media aritmética. Tras esto se guardan los nuevos valores de los centroides en una matriz. Los nuevos centroides con los clusters obtenidos serán los siguientes:

```
> k_meansRecalcular(clusters)
     [,1]     [,2]
[1,]    0 3.000000
[2,]    1 3.428571
```

Por último se realiza la función principal del algoritmo.

```
k_means <- function(matriz, centroides)
{
        cluster <- rep(0, longitud(matriz[,1]))
        cluster <- as.data.frame(cluster)

        repeat
        {
                nuevoCluster <- k_meansCluster(matriz, centroides)

                if (all(cluster == nuevoCluster))
                {
                        return(centroides)
                }

                clusters <- k_meansExtraer(matriz, nuevoCluster,
                centroides)
                centroides <- k_meansRecalcular(clusters)

                cluster <- nuevoCluster
        }
}
```

La función *k_means* recibe como argumento la matriz de datos y de centroides y utiliza las funciones explicadas anteriormente para realizar todo los pasos del algoritmo en el orden correspondiente. Se comienza creando un cluster vacío de ceros con la longitud del número de datos, luego se sigue el orden de los pasos del algoritmo, se calcula el vector de clusters con la matriz de datos y centroides, se separan los clusters con sus respectivos puntos y se recalculan los centroides. Este orden se repetirá hasta que el vector de clusters calculado (el que indica a qué cluster pertenece cada punto) sea igual que el anterior, es decir no haya cambios. Cuando se cumpla la condición se devolverán los centroides finales calculados.

Los centroides finales, que coinciden con los que ofrece la función de R, serán los siguientes:

```
> k_means(m, c)
     [,1] [,2]
[1,] 1.25 4.00
[2,] 1.50 4.75
```

A continuación se muestra la aplicación de la función con la muestra de datos y centroides del enunciado.

```
> (m2 <- read.csv("muestra.csv", header = TRUE))
   Velocidad Temperatura
1       3.50        4.50
2       0.75        3.32
3       0.00        3.00
4       1.75        0.75
5       3.00        3.75
6       3.75        4.50
7       1.25        0.75
8       0.25        3.00
9       3.50        4.25
10      1.50        0.50
11      1.00        1.00
12      3.00        4.00
13      0.50        3.00
14      2.00        0.25
15      0.00        2.50
```

```
> (c2<-t(matrix(c(0,3,1,1,2,0.25),2,3)))
     [,1] [,2]
[1,]    0 3.00
[2,]    1 1.00
[3,]    2 0.25
```

```
> k_means(m2, c2)
     [,1]  [,2]
[1,] 3.35 4.200
[2,] 0.30 2.964
[3,] 1.50 0.650
```

Tras utilizar la función *k_means* creada se comprueba si coinciden los resultados con la función que ofrece R.

```
> kmeans(m2, c2)
K-means clustering with 3 clusters of sizes 5, 5, 5

Cluster means:
  Velocidad Temperatura
1      3.35       4.200
2      0.30       2.964
3      1.50       0.650

Clustering vector:
 [1] 1 2 2 3 1 1 3 2 1 3 3 1 2 3 2

Within cluster sum of squares by cluster:
[1] 0.87500 0.77092 0.95000
 (between_SS / total_SS =  95.6 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss"      "betweenss"    "size"         "iter"
[9] "ifault"
```

Se puede comprobar que los centroides de los tres clusters devueltos por ambas funciones coinciden.

3.2.2 Clustering jerárquico aglomerativo

En la primera parte, se hizo un pequeño acercamiento al análisis del paquete *LearnClust*[1]. Ahora se adentrará en el paquete conociendo más a fondo los aspectos más importantes. Se irá probando una a una las funciones que contiene este paquete y comentando los aspectos de cada una de ellas.

Para empezar se consulta la documentación donde se explican las funciones del paquete. Esta se puede encontrar en el documento *LearnClust.pdf* en el apartado *Reference manual* de la página web del paquete (Figura 9).

En un primer vistazo de la documentación se puede ver que en el apartado en el que se listan todas las funciones, *R topics documented*, se repite un patrón (Figura 19). Cada función tiene su homóloga con la extensión *.details*; estas contienen explicación sobre el algoritmo o fórmula usados en esa función.

R topics documented:

Figure 19: *R topics documented.*

Las funciones del paquete son las siguientes:

1. *agglomerativeHC(data, distance, approach)*: Tiene la opción con *.details*. Se usó y explicó en la parte anterior.
2. *canberradistance(x, y)*: Tiene la opción con *.details*. Tiene dos argumentos, "x" e "y", que son vectores numéricos o matrices y que representan los valores de dos clústeres. Calcula la distancia de Canberra de dos clústers. Un ejemplo de ejecución es la siguiente:

```
> x <- c(1,2)
> y <- c(1,3)
> canberradistance(x,y)
[1] 0.2
> canberradistance.details(x,y)
Error in magick_image_readpath(path, density, depth, strip,
defines) :
rsession.exe: UnableToOpenBlob 'C:\Users\atien\OneDrive
- Universidad de Alcala\Documents\man\images\
canberraDistance.PNG': No such file or directory
@ error/blob.c/OpenBlob/2924
```

NOTA IMPORTANTE: Se ha encontrado un error al cargar una imagen PNG en la función *canberradistance.details*. El fichero o directorio no existe. Se ha comprobado en varios PCs dándose el mismo error. También se probó diferentes métodos de instalación del paquete por si el motivo era una mala instalación pero el error sigue siendo persistente. Se ha observado que en el código fuente de esta función usa una ruta relativa a un fichero que no existe en la función *initImages*, lo que es el motivo del error. Además en el apartado *Examples* de la documentación de la función *canberradistance.details* no aparece un ejemplo de ejecución de esta función sino de la función *canberradistance*.

3. *canberradistanceW(cluster1, cluster2, weight)*: Tiene la opción con *.details*.
   Tiene tres argumentos: "cluster1" es un clúster, "cluster2" es un clúster y "weight" es un vector numérico. Calcula la distancia de Canberra entre clústers aplicando los pesos dados. Un ejemplo de ejecución es la siguiente:

```
> cluster1 <- matrix(c(1,2),ncol=2)
> cluster2 <- matrix(c(1,3),ncol=2)
> weight1 <- c(0.4,0.6)
> canberradistanceW(cluster1,cluster2,weight1)
[1] 0.12
> canberradistanceW.details(cluster1,cluster2,weight1)

This function calculates the canberra distance applying some
weight to each element in the clusters.


It allows the algorithm to use some categories more importante
than the others.


Due to there is weight, the formula has to multiply values by
each weight.


Canberra distance is 0.12

[1] 0.12
```

4. *chebyshevDistance(x, y)*: Tiene la opción con *.details*. Tiene dos argumentos, "x" e "y", que son vectores numéricos o una matrices que representan los valores de dos clústers. Calcula la distancia de Chebyshev de dos clústers. Un ejemplo de ejecución es la siguiente:

```
> x <- c(1,2)
> y <- c(1,3)
> chebyshevDistance(x,y)
[1] 1
> chebyshevDistance.details(x,y)
Error in magick_image_readpath(path, density, depth, strip,
defines) :
rsession.exe: UnableToOpenBlob 'C:\Users\atien\OneDrive
- Universidad de Alcala\Documents\man\images\
chebyshevDistance.PNG': No such file or directory
@ error/blob.c/OpenBlob/2924
```

   NOTA IMPORTANTE: Se ha encontrado un error al cargar una imagen PNG en la función *chebyshevDistance.details*. El fichero o directorio no existe. El motivo del error es el mismo de la función *canberradistance.details*, la ruta relativa de la función *initImages* en su código fuente. En el apartado *Examples* de la documentación de la función *chebyshevDistance.details* no aparece un ejemplo de ejecución de esta función sino de la función *chebyshevDistance*.

5. *chebyshevDistanceW(cluster1, cluster2, weight)*: Tiene la opción con *.details*. Tiene tres argumentos: "cluster1" es un clúster, "cluster2" es un clúster y "weight" es un vector numérico. Calcula la distancia de Chebyshev entre clústers aplicando los pesos dados. Un ejemplo de ejecución es la siguiente:

```
> cluster1 <- matrix(c(1,2),ncol=2)
> cluster2 <- matrix(c(1,3),ncol=2)
> weight1 <- c(0.4,0.6)
> chebyshevDistanceW(cluster1,cluster2,weight1)
[1] 0.6
> chebyshevDistanceW.details(cluster1,cluster2,weight1)

This function calculates the chebyshev distance applying some
weight to each element in the clusters.


It allows the algorithm to use some categories more importante
than the others.


Due to there is weight, the formula has to multiply values by
each weight.


Chebyshev distance is 0.6

[1] 0.6
```

6. *clusterDistance(cluster1, cluster2, approach, distance)*: Tiene la opción con *.details*. Tiene cuatro argumentos: "cluster1" es una matriz, "cluster2" es una matriz, "approach" es una cadena que representa el tipo de función a aplicar y "distance" es una cadena que representa el tipo de distancia a utilizar cuyos posibles valores son 'MAX', 'MIN', 'AVG'. Calcula la distancia entre clusters en función del tipo de aproximación y distancia. Un ejemplo de ejecución es la siguiente:

```
> cluster1 <- matrix(c(1,2),ncol=2)
> cluster2 <- matrix(c(1,4),ncol=2)
> (clusterDistance(cluster1,cluster2,'AVG','MAN'))
[1] 2
> clusterDistance.details(cluster1,cluster2,'AVG','MAN')

Using cluster1:

     [,1] [,2]
[1,]    1    2

And cluster2:

     [,1] [,2]
[1,]    1    4

'clusterDistanceByDistance' method calculates the distance
between every element in each cluster.


It uses MAN distance and it calculates the final distance value
depending on AVG approach.
```

```
Cluster distance = 2
```

NOTA IMPORTANTE: En el apartado *Examples* de la documentación de la función *clusterDistance* no aparece un ejemplo de ejecución de esta función sino de la función *clusterDistance.details*.

7. *clusterDistanceByApproach(distances, approach)*: Tiene la opción con *.details*. Tiene dos argumentos: "distances" es un vector numérico y "approach" es una cadena que representa el tipo de función a aplicar cuyos posibles valores son 'MAX', 'MIN', 'AVG'. Calcula la distancia en función de la opción dada. Un ejemplo de ejecución es la siguiente:

```
> distances2 <- c(1:10)
> clusterDistanceByApproach(distances2,'MIN')
[1] 1
> clusterDistanceByApproach.details(distances2,'MIN')

'clusterDistanceByApproach' method calculates the MIN of the
distance vector given.


It returns the MIN of 12345678910


[1] 1
```

NOTA IMPORTANTE: En el apartado *Examples* de la documentación de la función *clusterDistanceByApproach.details* no aparece un ejemplo de ejecución de esta función sino de la función *clusterDistanceByApproach*.

8. *complementaryClusters(components, cluster1, cluster2)*: Tiene la opción con *.details*. Tiene tres argumentos: "components" es una lista de elementos que contiene todos los componentes que deben estar en un clúster o en el otro pero cada elemento solo se puede incluir en un grupo, "cluster1" es un cluster (matriz) y "cluster2" es un cluster (matriz). Comprueba si dos clústeres incluyen todos los elementos pero sin repetir a nadie. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1,2,1,3,1,4,1,5)
> components <- toListDivisive(data)
> cluster1 <- matrix(c(1,2,1,3),ncol=2)
> cluster2 <- matrix(c(1,4,1,5),ncol=2)
> complementaryClusters(components,cluster1,cluster2)
[1] FALSE
> complementaryClusters.details(components,cluster1,cluster2)

'complementaryClusters' checks if clusters are complementary.



Each element from 'components' list has to be in one cluster,
but never included in both.


If every element is in one cluster, the function will return
'TRUE',
```

```
if not, the result will be 'FALSE'



The clusters to be checked are:

     [,1] [,2]
[1,]    1    1
[2,]    2    3
     [,1] [,2]
[1,]    1    1
[2,]    4    5

And they have to include:

[[1]]
     [,1] [,2]
[1,]    1    2

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

[[3]]
     [,1] [,2]
[1,]    1    4

[[4]]
     [,1] [,2]
[1,]    1    5



So, the result is FALSE.


