Fundamentos de la Ciencia de Datos

Práctica 2

Grado en Ingeniería Informática
Universidad de Alcalá

Grupo 9

Pablo García García
Abel López Martínez
Álvaro Jesús Martínez Parra
Raúl Moratilla Núñez

20 de diciembre de 2023

Índice general

Introducción

Parte 1

Ejercicios guiados

En esta primera parte de la práctica, se repetirán los ejercicios explicados y realizados por el profesor en las clases de laboratorio, utilizando los mismos procedimientos vistos y plasmándolos en este documento.

1.1. Clasificación No Supervisada con K-Means

Ejercicio 1.1.1. *El primer conjunto de datos, que se empleará para realizar el análisis de clasificación no supervisada con KMeans, estará formado por las siguientes 8 calificaciones de estudiantes: 1. {4, 4}; 2. {3, 5}; 3. {1, 2}; 4. {5, 5}; 5.{0, 1}; 6. {2, 2}; 7. {4, 5}; 8. {2, 1}, donde las características de las calificaciones son: {Teoría, Laboratorio}.*

En este ejercicio se va a detallar cómo realizar una clasificación no supervisada en un conjunto de datos en R; concretamente utilizando el algoritmo K-Means. El objetivo será agrupar subconjuntos de los datos en clusters o grupos, creando de esta forma una clasificación del conjunto total. Estos clusters se determinarán en base a la muestra, obteniéndose durante el mismo proceso de clasificación.

El algoritmo K-Means ofrece una técnica de clusterización en base a una muestra de datos y una cantidad n de clusters. Para empezar a realizar el algoritmo se necesita la ubicación de los centroides de cada cluster. Un centroide es el punto medio del grupo de sucesos que componen el cluster, por lo que habrá tantos centroides como n clusters haya. La cantidad de clusters así como sus centroides iniciales serán elegidos arbitrariamente por el usuario; estos se irán reubicando a medida que se itere en el algoritmo.

Para empezar, en R se necesita introducir la muestra o conjunto de datos con el que se va a trabajar. Estos datos serán introducidos en una matriz utilizando la función `matrix`.

```
m<-matrix(c(4,4, 3,5, 1,2, 5,5, 0,1, 2,2, 4,5, 2,1),2,8)
(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
```

Para poder trabajar debidamente se debe trasponer la matriz, de tal forma que cada columna represente una componente. El primer punto conformará la primera fila, el segundo punto la segunda fila y así sucesivamente con todos los datos.

Además del conjunto de datos, el algoritmo K-Means necesita unos centroides iniciales. Estos se introducirán de la misma forma que la muestra, teniendo en la primera fila el centroide del primer cluster, en la siguiente fila el centroide del segundo... Al tratarse de pocos puntos se eligen arbitrariamente dos centroides, por lo que la clasificación final será realizada con dos clusters. Se introducen así los centroides:

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

##      [,1] [,2]
## [1,]    0    1
## [2,]    2    2
```

Con la muestra y los centroides introducidos en el entorno de trabajo, ya se puede realizar el algoritmo. Para ello se utilizará la función `kmeans`, función del paquete cargado por defecto `stats`. La función recibe tres parámetros: la muestra de los datos (`m`), los centroides iniciales de los clusters (`c`) y el número de iteraciones que se desean. En este caso se eligen cuatro iteraciones, las mismas que se necesitaron en clase de teoría. En teoría, el número de iteraciones no se sabe a priori, por lo que habría que ir probando. Sin embargo, como ya se sabe el número que se necesita, se pone directamente 4.

```
(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"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Entre los parámetros que aparecen en la salida se encuentran `Cluster means` y `Clustering vector`. El primero de ellos proporciona la ubicación de los dos centroides de los clusters que conforman la clasificación final. Como se puede apreciar, salen los dos centroides que salieron en teoría. El segundo parámetro indica a qué cluster pertenece cada suceso de la muestra introducida. Así el punto 1 pertenece al cluster 2, el punto 2 pertenece al cluster 2, el punto 3 al cluster 1... Este último se puede utilizar como entrada para realizar otras tareas. Puede ser muy útil para analizar cada cluster por separado, ver sus caracteristicas comunes e intentar deducir por qué esos datos están relacionados.