[1] FALSE
```

NOTA IMPORTANTE: En los apartados *Examples* de la documentación de las funciones *complementaryClusters* y *complementaryClusters.details* los ejemplos que aparecen no retornan el valor que se dice. Tanto *complementaryClusters(components,cluster1,cluster2) #TRUE* como *complementaryClusters (components,cluster3,cluster2) #FALSE* retornan *FALSE*. Esto parece ser que es debido a que no se procesan bien las matrices que representan a los clusters. Por ejemplo, "cluster1" que es *matrix(c(1,2,1,3),ncol=2)* debería representar los clusters (1,2) y (1,3) pero en la ejecución de *complementaryClusters(components,cluster1,cluster2)* que se ha realizado en el ejemplo se observa como los clusteres que se usan son (1,1) y (2,3). Una posible solución sería trasponer la matriz "cluster1".

9. *correlationHC(data, target = NULL, weight = c(), distance = "EUC", normalize = TRUE, labels = NULL)*: Tiene la opción con *.details*. Tiene seis argumentos: "data" es una estructura de datos con los datos principales, "target" es una estructura de datos, un vector numérico o una matriz, "weight" es un vector numérico, "distance" es una cadena que representa el tipo de distancia, "nor-

malize" es un parámetro booleano que representa si el usuario quiere normalizar pesos y "labels" es un vector de cadenas para la solución gráfica. Ejecuta el algoritmo de correlación jerárquica aplicando pesos, tipos de distancia, etc. Un ejemplo de dendograma es la Figura 20 y de ejecución es la siguiente:

```
> data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)
> dataFrame <- data.frame(data)
> target1 <- c(1,2,3)
> correlationHC(dataFrame, target1)
$sortedValues
  cluster X1 X2 X3
1       1  1  1  3
2       3  1  2  8
3       5  5  6  4
4       2  2  8  5
5       4  4  9  5

$distances
  cluster sortedDistances
1       1       0.5773503
2       3       2.8867513
3       5       3.3166248
4       2       3.6968455
5       4       4.5460606

$dendrogram
Number of objects: 5


> correlationHC.details(dataFrame, target1)
  Correlation hierarchical function is a classification
  technique that initializes a cluster for each data.


  It calculates the distance between clusters and a target
  given depending on the distance type.


 The function applies weights to each property from main data
 to get weighted results.


 Due to normalized = TRUE, the initial weights change to a
 [0,1] values.


 These are the weight to be used:

[1] 0.3333333 0.3333333 0.3333333

 Initialized data are (more information about how to initialize
 data in 'initData.details'):

[[1]]
     [,1] [,2] [,3]
```

```
[1,]    1    1    3

[[2]]
     [,1] [,2] [,3]
[1,]    2    8    5

[[3]]
     [,1] [,2] [,3]
[1,]    1    2    8

[[4]]
     [,1] [,2] [,3]
[1,]    4    9    5

[[5]]
     [,1] [,2] [,3]
[1,]    5    6    4


 Initialized target is (more information about how to
 initialize data in 'initTarget.details'):

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

 The function calculates the distances between each cluster and
 the target. It applies weights and uses EUC distance type.



 The calculated distances are:

[1] 0.5773503 3.6968455 2.8867513 4.5460606 3.3166248

 The previous distances sorted are:

[1] 0.5773503 2.8867513 3.3166248 3.6968455 4.5460606

 Then, using sorted distances, the function order the clusters.



 Finally, the sorted distances are:

  cluster sortedDistances
1       1       0.5773503
2       3       2.8867513
3       5       3.3166248
4       2       3.6968455
5       4       4.5460606

 The sorted clusters are:

  cluster X1 X2 X3
```

```
1       1  1  1  3
2       3  1  2  8
3       5  5  6  4
4       2  2  8  5
5       4  4  9  5

 And the final dendrogram is on the image.


$sortedValues
  cluster X1 X2 X3
1       1  1  1  3
2       3  1  2  8
3       5  5  6  4
4       2  2  8  5
5       4  4  9  5

$distances
  cluster sortedDistances
1       1       0.5773503
2       3       2.8867513
3       5       3.3166248
4       2       3.6968455
5       4       4.5460606

$dendrogram
Number of objects: 5
```

10. *distances(cluster1, cluster2, distance, weight)*: Tiene la opción con *.details*. Tiene cuatro argumentos: "cluster1" es una matriz, "cluster2" es una matriz, "distance" es una cadena que representa el tipo de distancia que se aplicará y "weight" es un vector numérico. Calcula las distancias entre dos clusters aplicando pesos en función del tipo de distancia. Un ejemplo de ejecución es la siguiente:

```
> cluster1 <- matrix(c(2,3))
> cluster2 <- matrix(c(4,5))
> weight1 <- c(0.6,0.4)
> distances(cluster1, cluster2, 'MAN', weight1)
[1] 1.2
> distances.details(cluster1, cluster2, 'MAN', weight1)

 This function calculates MAN distance applying weights.



 It calculates the distance between:.

  cluster1
1        2
2        3


  cluster2
1        4
```

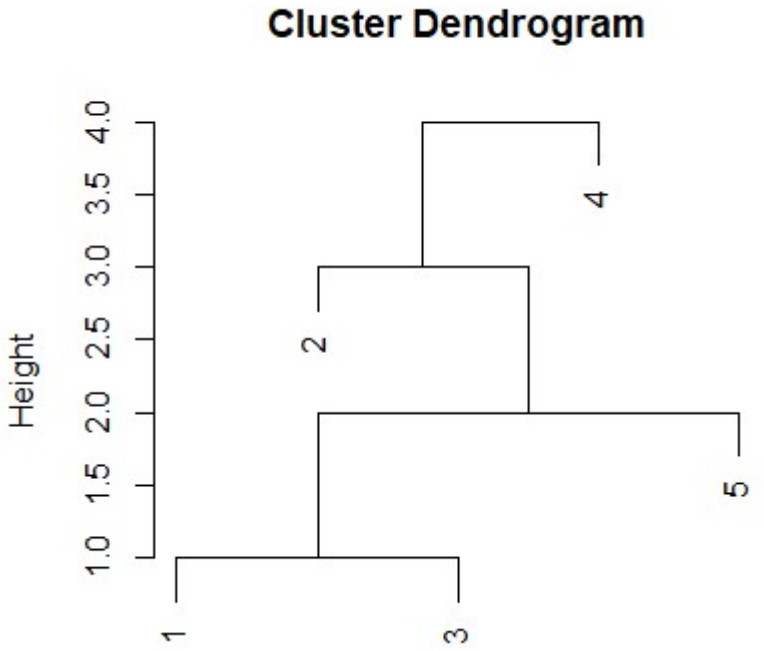


Figure 20: Dendograma de la ejecución de *correlationHC(dataFrame, target1)* y *correlationHC.details(dataFrame, target1)*.

```
2       5

 Applying these weights:

[1] 0.6 0.4

 The distance value is: 1.2.


[1] 1.2
```

11. *divisiveHC(data, distance, approach)*: Tiene la opción con *.details*. Tiene tres argumentos: "data" puede ser un vector numérico, una matriz o una estructura de datos que será transformado a una matriz y una lista para ser usados, "distance" es una cadena que representa la distancia a usar y "approach" es una cadena que representa el enfoque que utilizar. Ejecuta el algoritmo de clusterización jerárquica divisivo eligiendo los tipos de distancia y enfoque. Un ejemplo de gráfica de salida es la Figura 21 y de ejecución es la siguiente:

```
> a <- c(1,2,1,3,1,4,1,5,1,6)
> divisiveHC(a,'EUC','MAX')
[[1]]
  X1 X2
1  1  2
2  1  3
3  1  4
4  1  5
5  1  6

[[2]]
  X1 X2
1  1  2

[[3]]
  X1 X2
1  1  3
2  1  4
3  1  5
4  1  6

[[4]]
  X1 X2
1  1  3

[[5]]
  X1 X2
1  1  4
2  1  5
3  1  6

[[6]]
  X1 X2
1  1  4

[[7]]
```

```
  X1 X2
1  1  5
2  1  6

[[8]]
  X1 X2
1  1  5

[[9]]
  X1 X2
1  1  6

> divisiveHC.details(a,'EUC','MAX')

  Divisive hierarchical clustering is a classification
  technique that initializes a cluster with every data.


  It calculates the distance between datas depending on the
  approach type given and


 it divides the most different clusters until any cluster can
 be divided again.


  These are the clusters with only one element:

[[1]]
  X1 X2
1  1  2

[[2]]
  X1 X2
1  1  3

[[3]]
  X1 X2
1  1  4

[[4]]
  X1 X2
1  1  5

[[5]]
  X1 X2
1  1  6


  And this is the initial cluster with every element:

  X1 X2
1  1  2
2  1  3
```

```
3  1  4
4  1  5
5  1  6

  In each step:
   - It calculates a matrix distance between valid clusters
   depending on the approach and the distance types.

   - It gets the maximal distance value from the matrixes of
   every cluster. We have to look for the most different
   clusters available  in every cluster.

   - It divides the selected cluster in two new and
   complementary clusters using tha maximal distance between
   clusters.

   - It repeats these steps while there isn't any cluster that
   can be divided again.


-----------------------------------------------------------------

STEP => 1


 The algorithm calculates a matrix distance from every active
 cluster, but it has to find the maximal value from all matrix
 and then chooses which is the selected one.


 It gets the maximal value from every matrix and then chooses
 the maximal between them. So...



 Matrix Distance (distance type = EUC, approach type = MAX):

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
 [1,]    0    0    0    0    0    0    0    0    0     0     0
 [2,]    0    0    0    0    0    0    0    0    0     0     0
 [3,]    0    0    0    0    0    0    0    0    0     0     0
 [4,]    0    0    0    0    0    0    0    0    0     0     0
 [5,]    0    0    0    0    0    0    0    0    0     0     0
 [6,]    0    0    0    0    0    0    0    0    0     0     0
 [7,]    0    0    0    0    0    0    0    0    0     0     0
 [8,]    0    0    0    0    0    0    0    0    0     0     0
 [9,]    0    0    0    0    0    0    0    0    0     0     0
[10,]    0    0    0    0    0    0    0    0    0     0     0
[11,]    0    0    0    0    0    0    0    0    0     0     0
[12,]    0    0    0    0    0    0    0    0    0     0     0
[13,]    0    0    0    0    0    0    0    0    0     0     0
[14,]    0    0    0    0    0    0    0    0    0     0     0
[15,]    0    0    0    0    0    0    0    0    0     0     0
[16,]    0    0    0    0    0    0    0    0    0     0     0
```