El siguiente comando permite añadir una columna a la izquierda de la muestra de datos (`m`) que se había introducido al inicio, indicando a qué cluster pertenece cada suceso.

```
(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
```

Esto se realiza para poder dividir la muestra según la clasificación que se ha conseguido. Llamando a la función subset se puede extraer la porción de la muestra que pertenece al cluster 1 y la porción restante que pertenece al cluster 2.

```
(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

(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
```

Una vez se tienen los datos separados se puede eliminar la columna que indica el cluster al que pertenece cada dato, ya que se sobreentiende que todos pertenecen al mismo cluster porque ya han pasado por un proceso de separación en función de la clasificación. Por ejemplo, para los datos pertenecientes al primer cluster, se haría de la siguiente forma:

```
(mc1 = mc1[,-1])

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

Con todo esto se consigue, partiendo de una muestra de datos, una clasificación no supervisada utilizando la técnica de clusterización basada en el algoritmo K-Means. A partir de ella se podrán intentar deducir ciertas conclusiones.

1.2. Clasificación No Supervisada con CJA

Ejercicio 1.2.1. *El segundo conjunto de datos, que se empleará para realizar el análisis de clasificación no supervisada con Clusterización Jerárquica Aglomerativa, estará formado por 6 calificaciones de estudiantes: 1. {0.89, 2.94}; 2.{4.36, 5.21}; 3. {3.75, 1.12}; 4. {6.25, 3.14}; 5. {4.1, 1.8}; 6. {3.9, 4.27}.*

La Clusterización Jerárquica Aglomerativa es un método de análisis de datos que comienza tratando cada punto como un cluster individual y luego, iterativamente, combina los clusters más cercanos hasta formar un único cluster. Este método es útil para identificar grupos naturales en los datos. En clase se implementó mediante el paquete `LearnClust` del CRAN.

Inicialmente, se crea una matriz con la muestra del problema, y se transpone la matriz para que tenga el formato deseado.

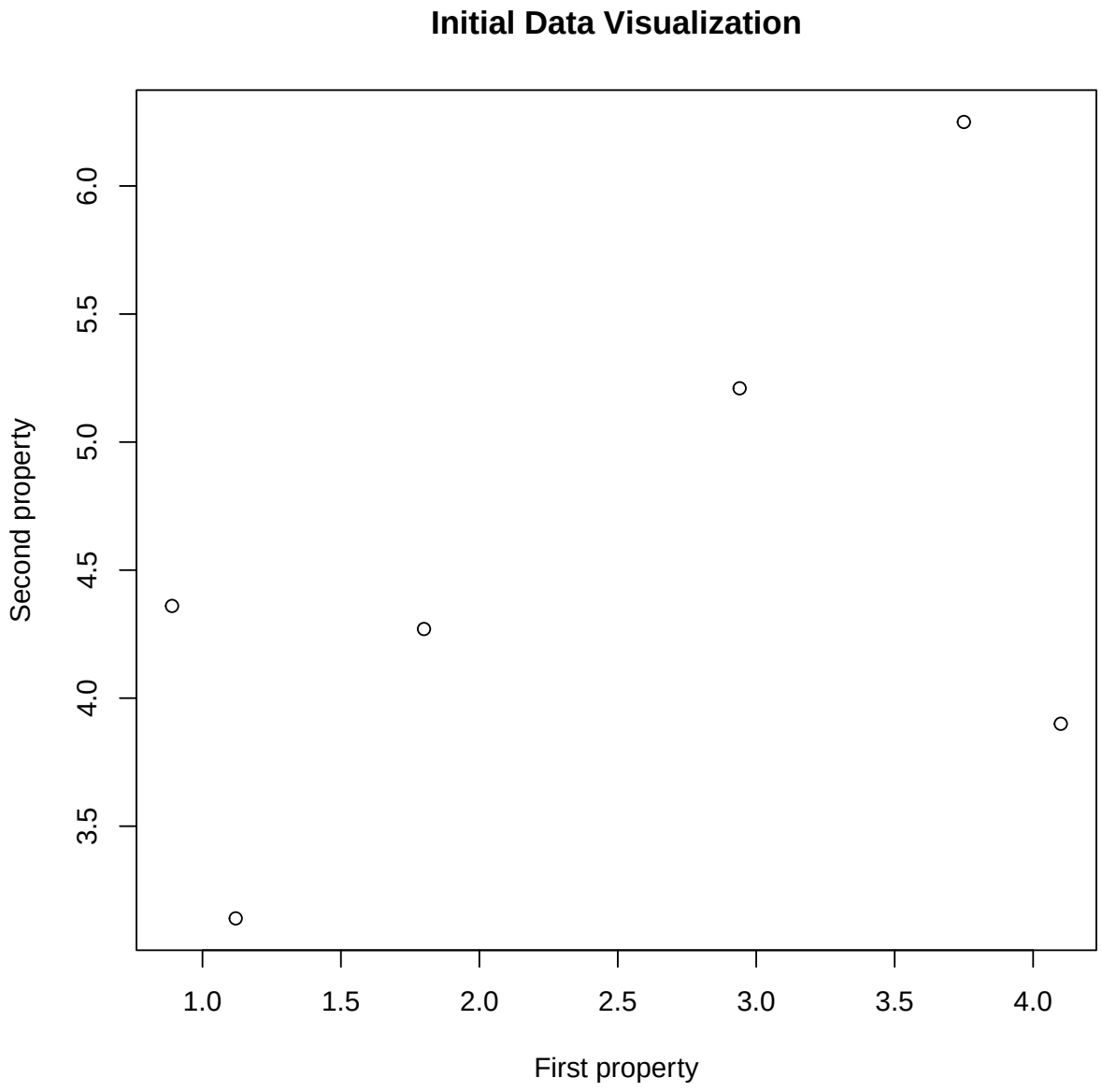
```
m <- matrix(
        c(0.89,2.94, 4.36,5.21, 3.75,1.12, 6.25,3.14, 4.1,1.8, 3.9,4.27),2,6)
(m <- t(m))

##      [,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
```

Para la ejecución del algoritmo utilizamos la función `agglomerativeHC`. Esta función recibe como parámetros la matriz de datos, la métrica de distancia y el criterio de enlace.

```
agglomerativeHC(m, 'EUC', 'MIN')
```



```
## $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 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
```

En este código, 'EUC' representa la métrica de distancia euclidiana y 'MIN' el criterio para la agrupación de clusters. En la salida se pueden observar cuatro apartados.

El primero es una figura los datos de la matriz que hemos introducido impresos en una gráfica.

El segundo es el número de puntos que se han introducido, mediante el atributo `$dendrogram`, en este caso 6.

El tercer apartado es una lista que muestra las coordenadas de todos los clusters al finalizar la ejecución, los 6 primeros son los puntos introducidos y los 5 siguientes son los formados por cada una de las iteraciones uniendo dos clusters anteriores hasta tener todos unidos en el mismo cluster, momento en el que finaliza el algoritmo. Se puede ver en el atributo `$clusters`.

El cuarto apartado es el proceso que se realiza en cada iteración, se pueden ver en el apartado `$groupedClusters`.

1. Se unen los clusters 3 y 5, para formar el cluster 7.
2. Se unen los clusters 2 y 6, para formar el cluster 8.
3. Se unen los clusters 4 y 8, para formar el cluster 9.
4. Se unen los clusters 7 y 9, para formar el cluster 10.
5. Se unen los clusters 1 y 10, para formar el cluster 11.

Para obtener una explicación detallada paso a paso tal y como se ha explicado en clase, usamos la función `agglomerativeHC.details`.