```
[17,]    0    0    0    0    0    0    0    0    0     0     0
[18,]    0    0    0    0    0    0    0    0    0     0     0
[19,]    0    0    0    0    0    0    0    0    0     0     0
[20,]    0    0    0    0    0    0    0    0    0     0     3
[21,]    0    0    0    0    0    0    0    0    0     3     0
[22,]    0    0    0    0    0    0    0    0    3     0     0
[23,]    0    0    0    0    0    0    0    4    0     0     0
[24,]    0    0    0    0    0    0    4    0    0     0     0
[25,]    0    0    0    0    0    4    0    0    0     0     0
[26,]    0    0    0    0    4    0    0    0    0     0     0
[27,]    0    0    0    3    0    0    0    0    0     0     0
[28,]    0    0    2    0    0    0    0    0    0     0     0
[29,]    0    3    0    0    0    0    0    0    0     0     0
[30,]    4    0    0    0    0    0    0    0    0     0     0
[31,]    0    0    0    0    0    0    0    0    0     0     0
      [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21]
 [1,]     0     0     0     0     0     0     0     0     0     0
 [2,]     0     0     0     0     0     0     0     0     0     0
 [3,]     0     0     0     0     0     0     0     0     0     0
 [4,]     0     0     0     0     0     0     0     0     0     0
 [5,]     0     0     0     0     0     0     0     0     0     0
 [6,]     0     0     0     0     0     0     0     0     0     0
 [7,]     0     0     0     0     0     0     0     0     0     0
 [8,]     0     0     0     0     0     0     0     0     0     0
 [9,]     0     0     0     0     0     0     0     0     0     0
[10,]     0     0     0     0     0     0     0     0     0     3
[11,]     0     0     0     0     0     0     0     0     3     0
[12,]     0     0     0     0     0     0     0     4     0     0
[13,]     0     0     0     0     0     0     3     0     0     0
[14,]     0     0     0     0     0     4     0     0     0     0
[15,]     0     0     0     0     4     0     0     0     0     0
[16,]     0     0     0     4     0     0     0     0     0     0
[17,]     0     0     4     0     0     0     0     0     0     0
[18,]     0     3     0     0     0     0     0     0     0     0
[19,]     4     0     0     0     0     0     0     0     0     0
[20,]     0     0     0     0     0     0     0     0     0     0
[21,]     0     0     0     0     0     0     0     0     0     0
[22,]     0     0     0     0     0     0     0     0     0     0
[23,]     0     0     0     0     0     0     0     0     0     0
[24,]     0     0     0     0     0     0     0     0     0     0
[25,]     0     0     0     0     0     0     0     0     0     0
[26,]     0     0     0     0     0     0     0     0     0     0
[27,]     0     0     0     0     0     0     0     0     0     0
[28,]     0     0     0     0     0     0     0     0     0     0
[29,]     0     0     0     0     0     0     0     0     0     0
[30,]     0     0     0     0     0     0     0     0     0     0
[31,]     0     0     0     0     0     0     0     0     0     0
      [,22] [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31]
 [1,]     0     0     0     0     0     0     0     0     4     0
 [2,]     0     0     0     0     0     0     0     3     0     0
 [3,]     0     0     0     0     0     0     2     0     0     0
 [4,]     0     0     0     0     0     3     0     0     0     0
 [5,]     0     0     0     0     4     0     0     0     0     0
 [6,]     0     0     0     4     0     0     0     0     0     0
```

```
 [7,]     0     0     4     0     0     0     0     0     0     0
 [8,]     0     4     0     0     0     0     0     0     0     0
 [9,]     3     0     0     0     0     0     0     0     0     0
[10,]     0     0     0     0     0     0     0     0     0     0
[11,]     0     0     0     0     0     0     0     0     0     0
[12,]     0     0     0     0     0     0     0     0     0     0
[13,]     0     0     0     0     0     0     0     0     0     0
[14,]     0     0     0     0     0     0     0     0     0     0
[15,]     0     0     0     0     0     0     0     0     0     0
[16,]     0     0     0     0     0     0     0     0     0     0
[17,]     0     0     0     0     0     0     0     0     0     0
[18,]     0     0     0     0     0     0     0     0     0     0
[19,]     0     0     0     0     0     0     0     0     0     0
[20,]     0     0     0     0     0     0     0     0     0     0
[21,]     0     0     0     0     0     0     0     0     0     0
[22,]     0     0     0     0     0     0     0     0     0     0
[23,]     0     0     0     0     0     0     0     0     0     0
[24,]     0     0     0     0     0     0     0     0     0     0
[25,]     0     0     0     0     0     0     0     0     0     0
[26,]     0     0     0     0     0     0     0     0     0     0
[27,]     0     0     0     0     0     0     0     0     0     0
[28,]     0     0     0     0     0     0     0     0     0     0
[29,]     0     0     0     0     0     0     0     0     0     0
[30,]     0     0     0     0     0     0     0     0     0     0
[31,]     0     0     0     0     0     0     0     0     0     0

 The maximal distance is: 4


 The most distant clusters are:

  X1 X2
1  1  2


  X1 X2
1  1  3
2  1  4
3  1  5
4  1  6

 The divided clusters are added to the solution.


 - The second divided cluster is added to the active clusters
 list because it can be divided again.


 If the divided clusters can't be divided again, then they
 don't be added to the active clusters.


------------------------------------------------------------------
```

```
STEP => 2


 The algorithm calculates a matrix distance from every active
 cluster, but it has to find the maximal value from all matrix
 and then chooses which is the selected one.


 It gets the maximal value from every matrix and then chooses
 the maximal between them. So...



 Matrix Distance (distance type = EUC, approach type = MAX):

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
 [1,]    0    0    0    0    0    0    0    0    0     0     0
 [2,]    0    0    0    0    0    0    0    0    0     0     0
 [3,]    0    0    0    0    0    0    0    0    0     0     0
 [4,]    0    0    0    0    0    0    0    0    0     0     3
 [5,]    0    0    0    0    0    0    0    0    0     3     0
 [6,]    0    0    0    0    0    0    0    0    3     0     0
 [7,]    0    0    0    0    0    0    0    2    0     0     0
 [8,]    0    0    0    0    0    0    2    0    0     0     0
 [9,]    0    0    0    0    0    3    0    0    0     0     0
[10,]    0    0    0    0    3    0    0    0    0     0     0
[11,]    0    0    0    3    0    0    0    0    0     0     0
[12,]    0    0    2    0    0    0    0    0    0     0     0
[13,]    0    2    0    0    0    0    0    0    0     0     0
[14,]    3    0    0    0    0    0    0    0    0     0     0
[15,]    0    0    0    0    0    0    0    0    0     0     0
      [,12] [,13] [,14] [,15]
 [1,]     0     0     3     0
 [2,]     0     2     0     0
 [3,]     2     0     0     0
 [4,]     0     0     0     0
 [5,]     0     0     0     0
 [6,]     0     0     0     0
 [7,]     0     0     0     0
 [8,]     0     0     0     0
 [9,]     0     0     0     0
[10,]     0     0     0     0
[11,]     0     0     0     0
[12,]     0     0     0     0
[13,]     0     0     0     0
[14,]     0     0     0     0
[15,]     0     0     0     0

 The maximal distance is: 3


 The most distant clusters are:

  X1 X2
```

```
1  1  3


  X1 X2
1  1  4
2  1  5
3  1  6
```

The divided clusters are added to the solution.

- The second divided cluster is added to the active clusters list because it can be divided again.

If the divided clusters can't be divided again, then they don't be added to the active clusters.

STEP => 3

The algorithm calculates a matrix distance from every active cluster, but it has to find the maximal value from all matrix and then chooses which is the selected one.

It gets the maximal value from every matrix and then chooses the maximal between them. So...

Matrix Distance (distance type = EUC, approach type = MAX):

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0    0    0    0    0    2    0
[2,]    0    0    0    0    1    0    0
[3,]    0    0    0    2    0    0    0
[4,]    0    0    2    0    0    0    0
[5,]    0    1    0    0    0    0    0
[6,]    2    0    0    0    0    0    0
[7,]    0    0    0    0    0    0    0
```

The maximal distance is: 2

The most distant clusters are:

```
  X1 X2
1  1  4
```

```
  X1 X2
1  1  5
2  1  6
```

The divided clusters are added to the solution.

- The second divided cluster is added to the active clusters list because it can be divided again.

If the divided clusters can't be divided again, then they don't be added to the active clusters.

STEP => 4

The algorithm calculates a matrix distance from every active cluster, but it has to find the maximal value from all matrix and then chooses which is the selected one.

It gets the maximal value from every matrix and then chooses the maximal between them. So...

Matrix Distance (distance type = EUC, approach type = MAX):

```
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    1    0    0
[3,]    0    0    0
```

The maximal distance is: 1

The most distant clusters are:

```
  X1 X2
1  1  5
```

```
  X1 X2
1  1  6
```

The divided clusters are added to the solution.

If the divided clusters can't be divided again, then they don't be added to the active clusters.

```
This loop has been repeated until there aren't any active
cluster, that is, any cluster can be divided again.

[[1]]
     [,1] [,2]
[1,]    1    2
[2,]    1    3
[3,]    1    4
[4,]    1    5
[5,]    1    6

[[2]]
     [,1] [,2]
[1,]    1    2

[[3]]
     [,1] [,2]
[1,]    1    3
[2,]    1    4
[3,]    1    5
[4,]    1    6

[[4]]
     [,1] [,2]
[1,]    1    3

[[5]]
     [,1] [,2]
[1,]    1    4
[2,]    1    5
[3,]    1    6

[[6]]
     [,1] [,2]
[1,]    1    4

[[7]]
     [,1] [,2]
[1,]    1    5
[2,]    1    6

[[8]]
     [,1] [,2]
[1,]    1    5

[[9]]
     [,1] [,2]
[1,]    1    6
```

12. *edistance(x, y)*: Tiene la opción con *.details*. Tiene dos argumentos: "x" e "y" que ambos son un vector numérico o una matriz que representa los valores de un cluster. Calcula la distancia Euclídea de dos clusters. Un ejemplo de ejecución es la siguiente:

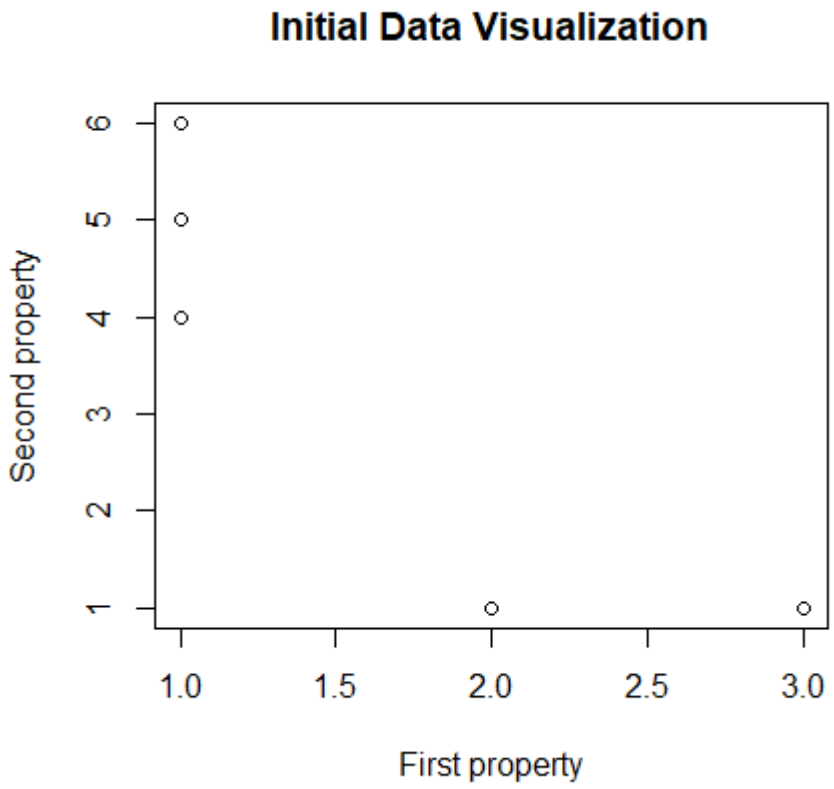


Figure 21: Gráfica de salida de *divisiveHC(a,'EUC','MAX')* y *divisiveHC.details (a,'EUC','MAX')*.

```
> x <- c(1,2)
> y <- c(1,3)
> edistance(x,y)
[1] 1
> edistance.details(x,y)
Error in magick_image_readpath(path, density, depth, strip,
defines) :
  rsession.exe: UnableToOpenBlob 'C:\Users\atien\OneDrive -
  Universidad de Alcala\Documents\GitHub\FCD\PL2\Parte 2\man\
  images\euclideanDistance.PNG': No such file or directory
  @ error/blob.c/OpenBlob/2924
```

NOTA IMPORTANTE: Se ha encontrado un error al cargar una imagen PNG en la función *edistance.details*. El fichero o directorio no existe. El motivo del error es el mismo de la función *canberradistance.details*, la ruta relativa de la función *initImages* en su código fuente. En el apartado *Examples* de la documentación de la función *edistance.details* no aparece un ejemplo de ejecución de esta función sino de la función *edistance*.

13. *edistanceW(cluster1, cluster2, weight)*: Tiene la opción con *.details*. Tiene tres argumentos: "cluster1" es un cluster, "cluster2" es un cluster y "weight" es un vector numérico. Calcula la distancia Euclídea entre clusters aplicando los pesos dados. Un ejemplo de ejecución es la siguiente:

```
> cluster1 <- matrix(c(1,2),ncol=2)
> cluster2 <- matrix(c(1,3),ncol=2)
> weight1 <- c(0.4,0.6)
> edistanceW(cluster1,cluster2,weight1)
[1] 0.7745967
> edistanceW.details(cluster1,cluster2,weight1)

 This function calculates the euclidean distance applying some
 weight to each element in the clusters.


 It allows the algorithm to use some categories more importante
 than the others.


 Due to there is weight, the formula has to multiply values by
 each weight.


 Euclidean distance is 0.774596669241483

[1] 0.7745967
```

14. *getCluster(distance, matrix)*: Tiene la opción con *.details*. Tiene dos argumentos: "distance" es un número que debería estar en la matriz y "matrix" es una matriz numérica. Devuelve los clusters con el mínimo valor de distancia; usando la distancia dada devuelve el índice de la matriz. Un ejemplo de ejecución es la siguiente:

```
> matrixExample <- matrix(c(1:10), ncol=2)
> (getCluster(2,matrixExample))
[1] 1 2
> getCluster.details(2,matrixExample)
```

```
 'getCluster' method searches the clusters which have the
 distance given.

 Search for 2 in:

     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10

 The clusters with the minimum distance are: 1, 2

[1] 1 2
```

15. *getClusterDivisive(distance, vector)*: Tiene la opción con *.details*. Tiene dos argumentos: "distance" es un número que debería estar en la matriz y "vector" es un vector numérico. Devuelve los clusters con el máximo valor de distancia; usando la distancia dada devuelve el índice de la matriz. Un ejemplo de ejecución es la siguiente:

```
> getClusterDivisive(2,c(1:10))
[1] 2
> getClusterDivisive.details(2,c(1:10))

 'getCluster' method searches the cluster which have the
 distance to the target given.


 Search for 2 in:

 [1]  1  2  3  4  5  6  7  8  9 10

 The cluster with the minimum distance is: 2

[1] 2
```

NOTA IMPORTANTE: En el apartado *Examples* de la documentación de la función *getClusterDivisive.details* el segundo ejemplo no corresponde a esa función sino a la función *getClusterDivisive*.

16. *initClusters(initList)*: Tiene la opción con *.details*. Tiene un argumento: "initList" es una lista de clusters que contendrá clusters con un elemento. Inicializa los clusters para el algoritmo divisivo. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1:8)
> listData <- toListDivisive(data)
> initClusters(listData)
[[1]]
     [,1] [,2]
[1,]    1    2

[[2]]
     [,1] [,2]
```

```
[1,]    3    4

[[3]]
     [,1] [,2]
[1,]    5    6

[[4]]
     [,1] [,2]
[1,]    7    8

[[5]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4

[[6]]
     [,1] [,2]
[1,]    1    2
[2,]    5    6

[[7]]
     [,1] [,2]
[1,]    1    2
[2,]    7    8

[[8]]
     [,1] [,2]
[1,]    3    4
[2,]    5    6

[[9]]
     [,1] [,2]
[1,]    3    4
[2,]    7    8

[[10]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8

[[11]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6

[[12]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    7    8

[[13]]
     [,1] [,2]
```

```
[1,]    1    2
[2,]    5    6
[3,]    7    8

[[14]]
     [,1] [,2]
[1,]    3    4
[2,]    5    6
[3,]    7    8

[[15]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

> initClusters.details(listData)

 'initClusters' method initializes the clusters used in the
 divisive algorithm.


 To know which are the most different clusters, we need to know
 the distance between every possible clusters that could be
 created with the initial elements.


 This step is the most computationally complex, so it will make
 the algorithm to get the solution with delay, or even, not to
 find a solution because of the computers capacities.


 The clusters created using

[[1]]
     [,1] [,2]
[1,]    1    2

[[2]]
     [,1] [,2]
[1,]    3    4

[[3]]
     [,1] [,2]
[1,]    5    6

[[4]]
     [,1] [,2]
[1,]    7    8
```

```
 are:

[[1]]
     [,1] [,2]
[1,]    1    2

[[2]]
     [,1] [,2]
[1,]    3    4

[[3]]
     [,1] [,2]
[1,]    5    6

[[4]]
     [,1] [,2]
[1,]    7    8

[[5]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4

[[6]]
     [,1] [,2]
[1,]    1    2
[2,]    5    6

[[7]]
     [,1] [,2]
[1,]    1    2
[2,]    7    8

[[8]]
     [,1] [,2]
[1,]    3    4
[2,]    5    6

[[9]]
     [,1] [,2]
[1,]    3    4
[2,]    7    8

[[10]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8

[[11]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

```
[[12]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    7    8

[[13]]
     [,1] [,2]
[1,]    1    2
[2,]    5    6
[3,]    7    8

[[14]]
     [,1] [,2]
[1,]    3    4
[2,]    5    6
[3,]    7    8

[[15]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

[[1]]
     [,1] [,2]
[1,]    1    2

[[2]]
     [,1] [,2]
[1,]    3    4

[[3]]
     [,1] [,2]
[1,]    5    6

[[4]]
     [,1] [,2]
[1,]    7    8

[[5]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4

[[6]]
     [,1] [,2]
[1,]    1    2
[2,]    5    6

[[7]]
     [,1] [,2]
[1,]    1    2
```

```
[2,]    7    8

[[8]]
     [,1] [,2]
[1,]    3    4
[2,]    5    6

[[9]]
     [,1] [,2]
[1,]    3    4
[2,]    7    8

[[10]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8

[[11]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6

[[12]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    7    8

[[13]]
     [,1] [,2]
[1,]    1    2
[2,]    5    6
[3,]    7    8

[[14]]
     [,1] [,2]
[1,]    3    4
[2,]    5    6
[3,]    7    8

[[15]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```

17. *initData(data)*: Tiene la opción con *.details*. Tiene un argumento: "data" es una estructura de datos con los datos principales. Inicializa los datos en el algoritmo de correlación jerárquica. Un ejemplo de ejecución es la siguiente:

```
> data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)
> dataFrame <- data.frame(data)
> initData(dataFrame)
```

```
[[1]]
     [,1] [,2] [,3]
[1,]    1    1    3

[[2]]
     [,1] [,2] [,3]
[1,]    2    8    5

[[3]]
     [,1] [,2] [,3]
[1,]    1    2    8

[[4]]
     [,1] [,2] [,3]
[1,]    4    9    5

[[5]]
     [,1] [,2] [,3]
[1,]    5    6    4

> initData.details(dataFrame)

 This function initializes the input data creating a cluster
 with each row of the data frame.


 It gets this data from the user:

  X1 X2 X3
1  1  1  3
2  2  8  5
3  1  2  8
4  4  9  5
5  5  6  4

 Each cluster will be a matrix with a row and the same columns
 as the initial data frame.


 Initialized data will be:

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

[[2]]
     [,1] [,2] [,3]
[1,]    2    8    5

[[3]]
     [,1] [,2] [,3]
[1,]    1    2    8

[[4]]
```

```
     [,1] [,2] [,3]
[1,]    4    9    5

[[5]]
     [,1] [,2] [,3]
[1,]    5    6    4

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

[[2]]
     [,1] [,2] [,3]
[1,]    2    8    5

[[3]]
     [,1] [,2] [,3]
[1,]    1    2    8

[[4]]
     [,1] [,2] [,3]
[1,]    4    9    5

[[5]]
     [,1] [,2] [,3]
[1,]    5    6    4
```

18. *initImages(path)*: Tiene un argumento: "path" es la ruta de un archivo. Es una función auxiliar para mostrar una imagen.

19. *initTarget(target, data)*: Tiene la opción con *.details*. Tiene dos argumentos: "target" es un vector numérico, una matriz o una estructura de datos y "data" es una estructura de datos con los datos principales. Inicializa el objetivo del algoritmo de correlación jerárquica comprobando si el objetivo es válido y si no lo es, lo inicializa. Un ejemplo de ejecución es la siguiente:

```
> data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)
> dataFrame <- data.frame(data)
> target1 <- matrix(c(2,3))
> initTarget(target1,dataFrame)

 invalid Target!


     [,1] [,2] [,3]
[1,]    0    0    0
> initTarget.details(target1,dataFrame)

 This function initializes the target and checks if it is a
 valid target.


 It gets this target from the user:

     [,1]
[1,]    2
```

```
[2,]    3

 After transforming the target into matrix, it checks if it is
 acceptable


 If the target does not have the same columns as main data or
 more than one row,
 it is not a valid target!


 The function will initialize as a '0's matrix.


 Target used will be:

     [,1] [,2] [,3]
[1,]    0    0    0
     [,1] [,2] [,3]
[1,]    0    0    0
```

20. *matrixDistance(list, distance)*: Tiene dos argumentos: "list" es una lista de clusters y "distance" es un literal (cadena). Calcula la matriz de distancias usando el tipo de distancia. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1:10)
> clusters <- toList(data)
> (matrixDistance(clusters, 'EUC'))
          [,1]     [,2]     [,3]     [,4]      [,5]
[1,]  0.000000 2.828427 5.656854 8.485281 11.313708
[2,]  2.828427 0.000000 2.828427 5.656854  8.485281
[3,]  5.656854 2.828427 0.000000 2.828427  5.656854
[4,]  8.485281 5.656854 2.828427 0.000000  2.828427
[5,] 11.313708 8.485281 5.656854 2.828427  0.000000
```

21. *maxDistance(matrix)*: Tiene la opción con *.details*. Tiene un argumento: "matrix" es una matriz numérica pero puede ser un vector numérico. Devuelve el máximo valor de la matriz. Un ejemplo de ejecución es la siguiente:

```
> maxDistance(1:10)
[1] 10
> maxDistance.details(1:10)

 'maxDistance' function gets the maximal value from a matrix.


 It returns the maximal value avoiding 0 values.


 [1]  1  2  3  4  5  6  7  8  9 10


 10 is the maximal value of the matrix.

[1] 10
```

22. *mdAgglomerative(list, distance, approach)*: Tiene la opción con *.details*. Tiene tres argumentos: "list" es una lista de clusters, "distance" es un literal que representa el tipo de distancia que se va a usar y "approach" es un literal que representa el tipo de enfoque que se va a usar. Calcula la matriz de distancias usando los tipos de distancia y enfoque. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1,2,1,3,1,4,1,5,1,6)
> clusters <- toList(data)
> (mdAgglomerative(clusters, 'EUC', 'MAX'))
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    2    3    4
[2,]    1    0    1    2    3
[3,]    2    1    0    1    2
[4,]    3    2    1    0    1
[5,]    4    3    2    1    0
> mdAgglomerative.details(clusters, 'EUC', 'MAX')

 'mdAgglomerative' creates the matrix distance of every cluster
 given by 'list' parameter.


 It usesEUCdistance andMAXapproach.


 It returns the matrix with all the distances between clusters
 depending on distance and approach.