```
agglomerativeHC.details(m, 'EUC', 'MIN')

##   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 = 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
```

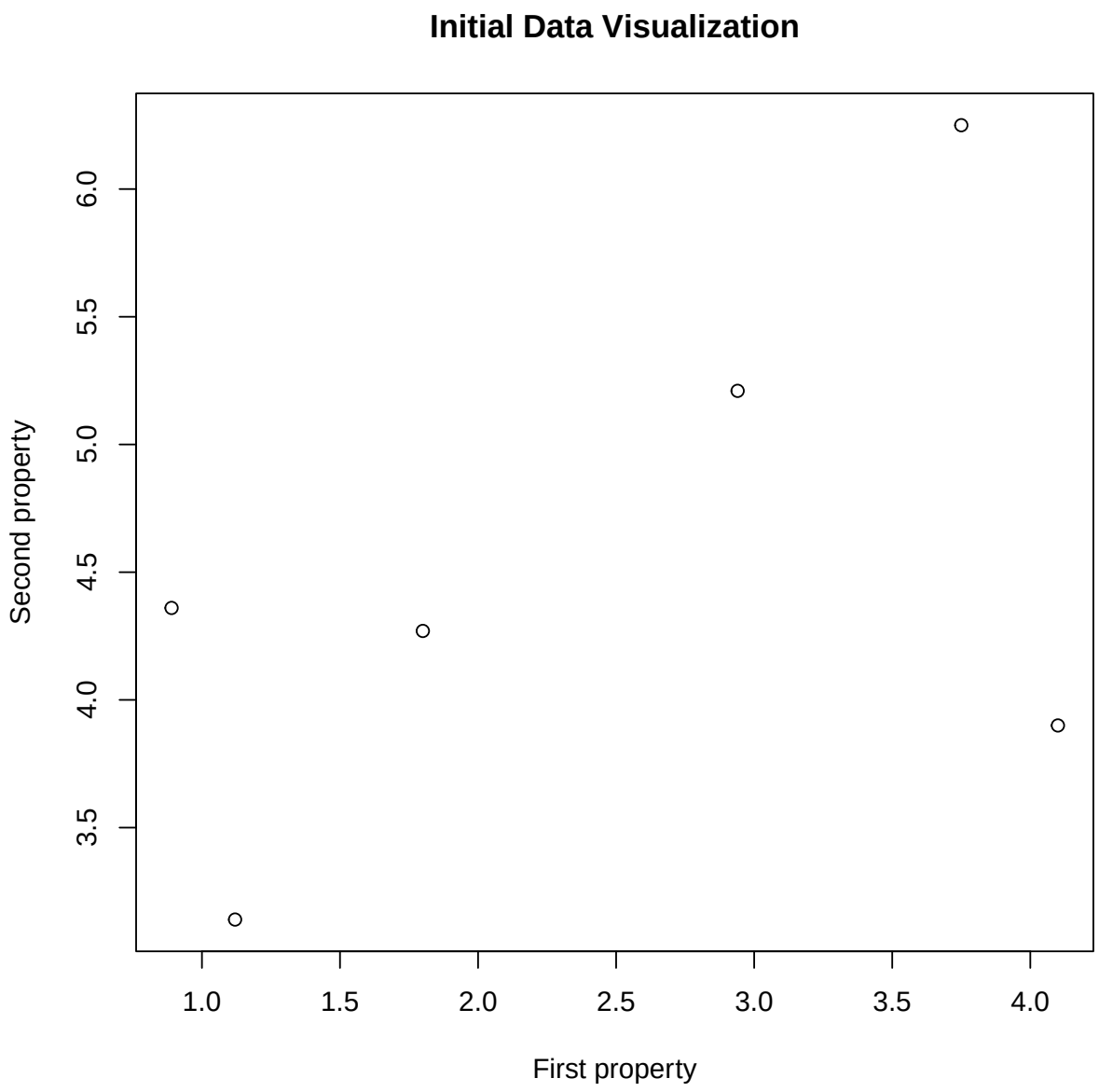
```
##
##  The new cluster is added to the solution.
##
## ------------------------------------------------------------------------------
## STEP =>5
##
##  Matrix Distance (distance type = EUC, approach type = MIN):