 The matrix distance is:


     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    2    3    4
[2,]    1    0    1    2    3
[3,]    2    1    0    1    2
[4,]    3    2    1    0    1
[5,]    4    3    2    1    0
```

23. *mdDivisive(list, distance, approach, components)*: Tiene la opción con *.details*. Tiene cuatro argumentos: "list" es una lista de clusters, "distance" es un literal que representa el tipo de distancia que se va a usar, "approach" es un literal que representa el tipo de enfoque que se va a usar y "components" es una lista de clusters que contiene cada cluster con un solo elemento y se usa para saber si la condición complementaria es *TRUE*. Calcula la matriz de distancias usando los tipos de distancia y enfoque. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1,2,1,3,1,4,1,5,1,6)
> clusters <- toList(data)
> components <- toList(data)
> (mdDivisive(clusters, 'EUC', 'MAX', components))
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0
```

```
There were 50 or more warnings (use warnings() to see the first
50)
> mdDivisive.details(clusters, 'EUC', 'MAX', components)

 'mdDivisive' creates a matrix distance including every cluster
 from 'list'.


 The function checks if the clusters are not valid, if they are
 the same cluster and


 if the clusters are not complementary. It will allocate a 0
 value if any condition is not 'TRUE'.


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

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

[[3]]
     [,1] [,2] [,3]
[1,]    1    4    1

[[4]]
     [,1] [,2] [,3]
[1,]    1    5    1

[[5]]
     [,1] [,2] [,3]
[1,]    1    6    1


 The matrix distance for the list above is:


     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0
There were 50 or more warnings (use warnings() to see the first
50)
```

NOTA IMPORTANTE: Las ejecuciones anteriores de las funciones *mdDivisive* y *mdDivisive.details* muestran mensajes de warnings, concretamente 50 o más. Todos ellos son iguales: *In matrix(clusterElement, ncol = 2): data length [3] is not a sub-multiple or multiple of the number of rows [2].*

24. *mdistance(x, y)*: Tiene la opción con *.details*. Tiene dos argumentos: "x" e "y"

son un vector numérico o una matriz que representan los valores de dos clusters. Calcula la distancia Manhattan de dos clusters. Un ejemplo de ejecución es la siguiente:

```
> x <- c(1,2)
> y <- c(1,3)
> mdistance(x,y)
[1] 1
> mdistance.details(x,y)
Error in magick_image_readpath(path, density, depth, strip,
defines) :
  rsession.exe: UnableToOpenBlob 'C:\Users\atien\OneDrive -
  Universidad de Alcala\Documents\GitHub\FCD\PL2\Parte 2\man\
  images\manhattanDistance.PNG': No such file or directory
  @ error/blob.c/OpenBlob/2924
```

NOTA IMPORTANTE: Se ha encontrado un error al cargar una imagen PNG en la función *mdistance.details*. El fichero o directorio no existe. El motivo del error es el mismo de la función *canberradistance.details*, la ruta relativa de la función *initImages* en su código fuente. En el apartado *Examples* de la documentación de la función *mdistance.details* no aparece un ejemplo de ejecución de esta función sino de la función *mdistance*.

25. *mdistanceW(cluster1, cluster2, weight)*: Tiene la opción con *.details*. Tiene tres argumentos: "cluster1" es un cluster, "cluster2" es un cluster y "weight" es un vector numérico. Calcula la distancia Manhattan entre clusters aplicando los pesos dados. Un ejemplo de ejecución es la siguiente:

```
> cluster1 <- matrix(c(1,2),ncol=2)
> cluster2 <- matrix(c(1,3),ncol=2)
> weight1 <- c(0.4,0.6)
> mdistanceW(cluster1,cluster2,weight1)
[1] 0.6
> mdistanceW.details(cluster1,cluster2,weight1)

 This function calculates the manhattan distance applying some
 weight to each element in the clusters.


 It allows the algorithm to use some categories more importante
 than the others.


 Due to there is weight, the formula has to multiply values by
 each weight.


 Manhattan distance is 0.6

[1] 0.6
```

26. *minDistance(matrix)*: Tiene la opción con *.details*. Tiene un argumento: "matrix" es una matriz numérica pero puede ser un vector numérico. Devuelve el mínimo valor de la matriz. Un ejemplo de ejecución es la siguiente:

```
> minDistance(1:10)
[1] 1
```

```
> minDistance.details(1:10)

 'minDistance' function gets the minimal value from a matrix.


 It returns the minimal value avoiding 0 values.


 [1]  1  2  3  4  5  6  7  8  9 10


 1 is the minimal value of the matrix.

[1] 1
```

NOTA IMPORTANTE: En el apartado *Examples* de la documentación de la función *minDistance.details* el primer ejemplo no pertenece a esta función sino a la función *minDistance*.

27. *newCluster(list, clusters)*: Tiene la opción con *.details*. Tiene dos argumentos: "list" es la lista de cluster genérica y "clusters" es un vector con los índices de los clusters en la matriz. Crea un cluster formado por dos clusters dados y añade el nuevo cluster a la lista. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1:10)
> list <- toList(data)
> clusters <- c(1,2)
> newCluster(list,clusters)
[[1]]
     [,1] [,2] [,3]
[1,]    1    2    0

[[2]]
     [,1] [,2] [,3]
[1,]    3    4    0

[[3]]
     [,1] [,2] [,3]
[1,]    5    6    1

[[4]]
     [,1] [,2] [,3]
[1,]    7    8    1

[[5]]
     [,1] [,2] [,3]
[1,]    9   10    1

[[6]]
     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    3    4    1

> newCluster.details(list,clusters)

 'newCluster' function creates a new cluster from the clusters
```

```
given.


It adds the new cluster to 'list' and disables the clusters
used to create the new one.


Using 1 and 2 it searches the clusters in 'list' parameter and
creates a new one with their components.


After the new cluster is created, the initial clusters must be
disabled.


The new cluster:


     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    3    4    1


is added to the list:


[[1]]
     [,1] [,2] [,3]
[1,]    1    2    0

[[2]]
     [,1] [,2] [,3]
[1,]    3    4    0

[[3]]
     [,1] [,2] [,3]
[1,]    5    6    1

[[4]]
     [,1] [,2] [,3]
[1,]    7    8    1

[[5]]
     [,1] [,2] [,3]
[1,]    9   10    1

[[6]]
     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    3    4    1
```

28. *normalizeWeight(normalize, weight, data)*: Tiene la opción con *.details*. Tiene tres argumentos: "normalize" es un valor booleano, "weight" es un vector numérico y "data" es una estructura de datos. Normaliza el peso de los valores si "normalize" es *TRUE*. Un ejemplo de ejecución es la siguiente:

```
> data <- data.frame(matrix(c(1:10),ncol = 2))
> weight1 <- c(2,4)
> normalizeWeight(FALSE, weight1, data)
[1] 2 4
> normalizeWeight(TRUE, weight1, data)
[1] 0.3333333 0.6666667
> normalizeWeight.details(FALSE, weight1, data)

 This function normalizes weight values.


 This are the initial weights:

  2
  4

 It checks if there is a weights vector. If not, it creates a
 vector with 2 '1's.


 Due to the fact that 'normalize' = FALSE, weight vector does
 not change.


 These are the new weights:

  2
  4
[1] 2 4
> normalizeWeight.details(TRUE, weight1, data)

 This function normalizes weight values.


 This are the initial weights:

  2
  4

 It checks if there is a weights vector. If not, it creates a
 vector with 2 '1's.


 Due to the fact that 'normalize' = TRUE, weight vector changes
 every weight as the initial value divided between the total sum
 of the vector.


 FinalWeight[i] = WeightValue[i]/TotalSum


 These are the new weights:

  0.333333333333333
```

```
  0.666666666666667
[1] 0.3333333 0.6666667
```

29. *octileDistance(x, y)*: Tiene la opción con *.details*. Tiene dos argumentos: "x" e "y" son un vector numérico o una matriz que representan los valores de dos clusters. Calcula la distancia octil de dos clusters. Un ejemplo de ejecución es la siguiente:

```
> x <- c(1,2)
> y <- c(1,3)
> octileDistance(x,y)
[1] 1
> octileDistance.details(x,y)
[1] 1
```

NOTA IMPORTANTE: La ejecución de la función *octileDistance* y *octileDistance.details* son iguales. La segunda no explica nada nuevo con respecto a la primera.

30. *octileDistanceW(cluster1, cluster2, weight)*: Tiene la opción con *.details*. Tiene tres argumentos: "cluster1" es un cluster, "cluster2" es un cluster y "weight" es un vector numérico. Calcula la distancia octil entre clusters aplicando los pesos dados. Un ejemplo de ejecución es la siguiente:

```
> cluster1 <- matrix(c(1,2),ncol=2)
> cluster2 <- matrix(c(1,3),ncol=2)
> weight1 <- c(0.4,0.6)
> octileDistanceW(cluster1,cluster2,weight1)
[1] 0.6
> octileDistanceW.details(cluster1,cluster2,weight1)

 This function calculates the octile distance applying some
 weight to each element in the clusters.


 It allows the algorithm to use some categories more importante
 than the others.


 Due to there is weight, the formula has to multiply values by
 each weight.


 Octile distance is 0.6

[1] 0.6
```

31. *toList(data)*: Tiene la opción con *.details*. Tiene un argumento: "data" puede ser un vector numérico, una matriz o una estructura de datos numérico. Transforma datos a lista. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1:10)
> toList(data)
[[1]]
     [,1] [,2] [,3]
[1,]    1    2    1

[[2]]
```

```
     [,1] [,2] [,3]
[1,]    3    4    1

[[3]]
     [,1] [,2] [,3]
[1,]    5    6    1

[[4]]
     [,1] [,2] [,3]
[1,]    7    8    1

[[5]]
     [,1] [,2] [,3]
[1,]    9   10    1

> toList.details(data)
  'toList' creates a list initializing datas by creating
  clusters with each one

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

[[2]]
     [,1] [,2] [,3]
[1,]    3    4    1

[[3]]
     [,1] [,2] [,3]
[1,]    5    6    1

[[4]]
     [,1] [,2] [,3]
[1,]    7    8    1

[[5]]
     [,1] [,2] [,3]
[1,]    9   10    1
```

NOTA IMPORTANTE: En el apartado *Examples* de la documentación de la función *toList.details* no aparece un ejemplo de ejecución de esta función sino de la función *toList*.

32. *toListDivisive(data)*: Tiene la opción con *.details*. Tiene un argumento: "data" puede ser un vector numérico, una matriz o una estructura de datos numérico. Transforma datos a lista. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1:10)
> toListDivisive(data)
[[1]]
     [,1] [,2]
[1,]    1    2

[[2]]
     [,1] [,2]
[1,]    3    4
```

```
[[3]]
     [,1] [,2]
[1,]    5    6

[[4]]
     [,1] [,2]
[1,]    7    8

[[5]]
     [,1] [,2]
[1,]    9   10

> toListDivisive.details(data)
  'toListDivisive' creates a list initializing datas by creating
  clusters with each one

[[1]]
     [,1] [,2]
[1,]    1    2

[[2]]
     [,1] [,2]
[1,]    3    4

[[3]]
     [,1] [,2]
[1,]    5    6

[[4]]
     [,1] [,2]
[1,]    7    8

[[5]]
     [,1] [,2]
[1,]    9   10
```

33. *usefulClusters(list)*: Tiene un argumento: "list" es una lista de clusters. Un ejemplo de ejecución es la siguiente:

```
> data <- c(1:10)
> list <- toList(data)
> usefulClusters(list)
[[1]]
     [,1] [,2] [,3]
[1,]    1    2    1

[[2]]
     [,1] [,2] [,3]
[1,]    3    4    1

[[3]]
     [,1] [,2] [,3]
[1,]    5    6    1
```

```
[[4]]
     [,1] [,2] [,3]
[1,]    7    8    1

[[5]]
     [,1] [,2] [,3]
[1,]    9   10    1
```

Una vez probadas todas las funciones ofrecidas por el paquete *LearnClust* se observan algunos errores encontrados en ellas o en su documentación (han sido descritas en los apartados de **NOTAS IMPORTANTES**).

A continuación se ha procedido a buscar en los paquetes de R alguna otra forma de realizar la Clusterización Jerárquica Aglomerativa. Finalmente se ha encontrado dentro del paquete *stats*[6] una función sobre la clusterización jerárquica. Esta función se llama *hclust*.

La función *hclust* realiza el análisis jerárquico de clusters sobre un conjunto de disimilitudes y ofrece métodos para analizarlo. Para darle uso se llama a la función de la siguiente forma: *hclust(d, method = "complete", members = NULL)*. Tiene tres argumentos: "d" es una estructura de disimilitud producida por *dist* (cálculo de la matriz de distancias), "method" es el método de aglomeración que se utilizará que puede ser una abreviación de "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" o "centroid" y "members" que puede ser *NULL* o un vector del mismo tamaño de "d". Luego se usa la función *plot* para visualizar correctamente los resultados.

En el siguiente ejemplo de ejecución se usará el grupo de datos de *USArrests*. Este conjunto de datos contiene estadísticas, en arrestados por cada 100.000 habitantes por asalto, asesinato y violación en cada uno de los 50 estados de EE.UU. en 1973. También ofrece el porcentaje de población que vive en zonas urbanas.

La gráfica obtenida es la de la Figura 22 y el código es el siguiente:

```
> hc <- hclust(dist(USArrests), "ave")
> plot(hc)
```

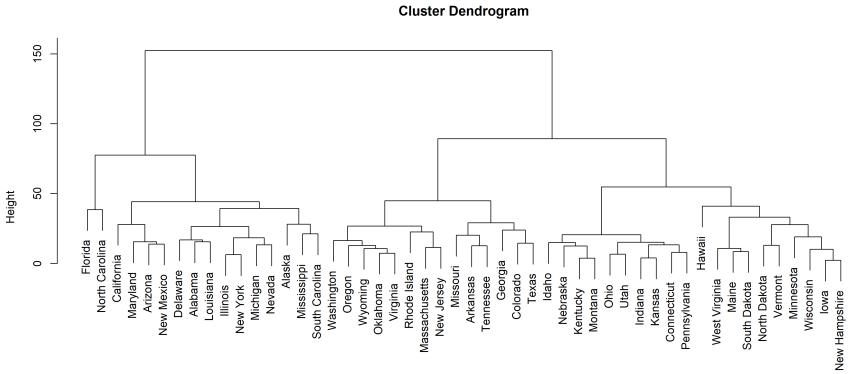


Figure 22: Dendograma de los datos de *USArrests*.

Además de la Clusterización Jerárquica Aglomerativa existe otro tipo de Clusterización Jerárquica que se puede realizar en R, la Clusterización Jerárquica Divisiva. La función que se puede usar para realizarla es la función *diana* del paquete *cluster*[3]. Un ejemplo de ejecución con las gráficas de salida (Figuras 23 y 24) es la siguiente:

```
> data(votes.repub)
> dv <- diana(votes.repub, metric = "manhattan", stand = TRUE)
```

```
> print(dv)
Merge:
      [,1] [,2]
 [1,]   -7  -32
 [2,]  -13  -35
 [3,]  -12  -50
 [4,]    1  -30
 [5,]  -26  -28
 [6,]   -5  -37
 [7,]  -22  -38
 [8,]  -21  -39
 [9,]  -16  -27
[10,]    4    2
[11,]  -25  -48
[12,]  -42  -46
[13,]   -6  -14
[14,]  -34  -41
[15,]   -8  -20
[16,]    5  -31
[17,]   10    7
[18,]  -17  -47
[19,]   -3  -44
[20,]  -33   12
[21,]   15   18
[22,]   17  -29
[23,]   22  -49
[24,]   21   11
[25,]   23  -15
[26,]  -11  -19
[27,]    3    9
[28,]    8  -23
[29,]   19   16
[30,]   27   14
[31,]    6   25
[32,]   -1  -10
[33,]   31   13
[34,]   29  -36
[35,]   -2  -45
[36,]   -9  -43
[37,]   24   20
[38,]   32   -4
[39,]  -24  -40
[40,]   38  -18
[41,]   33   30
[42,]   34   37
[43,]   35   26
[44,]   41   28
[45,]   40   36
[46,]   42   44
[47,]   45   39
[48,]   43   46
[49,]   47   48
Order of objects:
 [1] Alabama        Georgia        Arkansas
```

```
 [4] Louisiana      Florida        Texas
 [7] Mississippi    South Carolina Alaska
[10] Vermont        Hawaii         Maine
[13] Arizona        Utah           Montana
[16] Nevada         New Mexico     Oklahoma
[19] Delaware       Maryland       Kentucky
[22] Washington     Missouri       West Virginia
[25] North Carolina Tennessee      Virginia
[28] California     Oregon         Connecticut
[31] New York       New Jersey     Illinois
[34] Ohio           Michigan       Pennsylvania
[37] New Hampshire  Wisconsin      Iowa
[40] Colorado       Indiana        Idaho
[43] Wyoming        Kansas         Nebraska
[46] North Dakota   South Dakota   Massachusetts
[49] Rhode Island   Minnesota
Height:
 [1]  27.363453  33.969252  39.658259  48.534276  31.899654
 [6]  72.598496  35.691518 167.580197  31.582223  43.846009
[11]  24.487963  85.552482  18.393392  25.676314  11.493967
[16]  17.455521  28.625502  42.544800  16.485096  20.044499
[21]  17.875161  21.983729  14.218077  33.610713  18.397326
[26]  14.757619  56.556754  11.701321  27.058874   8.382005
[31]  11.368197  13.252375   9.230040  17.834836  12.708189
[36]  20.667139  21.039972  23.665856  28.605405  15.317027
[41]  40.339045  10.462936  24.835249  12.804188  26.362915
[46]  16.251922  47.257725  12.791603  24.872061
Divisive coefficient:
[1] 0.8869182

Available components:
[1] "order"     "height"    "dc"        "merge"     "diss"
[6] "call"      "order.lab" "data"
> plot(dv)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

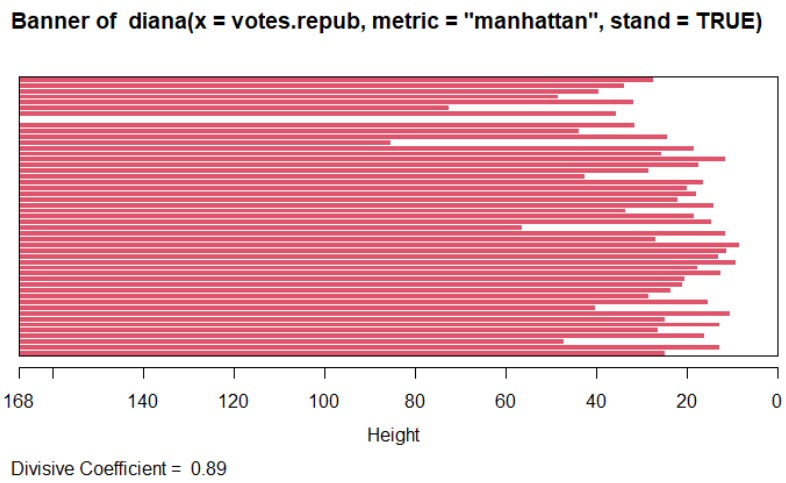


Figure 23: Primera gráfica de la ejecución de *diana*.

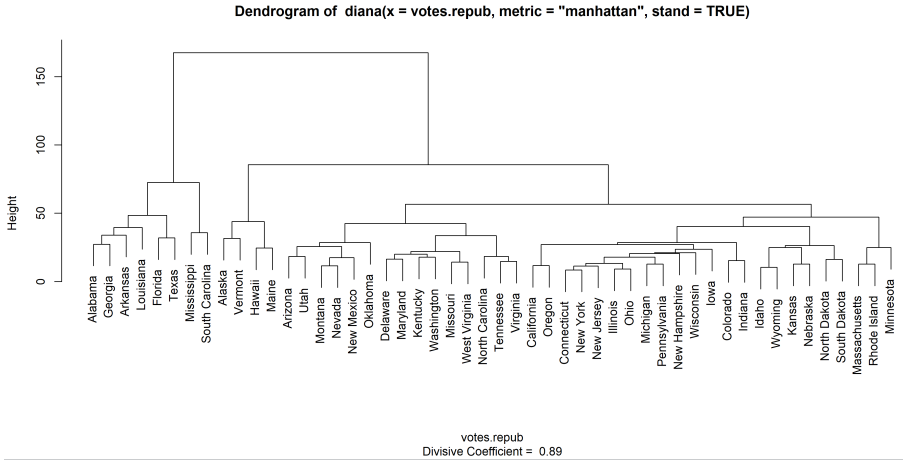


Figure 24: Segunda gráfica de la ejecución de *diana*.

Lo siguiente que se va a ver, después de analizar el paquete y proponer otras alternativas existentes, se va a realizar algún cambio a *LearnClust*. En este caso se trata de mejorar el dendrograma, que como se vio en la primera parte no se muestra de forma correcta.

Antes que nada, se ha detectado cuáles serían los problemas a la hora de crear el dendrograma. En el fichero *agglomerativeHC.R* entre las líneas 74-79 es dónde se encuentra el problema.

```
dendrogram <- list()
dendrogram$merge <- as.matrix(clustersDendrogram)
dendrogram$height <- c(1:(lengthList-1))
dendrogram$order <- c(1:lengthList)
dendrogram$labels <- c(1:lengthList)
class(dendrogram) <- "hclust"
```

La variable *height* está asignada como un vector de 1 a $lengthList-1$ siendo *lengthList* el tamaño de la lista de los datos que se le pasa a la función *agglomerativeHC()*. Al igual que está mal la asignación de la variable *order* que también es un vector, pero de 1 hasta *lengthList*. El ajuste de estas dos variables son las que solucionan el que se muestre de forma correcta el dendrograma, por ello se explica una posible solución.

Para obtener el *height* correcto del dendrograma, se ha añadido una variable nueva antes del *while (!oneCluster)* con el nombre de *heightDendrogram* que se trata de un vector vacío. Dentro del bucle *while*, dicho vector va a ir almacenando las distancias mínimas que son calculadas con la función *minDistance()*.

```
while (!oneCluster){
  matrixDistance <- mdAgglomerative(list,distance,approach)
  minDistance <- minDistance(matrixDistance)
  heightDendrogram <- c(heightDendrogram, minDistance)
  ...
```

Esta nueva variable será la que se le asigne después de terminar el bucle while a *dendrogram$height*.

Para tener el *order* correcto se hace a partir del último cluster que se aporta como solución. Se recorre para ir obteniendo cada uno de los vectores y usar la función *match()* para conocer la posición en la que se encuentra dicho vector en la variable *data*. Esta posición se va almacenando en la variable *orderDendrogram* que es un vector.

```
lastCluster <- res[length(res)][[1]]
orderDendrogram <- c()
for (index in 1:nrow(lastCluster))
{
  vector <- as.vector(t(lastCluster[index,]))
  orderDendrogram <- c(orderDendrogram, match(vector, data)[1])
}
```

Una vez con el orden correcto, se le asigna la variable *orderDendrogram* a *dendrogram$order*. De tal forma, que el código visto al principio donde se detectaba los problemas, ha quedado ahora así:

```
dendrogram <- list()
dendrogram$merge <- as.matrix(clustersDendrogram)
dendrogram$height <- heightDendrogram
dendrogram$order <- orderDendrogram
dendrogram$labels <- c(1:lengthList)
class(dendrogram) <- "hclust"
```

Si se hace ahora un *plot()* con los datos de la primera parte, el resultado es el siguiente:

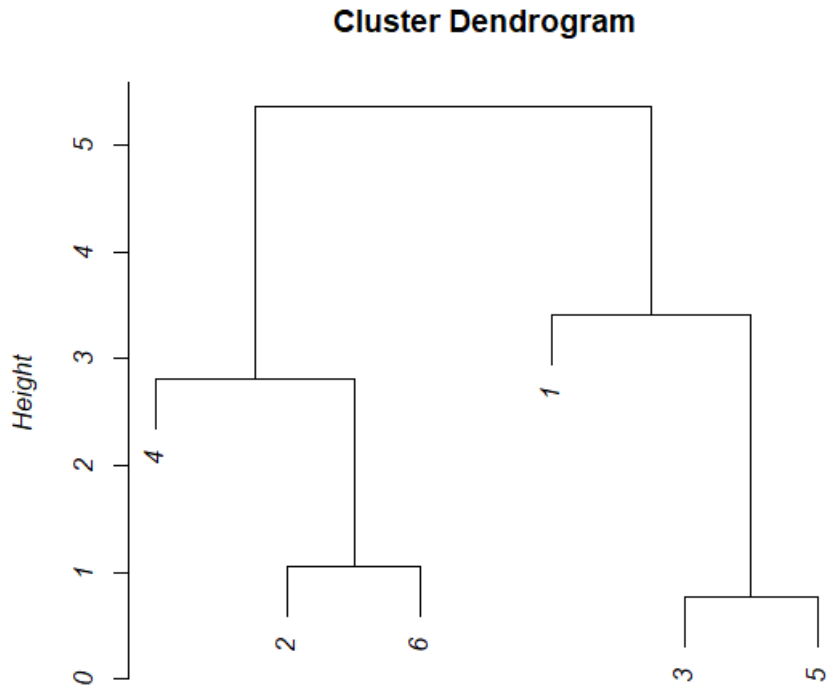


Figure 25: Dendrograma corregido en el paquete LearnClust.

Si se compara con el resultado que ofrece el paquete *stats*, se puede comprobar que son el mismo dendrograma:

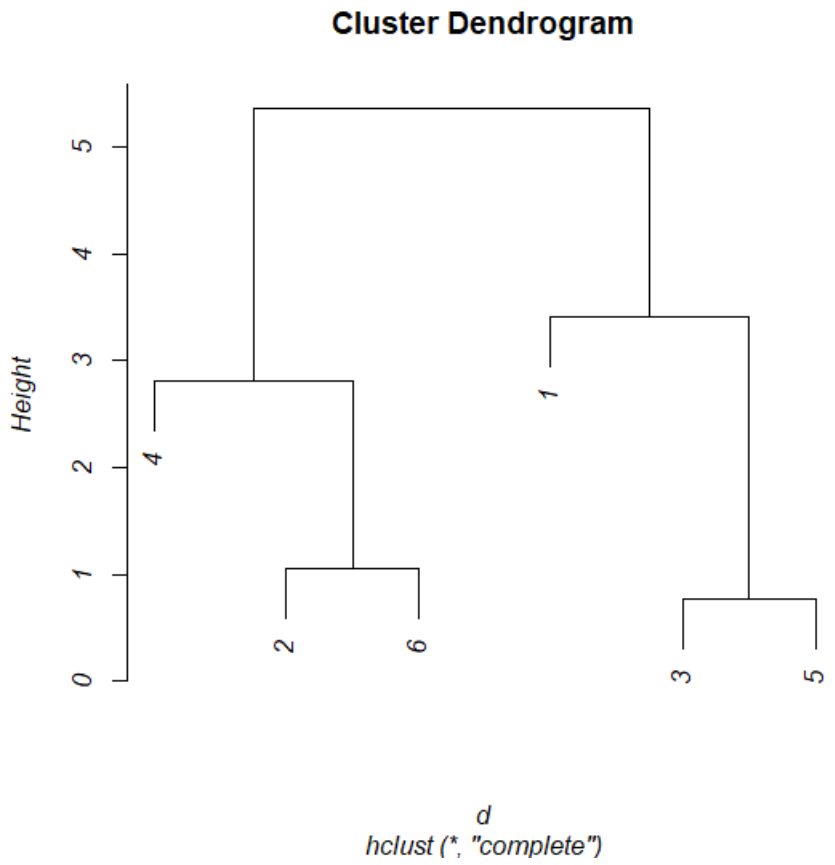


Figure 26: Dendrograma del paquete stats.

3.3 Análisis de clasificación supervisada

3.3.1 Utilizando árboles de decisión

El primer análisis de clasificación supervisada es el del algoritmo de los árboles de decisión. Para ello se han creado funciones que permiten realizar el algoritmo de la manera más parecida a la del método *rpart()*.

La muestra que se va a utilizar para la clasificación es la de los vehículos cuyas variables son el tipo de carné, el número de ruedas, el número de pasajeros y el tipo de vehículo. El clasificador es el tipo de vehículo.

La muestra en formato CSV se carga en R con el método *read.csv()*.

```
> (vehiculos = read.csv("vehiculos.csv"))
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
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
```

Para empezar se crean funciones de utilidad que se usarán en el desarrollo del algoritmo. Estas son los valores únicos de las columnas, las clases de la tabla, la comprobación de si un elemento es un número y la función para crear una pregunta con la columna y un valor.

```
valoresUnicos <- function(datos, columna)
{
        unique(datos[columna])
}

contarClases <- function(datos)
{
        table(datos[, ncol(datos)])
}

esNumero <- function(valor)
{
        is.numeric(valor)
}

crearPregunta <- function(columna, valor)
{
        list(columna = columna, valor = valor)
}
```

A continuación se crea la función *coincide()* que lo que hace es comprobar si una pregunta realizada coincide con el valor de una fila, es decir, si el elemento que se quiere buscar se encuentra en la fila que se está observando. Esta devolverá TRUE si coincide y el valor de la columna a la que se hace referencia en la pregunta.

```
coincide <- function(pregunta, fila)
{
        valor = fila[pregunta$columna]

        if (esNumero(valor))
        {
                valor >= pregunta$valor
        }
        else
        {
                valor == pregunta$valor
        }
}
```

Para entenderlo mejor se realizan los siguientes ejemplos.

En el primero la pregunta que se realiza es si en la primera columna, que en la muestra hace referencia al tipo de carné, existe el valor "A". Este valor tendrá que coincidir con el de la primera fila, lo que no lo hace como se puede observar.

Otros ejemplos son los siguientes:

```
> coincide(crearPregunta(1, "A"), vehiculos[1,])
  TipoCarnet
1      FALSE
```

```
> coincide(crearPregunta(1, "B"), vehiculos[1,])
  TipoCarnet
1       TRUE
> coincide(crearPregunta(2, 3), vehiculos[1,])
  NumeroRuedas
1        FALSE
> coincide(crearPregunta(2, 4), vehiculos[1,])
  NumeroRuedas
1         TRUE
```

La función con la que se empieza el desarrollo del algoritmo que se crea ahora es la de *dividir()*. Ésta divide la tabla de la muestra en dos nodos, el de la izquierda y la derecha, según la pregunta que se haga para ir clasificando los datos que permita obtener la mayor ganancia posible.

```
dividir <- function(datos, pregunta)
{
    coincidencias <- data.frame()
    eliminar <- c()

        for (i in 1:nrow(datos))
        {
                fila <- datos[i, ]
                if (coincide(pregunta, fila))
                {
                        coincidencias <- rbind(coincidencias, fila)
            eliminar <- c(eliminar, i)
                }
        }

    izquierda <- datos[-eliminar, ]

    claseUnica <- valoresUnicos(coincidencias, ncol(coincidencias))
    if (nrow(claseUnica) == 1)
    {
        derecha <- coincidencias
    }
    else
    {
        derecha <- data.frame()
    }

    list(izquierda = izquierda, derecha = derecha)
}
```

En la muestra de vehículos se divide la tabla mediante la pregunta de los valores "N" en el tipo de carné por lo que el nodo padre que se utiliza es el de "TipoCarnet". Lo que hace que se separen las bicicletas del resto de vehículos.

```
> (division <- dividir(vehiculos, crearPregunta(1, "N")))
$izquierda
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
1          B            4               5        Coche
2          A            2               2         Moto
4          B            6               4       Camion
5          B            4               6        Coche
```

```
6          B            4               4        Coche
8          B            2               1         Moto
9          B            6               2       Camion

$derecha
   TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
3           N            2               1    Bicicleta
7           N            2               2    Bicicleta
10          N            2               1    Bicicleta
```

A continuación se crea la función de impureza del *gini* que teniendo como referencia la fórmula teórica quedaría de tal forma:

```
gini <- function(datos)
{
        contadores <- contarClases(datos)
        impureza <- 1
        n <- nrow(datos)

        for (contador in contadores)
        {
                probabilidad <- contador / n
                impureza <- impureza - probabilidad ^ 2
        }

        impureza
}
```

La impureza del nodo de la izquierda obtenido anteriormente será la siguiente:

```
> gini(division$izquierda)
[1] 0.6530612
```

Tras tener calculado las impurezas del nodo padre y los nodos hijos (izquierda y derecha), se crea la función que permite obtener la ganancia de información del árbol elegido. Esta seguirá la fórmula teórica.

```
gananciaInformacion <- function(izquierda, derecha, impurezaPadre)
{
        p = nrow(izquierda) / (nrow(izquierda) + nrow(derecha))
        impurezaPadre - p * gini(izquierda) - (1 - p) * gini(derecha)
}
```

La ganancia de información de "TipoCarnet" en la primera iteración es la siguiente:

```
> gananciaInformacion(division$izquierda, division$derecha, gini(vehiculos))
[1] 0.2828571
```

Después de tener las funciones principales para obtener los nodos hijos según el nodo padre, el cálculo de impurezas con *gini* y la ganancia de información; se crea la función que permite saber cuál es la mejor división entre los posibles nodos de la muestra, es decir, la que resulta en una mayor ganancia. Para ello se comprueba en cada posible nodo (columna) cual es la ganancia de este separando los nodos hijos según las preguntas que se hagan. Se realizan todas las preguntas posibles, es decir, se comprueba si todos los valores posibles están en cada una de las filas para así poder realizar todas las divisiones posibles. Para cada una de las divisiones de cada columna (nodo padre) se obtiene el cálculo de ganancia y se van comparando para así obtener la mayor ganancia posible desde una columna específica (nodo padre) y la división que se realiza.

```
buscarMejorDivision <- function(datos)
{
        mejorGanancia <- 0
        mejorDivision <- NULL
        mejorColumna <- 0
        impurezaPadre <- gini(datos)
        columnas <- ncol(datos) - 1

        for (columna in 1:columnas)
        {
            valores <- valoresUnicos(datos, columna)

            for (i in 1:nrow(valores))
        {
            valor <- valores[i, 1]

            pregunta <- crearPregunta(columna, valor)

            division <- dividir(datos, pregunta)

            if (nrow(division$izquierda) == 0 || nrow(division$derecha) == 0)
            {
                next
            }

            ganancia <- gananciaInformacion(division$izquierda, division$derecha, impurezaPadre)

            if (ganancia > mejorGanancia)
            {
                mejorGanancia <- ganancia
                mejorDivision <- division
                mejorColumna <- columna
            }
        }
        }

        list(ganancia = mejorGanancia, columna = names(datos[mejorColumna]), division = mejorDivisio
}
```

En la primera instancia, es decir, con todas las filas de la muestra la mayor ganancia de información se obtiene desde el nodo padre "TipoCarnet" que consigue separar en sus nodos hijos el "TipoVehiculo" bicicleta de los demás vehículos. Tras esto se puede empezar a construir el árbol de decisiones.

```
> buscarMejorDivision(vehiculos)
$ganancia
[1] 0.2828571

$columna
[1] "TipoCarnet"

$division
$division$izquierda
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
1          B            4               5        Coche
```

```
2          A            2               2         Moto
4          B            6               4        Camion
5          B            4               6         Coche
6          B            4               4         Coche
8          B            2               1         Moto
9          B            6               2        Camion

$division$derecha
   TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
3           N            2               1    Bicicleta
7           N            2               2    Bicicleta
10          N            2               1    Bicicleta
```

Para obtener el árbol de decisiones completo se crean funciones que permiten simular la estructura de datos de los árboles.