##           [,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.
```



La función también se puede ejecutar con diferentes criterios de unión, como 'MAX' y 'AVG', para observar cómo cambian los clusters:

```
agglomerativeHC.details(m, 'EUC', 'MAX')

agglomerativeHC.details(m, 'EUC', 'AVG')
```

1.3. Clasificación Supervisada con Árboles de Decisión

Ejercicio 1.3.1. *El tercer conjunto de datos, que se empleará para realizar el análisis de clasificación supervisada utilizando árboles de decisión, estará formado por las siguientes 9 calificaciones de estudiantes: 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}, donde las características de las calificaciones son: {Teoría, Laboratorio, Prácticas, Calificación Global).*

1.4. Clasificación Supervisada con Regresión

Ejercicio 1.4.1. *El cuarto conjunto de datos, que se empleará para realizar el análisis de clasificación supervisada utilizando regresión, estará formado por los siguientes 4 radios ecuatoriales y densidades de los planetas interiores: {Mercurio, 2.4, 5.4; Venus, 6.1, 5.2; Tierra, 6.4, 5.5; Marte, 3.4, 3.9}.*

Para empezar este ejercicio se deberán introducir los datos expuestos en el enunciado en un archivo de texto (`.txt`), con el fin de ser posteriormente leídos. La inserción de los datos en este fichero se hará atendiendo a las normas relacionadas con este tipo de archivos que ya se vieron en la primera práctica. Estas normas son las siguientes:

- Existirá una tabulación entre dato y dato.
- La primera columna numera las filas, y en la primera fila se introduce un espacio y el nombre de las variables.
- Se introducirá un salto de línea en la última fila.
- Para los números decimales se utilizarán puntos.
- Al escribir nombres, no se deberán introducir espacios.

Obedeciendo estas normas, se copian los datos en un fichero llamado `planetas.txt`, y se carga en R de la siguiente manera:

```
(muestra = read.table("data/planetas.txt"))

##      R   D
## 1. 2.4 5.4
## 2. 6.1 5.2
## 3. 6.4 5.5
## 4. 3.4 3.9
```

Una vez se tienen los datos en R, se procede a hacer uso de la función `lm` contenida en los paquetes básicos (concretamente en el paquete `stats`). Esta función recibe dos parámetros. El primero de ellos es la fórmula en donde se va a decir en función de qué parámetro se quiere otro parámetro. En este caso se tiene la columna `R` que representa el radio y la columna `D` que representa la densidad. Como se quiere la densidad en función del radio, el primer parámetro tendrá que ser formula=D∼R. Se indica de esta forma que se pretende obtener la columna `D` en función de la columna `R`; o lo que es lo mismo, la densidad en función del radio.

El segundo parámetro que entra a la función es la estructura que contiene los datos que se pretenden estudiar. En este caso, el parámetro `data` será la variable `muestra`, confeccionada previamente.

Con los parámetros de la función `lm` claros, se invoca a la misma de la siguiente forma:

```
(regresion=lm(D~R, data=muestra))

##
## Call:
## lm(formula = D ~ R, data = muestra)
##
## Coefficients:
## (Intercept)            R
##      4.3624       0.1394
```

En la salida de la función se observan los coeficientes que conforman la recta de regresión que mejor se adapta a los datos introducidos. El método de obtención o ajuste de la función es el de mínimos cuadrados, pudiendo comprobar que el resultado es el mismo que el que se ha visto en clase. En este método, los coeficientes se calculan de la siguiente forma, siendo x el radio e y la densidad:

$$b = \frac{s_{xy}}{s_x^2};\ a = \overline{y} - b\overline{x}$$

La salida proporciona directamente los valores de a y b, siendo estos el primero y el segundo respectivamente. Con estos valores se puede decir que la densidad de un planeta se puede sacar atendiendo a la siguiente fórmula:

$$D = 4{,}3624 + 0{,}1394R$$

Con el comando `summary` aplicado a la regresión que se acaba de hacer se pueden ver parámetros que detallan esta recta de regresión con respecto a los datos.