Las primeras sirven para crear las hojas y los nodos del árbol. Después se crea la función principal *construirArbol()* que guardará las mejores divisiones para ir construyendo el árbol recursivamente. Si la ganancia de una división es 0 entonces se crea un nodo hoja para dar fin al árbol, si no lo fuera se crean los nodos a partir del resultado de la mejor división, la rama izquierda y la rama derecha.

```
crearHoja <- function(datos)
{
        list(n = contarClases(datos))
}

crearNodo <- function(resultado, ramaIzquierda, ramaDerecha)
{
        list(resultado = resultado, ramaIzquierda = ramaIzquierda, ramaDerecha = ramaDerecha)
}

construirArbol <- function(datos)
{
        resultado <- buscarMejorDivision(datos)

        if (resultado$ganancia == 0)
        {
                return(crearHoja(datos))
        }

        ramaIzquierda <- construirArbol(resultado$division$izquierda)
        ramaDerecha <- construirArbol(resultado$division$derecha)

        return(crearNodo(resultado, ramaIzquierda, ramaDerecha))
}
```

La construcción del árbol de decisiones de la muestra de vehículos quedará de la siguiente forma:

```
> construirArbol(vehiculos)
$resultado
$resultado$ganancia
[1] 0.2828571

$resultado$columna
[1] "TipoCarnet"
```

```
$resultado$division
$resultado$division$izquierda
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
1          B            4               5        Coche
2          A            2               2         Moto
4          B            6               4       Camion
5          B            4               6        Coche
6          B            4               4        Coche
8          B            2               1         Moto
9          B            6               2       Camion

$resultado$division$derecha
   TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
3           N            2               1    Bicicleta
7           N            2               2    Bicicleta
10          N            2               1    Bicicleta



$ramaIzquierda
$ramaIzquierda$resultado
$ramaIzquierda$resultado$ganancia
[1] 0.3673469

$ramaIzquierda$resultado$columna
[1] "NumeroRuedas"

$ramaIzquierda$resultado$division
$ramaIzquierda$resultado$division$izquierda
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
2          A            2               2         Moto
4          B            6               4       Camion
8          B            2               1         Moto
9          B            6               2       Camion

$ramaIzquierda$resultado$division$derecha
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
1          B            4               5        Coche
5          B            4               6        Coche
6          B            4               4        Coche



$ramaIzquierda$ramaIzquierda
$ramaIzquierda$ramaIzquierda$resultado
$ramaIzquierda$ramaIzquierda$resultado$ganancia
[1] 0.5

$ramaIzquierda$ramaIzquierda$resultado$columna
[1] "NumeroRuedas"

$ramaIzquierda$ramaIzquierda$resultado$division
$ramaIzquierda$ramaIzquierda$resultado$division$izquierda
```

```
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
4          B            6               4       Camion
9          B            6               2       Camion

$ramaIzquierda$ramaIzquierda$resultado$division$derecha
  TipoCarnet NumeroRuedas NumeroPasajeros TipoVehiculo
2          A            2               2         Moto
8          B            2               1         Moto



$ramaIzquierda$ramaIzquierda$ramaIzquierda
$ramaIzquierda$ramaIzquierda$ramaIzquierda$n

Camion
     2


$ramaIzquierda$ramaIzquierda$ramaDerecha
$ramaIzquierda$ramaIzquierda$ramaDerecha$n

Moto
   2



$ramaIzquierda$ramaDerecha
$ramaIzquierda$ramaDerecha$n

Coche
    3



$ramaDerecha
$ramaDerecha$n

Bicicleta
        3
```

Para entender mejor los resultados de esta función se crean otras funciones para visualizar por pantalla el árbol de forma más clara.

La primera pinta el nodo raíz de forma que muestre la columna de éste y la ganancia de información.

Después se pintan las ramas derecha e izquierda de manera que se muestren las variables que haya en cada una de ellas junto con el número de veces que aparecen. Esto se consigue con *contarClases(rama)* usada en *pintarContenidoRama()* que a su vez se utiliza en *pintarRama()*.

Por último se crea *pintarArbol()* que utilzará las funciones anteriores para obtener el árbol de decisiones completo con los datos de las ganancias y el número de veces que aparece una clase en cada rama.

```
pintarRaiz <- function(raiz)
{
    cat(paste(raiz$resultado$columna, "ganancia:", raiz$resultado$ganancia, "\n"))
```

```
}



pintarRama <- function(raiz, tab="\t")
{
    if (length(raiz$n) > 0)
    {
        cat("*\n")
        return()
    }
    cat(paste("\n", tab))
    pintarRaiz(raiz)

    cat(tab)
    pintarContenidoRama(raiz$resultado$division$derecha)
    pintarRama(raiz$ramaDerecha, paste(tab, "\t"))

    cat(tab)
    pintarContenidoRama(raiz$resultado$division$izquierda)
    pintarRama(raiz$ramaIzquierda, paste(tab, "\t"))
}

pintarContenidoRama <- function(rama)
{
    clase <- contarClases(rama)

    for (i in 1:length(clase))
    {
        cat(paste(names(clase[i]), ":", clase[[i]], ""))
    }
}

pintarArbol <- function(arbol)
{
    pintarRaiz(arbol)

    pintarContenidoRama(arbol$resultado$division$derecha)
    pintarRama(arbol$ramaDerecha)

    pintarContenidoRama(arbol$resultado$division$izquierda)
    pintarRama(arbol$ramaIzquierda)

    cat("\n")
}
```

Por último se reune todo el desarrollo del algoritmo en la función *clasificacion()*, en la que se podrá visualizar la clasificación supervisada con el árbol de decisiones dados unos datos de muestra con clases específicas y un clasificador.

```
clasificacion <- function(datos)
{
    arbol <- construirArbol(datos)
    pintarArbol(arbol)
}
```

Con los datos de los vehículos el árbol resultante será el siguiente:

```
> clasificacion(vehiculos)
TipoCarnet ganancia: 0.282857142857143
Bicicleta : 3 *
Camion : 2 Coche : 3 Moto : 2
        NumeroRuedas ganancia: 0.36734693877551
        Coche : 3 *
        Camion : 2 Moto : 2
                NumeroRuedas ganancia: 0.5
                Moto : 2 *
                Camion : 2 *
```

Comparando el algoritmo desarrollado con la función *rpart()* usada en la primera parte se puede ver que los nodos coinciden así como las divisiones realizadas y el conteo de las clases en cada rama..

```
> rpart(TipoVehiculo~.,data=vehiculos,method="class",minsplit=1)
n= 10

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

 1) root 10 7 Bicicleta (0.3000000 0.2000000 0.3000000 0.2000000)
   2) TipoCarnet=N 3 0 Bicicleta (1.0000000 0.0000000 0.0000000 0.0000000) *
   3) TipoCarnet=A,B 7 4 Coche (0.0000000 0.2857143 0.4285714 0.2857143)
     6) NumeroRuedas>=3 5 2 Coche (0.0000000 0.4000000 0.6000000 0.0000000)
      12) NumeroRuedas>=5 2 0 Camion (0.0000000 1.0000000 0.0000000 0.0000000) *
      13) NumeroRuedas< 5 3 0 Coche (0.0000000 0.0000000 1.0000000 0.0000000) *
     7) NumeroRuedas< 3 2 0 Moto (0.0000000 0.0000000 0.0000000 1.0000000) *
```

3.3.2 Utilizando regresión

En este apartado se hará el análisis de clasificación supervisada utilizando regresión. Se ha creado 4 archivos *.csv*, cada uno tiene un subconjunto de datos al que se le determinará su recta de regresión y su parámetro de ajuste.

La función *read.csv.conjuntos()* recibe como parámetro la ruta donde se encuentra los *.csv* mencionados anteriormente y devuelve una lista con todo su contenido.

```
read.csv.conjuntos <- function(path)
{
        filenames <- list.files(path, "*.csv")

        conjuntos <- list()

        for (i in 1:length(filenames))
        {
                filepath = file.path(path, filenames[i])
                conjuntos[[i]] <- read.csv(filepath, header = TRUE)
        }
        conjuntos
}
```

La variable *conjuntos* almacena el contenido de todos los *.csv* y se muestra los datos del primer archivo.

```
> conjuntos <- read.csv.conjuntos("conjuntos/")
> conjuntos[[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
```

Las funciones *bRecta()*, *aRecta()* e *yCalculada()* han sido programadas y explicada en el apartado de *Regresión* de la *Segunda Parte* es por ese motivo que no se añade el código. Se recuerda que *muestraR* y *muestraD* contienen los datos del radio y densidad respectivamente del archivo *planeta.txt*. La función *mediaAritmetica()* se ha reutilizado de la *PL1*.

Teniendo la recta de regresión se coloca en el valor de las x los datos del radio de los planetas para calcular la media de las ycalculada.

```
> (b <- bRecta(muestra$R, muestra$D))
[1] 0.1393669
> (a <- aRecta(muestra$R, muestra$D, b))
[1] 4.362396
> (yc <- yCalculada(muestra$R, a, b))
[1] 4.696877 5.212535 5.254345 4.836244
> (yMedia <- mediaAritmetica(yc))
[1] 5
```

La función *SSR()* y *SSy()* reciben como primer parámetro *yc* (ycalculada) e *y* respectivamente, y el segundo parámetro yMedia (media de las y) que usando la función *sumatorio()* se rectan estos valores y se elevan al cuadrado.

```
SSR <- function(yc, yMedia)
{
        sumatorio(yc, -yMedia, 2)
}

SSy <- function(y, yMedia)
{
        sumatorio(y, -yMedia, 2)
}
```

Resultado del *SSR()* usando los datos del radio de los planetas.

```
> SSR(yc, yMedia)
[1] 0.2285617
```

Resultado del *SSy()* usando los datos de la densidad de los planetas.

```
> SSy(muestra$D, yMedia)
[1] 1.66
```

La función *Rcuadrado()* contiene en su interior todas las funciones antes vistas en este apartado.

```
Rcuadrado <- function(x, y)
{
        b <- bRecta(x, y)
```

```
        a <- aRecta(x, y, b)
        yc <- yCalculada(x, a, b)
        yMedia <- mediaAritmetica(yc)

        SSR(yc, yMedia) / SSy(y, yMedia)
}
```

Resultado de la función *Rcuadrado()* usando los datos del radio y la densidad.

```
> Rcuadrado(muestra$R, muestra$D)
[1] 0.1376878
```

Se ha calculado la recta de regresión de cada uno de los subconjuntos.

```
> (b <- bRecta(conjuntos[[1]]$x, conjuntos[[1]]$y))
[1] 0.5000909
> (a <- aRecta(conjuntos[[1]]$x, conjuntos[[1]]$y, b))
[1] 3.000091

> (b <- bRecta(conjuntos[[2]]$x, conjuntos[[2]]$y))
[1] 0.5
> (a <- aRecta(conjuntos[[2]]$x, conjuntos[[2]]$y, b))
[1] 3.000909

> (b <- bRecta(conjuntos[[3]]$x, conjuntos[[3]]$y))
[1] 0.4997273
> (a <- aRecta(conjuntos[[3]]$x, conjuntos[[3]]$y, b))
[1] 3.002455

> (b <- bRecta(conjuntos[[4]]$x, conjuntos[[4]]$y))
[1] 0.4999091
> (a <- aRecta(conjuntos[[4]]$x, conjuntos[[4]]$y, b))
[1] 3.001727
```

Se ha calculado *Rcuadrado()* con las variables (x e y) de cada uno de los conjuntos de datos de los archivos.

```
> Rcuadrado(conjuntos[[1]]$x, conjuntos[[1]]$y)
[1] 0.6665425

> Rcuadrado(conjuntos[[2]]$x, conjuntos[[2]]$y)
[1] 0.666242

> Rcuadrado(conjuntos[[3]]$x, conjuntos[[3]]$y)
[1] 0.666324

> Rcuadrado(conjuntos[[4]]$x, conjuntos[[4]]$y)
[1] 0.6667073
```

Si los resultados anteriores se comparan a como se calculaban en la *Primera parte* usando la función *lm()* junto al *summary()* obteniendo los valores de *$r.squared* que son los que interesan, se puede ver que son los mismos resultados.

```
> summary(lm(x~y, data=conjuntos[[1]]))$r.squared
[1] 0.6665425

> summary(lm(x~y, data=conjuntos[[2]]))$r.squared
[1] 0.666242
```

```
> summary(lm(x~y, data=conjuntos[[3]]))$r.squared
[1] 0.666324

> summary(lm(x~y, data=conjuntos[[4]]))$r.squared
[1] 0.6667073
```

4 Conclusiones

En esta práctica se han visto los algoritmos más conocidos para la clasificación no supervisada y supervisada, se ha aprendido como funcionan estos algoritmos y su aplicación en diferentes muestras de datos. Se han usado las herramientas que proporciona R, métodos y paquetes, y se han visto y entendido los datos que proporcionan estos. También se han sabido crear funciones directamente en R que simulen el comportamiento de los métodos que ofrece R por defecto que permiten seguir los algoritmos en cada tipo de clasificación.

References

[1] Roberto Alcantara, Juan Jose Cuadrado, and Universidad de Alcala de Henares. *LearnClust: Learning Hierarchical Clustering Algorithms*. R package version 1.1. 2020. URL: https://CRAN.R-project.org/package=LearnClust.

[2] Douglas Bates and Martin Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.3-4. 2021. URL: https://CRAN.R-project.org/package=Matrix.

[3] Martin Maechler et al. *cluster: Cluster Analysis Basics and Extensions*. R package version 2.1.2 — For new features, see the 'Changelog' file (in the package source). 2021. URL: https://CRAN.R-project.org/package=cluster.

[4] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: https://www.R-project.org/.

[5] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: https://www.R-project.org/.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: https://www.R-project.org/.

[7] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: https://www.R-project.org/.

[8] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-15. 2019. URL: https://CRAN.R-project.org/package=rpart.