```
(summary(regresion))

##
## Call:
## lm(formula = D ~ R, data = muestra)
##
## Residuals:
##       1.       2.       3.       4.
##  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
```

Entre todos los detalles que aparecen, se observa el parámetro `Residuals`. Este parámetro devuelve un vector con los residuos, o dicho de otra forma, la distancia entre el valor real y el valor que se obtiene por medio de la recta calculada. También se puede apreciar el parámetro `Multiple R-squared`, el cual sirve como medida de cuán bueno ha sido el ajuste. Este valor podrá tomar valores entre 0 y 1, siendo 0 un ajuste muy malo y 1 un ajuste perfecto. Como se observa, el valor de este parámetro es 0.1377, lo cual quiere decir que el ajuste es malo; y con ello se puede concluir que el radio no explica la densidad de los planetas.

Una vez visto en detalle el cálculo y valoración de la recta de regresión se va a ver cómo se pueden identificar sucesos anómalos mediante la técnica de regresión. El proceso por lo general sigue 5 pasos:

1. Determinar el grado de outlier `d`
2. Obtener la ecuación de la recta de regresión
3. Obtener el error estándar s_r del vector de residuos
4. Calcular el límite para los valores típicos como $lim = d \cdot s_r$
5. Identificar como outliers los residuos (en valor absoluto) que superen ese límite

Hasta ahora se tiene la recta de regresión y el vector de residuos que está en `summary`. Para extraerlo y poder operar con él se hace de la siguiente forma:

```
(res=summary(regresion)$residuals)

##          1.          2.          3.          4.
##  0.70312301 -0.01253452  0.24565541 -0.93624389
```

Una vez se tiene el vector de residuos se calcula el error estándar del mismo:

```
(sr = sqrt(sum(res^2)/4))

## [1] 0.5982136
```

Con ello se puede plantear un bucle el cual compruebe qué elemento o elementos se presentan como anomalías. Se va a elegir como grado de outlier $d = 3$

```
for (i in 1:length(res)){
                  if(res[i]>3*sr){
                         print("el suceso");
                         print(res[i]);
                         print("es un suceso anómalo o outlier")
                  }
}
```

Este bucle comprueba por cada elemento del vector de residuos si supera o no el límite establecido para outliers; en caso afirmativo lo imprime por pantalla. En la salida anterior se puede observar que no hay ningún outlier, ya que ningún residuo supera el umbral establecido.

Se va a probar con otro conjunto de datos para demostrar que por medio de este método podemos identificar las anomalías. En primer lugar, se vuelve a confeccionar un archivo de texto (`.txt`) atendiendo a las normas previamente expuestas. Este archivo de texto se llamará `planetas2.txt` para evitar problemas de sobrescritura.

```
(muestra = read.table("data/planetas2.txt"))

##       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
```

Una vez hecho esto se invoca a la función `lm` para sacar la regresión y se guarda el vector de residuos tal y como se ha hecho previamente.

```
(dfr=lm(D~R, data=muestra))

##
## Call:
## lm(formula = D ~ R, data = muestra)
##
## Coefficients:
## (Intercept)            R
##     6.01445     -0.05723

(res=summary(dfr)$residuals)

##         1.         2.         3.         4.         5.         6.         7.
## -3.8427477  6.1858698 -1.6454482 -0.8168308  0.4919157 -0.4595958  0.0868370
```

Se vuelve a calcular el error estándar de los residuos. Atendiendo a la fórmula hay que dividir entre 7, ya que se tienen 7 entradas en el conjunto de datos del fichero.

```
(sr = sqrt(sum(res^2)/7))

## [1] 2.850242
```

Una vez se tienen todos los datos necesarios, se pone a prueba el código visto previamente para detectar anomalías. Esta vez se cambiará el grado de outlier $d = 2$.

```
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"
```

Se observa en la salida del código que el suceso 2 es un outlier. Atendiendo a los parámetros que se han ido obteniendo, se observa a simple vista que el suceso 2 supera el límite establecido.