

# 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

22 de diciembre de 2023

# Índice general

<b>Introducción</b>	<b>2</b>
<b>1. Ejercicios guiados</b>	<b>5</b>
1.1. Clasificación No Supervisada con K-Means . . . . .	5
1.2. Clasificación No Supervisada con CJA . . . . .	8
1.3. Clasificación Supervisada con Árboles de Decisión . . . . .	18
1.4. Clasificación Supervisada con Regresión . . . . .	20
<b>2. Ejercicios autónomos</b>	<b>25</b>
2.1. Clasificación No Supervisada con K-Means . . . . .	25

# Introducción

## RStudio

RStudio es un entorno de desarrollo integrado (IDE) para R, un lenguaje de programación ampliamente utilizado en estadística, análisis de datos, investigación científica y visualización de datos.

## De RStudio a Posit

El artículo del blog de Posit (<https://posit.co/blog/rstudio-is-now-posit/>), anteriormente conocido como RStudio, detalla los cambios y compromisos de la compañía a medida que evoluciona. Fundada en 2009, RStudio se centró en crear software de código abierto de alta calidad para científicos de datos. La compañía ha invertido significativamente en el desarrollo de código abierto, educación y la comunidad, con el objetivo de servir a los creadores de conocimiento durante los próximos 100 años.

La transformación de RStudio a Posit no solo implica un cambio de nombre, sino también una reafirmación de sus valores y objetivos. Como una Corporación de Beneficio Público y una B Corp certificada, Posit se compromete a cumplir con los más altos estándares de desempeño social y ambiental, transparencia y responsabilidad. Los directivos de Posit tienen la responsabilidad fiduciaria de abordar las necesidades sociales, económicas y ambientales, además de supervisar los objetivos comerciales de la empresa. La misión de Posit incluye mantener su independencia a largo plazo para cumplir con estos objetivos.

El compromiso de Posit con el código abierto sigue siendo una prioridad central. La empresa cree firmemente que el progreso significativo se logra poniendo herramientas de ciencia de datos al alcance de todos, independientemente de sus medios económicos. Además, Posit busca expandir su enfoque más allá del lenguaje de programación R, abrazando también a la comunidad de Python y a otras comunidades de programación. Aunque R sigue siendo un componente clave, el objetivo es mejorar la comunicación científica para todos, creando software tanto de código abierto como comercial para R, Python y más.

Finalmente, Posit desea fomentar una comunidad más amplia y diversa. Inspirándose en la comunidad de R, la compañía aspira a ayudar a que otras áreas de la ciencia sean tan abiertas, dinámicas, inclusivas y diversas como la comunidad a la que pertenecen. A pesar del cambio de nombre, la dedicación de Posit a ayudar a los científicos de datos a utilizar software

de código abierto para plantear y responder preguntas importantes permanece inalterada.

## Posit

La página web de Posit (<https://posit.co/>) ofrece una amplia gama de productos y soluciones para la ciencia de datos, enfocándose en el uso de R y Python. Presenta opciones de software de código abierto, soluciones empresariales y basadas en la nube. Destaca su compromiso con la ciencia de datos accesible para todos, ofreciendo recursos educativos, historias de clientes y soporte para la comunidad. La empresa busca promover un entorno inclusivo y empoderador, destacando su papel en el avance de la ciencia de datos a través de la colaboración y la innovación tecnológica.

## Entorno RStudio

La interfaz del IDE de RStudio está diseñada para facilitar el análisis estadístico y la visualización de datos.

- **File:** Aquí puedes crear nuevos archivos y proyectos, incluidos scripts de Python y documentos Sweave. Trabajar con proyectos es una práctica profesional estándar y permite una mejor organización.
- **Edit:** Esta sección funciona de manera similar a otros entornos de desarrollo, proporcionando herramientas de edición de texto estándar.
- **Code:** Incluye herramientas específicas para escribir y ejecutar código R, como insertar operadores o ejecutar scripts completos.
- **View:** Permite ajustar la apariencia de la interfaz de RStudio, como mostrar u ocultar diferentes paneles.
- **Plots:** Muy utilizado para la visualización de datos, permite generar, ver y manipular gráficos estadísticos.
- **Session:** Controla las sesiones de R, permitiendo iniciar, interrumpir o terminar sesiones.
- **Build:** Herramientas relacionadas con la construcción de paquetes de R.
- **Debug:** Funciones para depurar el código R.
- **Profile:** Herramientas para medir el rendimiento del código R.
- **Tools:** Opciones para configurar el entorno de desarrollo globalmente, como la disposición del panel y opciones de proyecto.
- **Help:** Ofrece documentación y ayuda para usar R y RStudio, un recurso esencial para el aprendizaje y la solución de problemas.

El entorno de RStudio es altamente personalizable y se organiza en cuatro áreas principales. Puedes redactar código en el panel de **scripts**, ejecutar instrucciones en el **console**, y ver los objetos activos y el historial en el área de **environment/history**. El cuarto panel es multifuncional, permitiendo administrar archivos, visualizar datos a través de gráficos, instalar y administrar paquetes y buscar en la documentación en la sección **files/plots/packages/help/viewer**. Este diseño modular facilita la gestión eficiente del flujo de trabajo en análisis de datos y desarrollo estadístico.



Figura 1: Logo de la compañía Posit



Figura 2: Logo del entorno RStudio

# 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 centroeide 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 características 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))

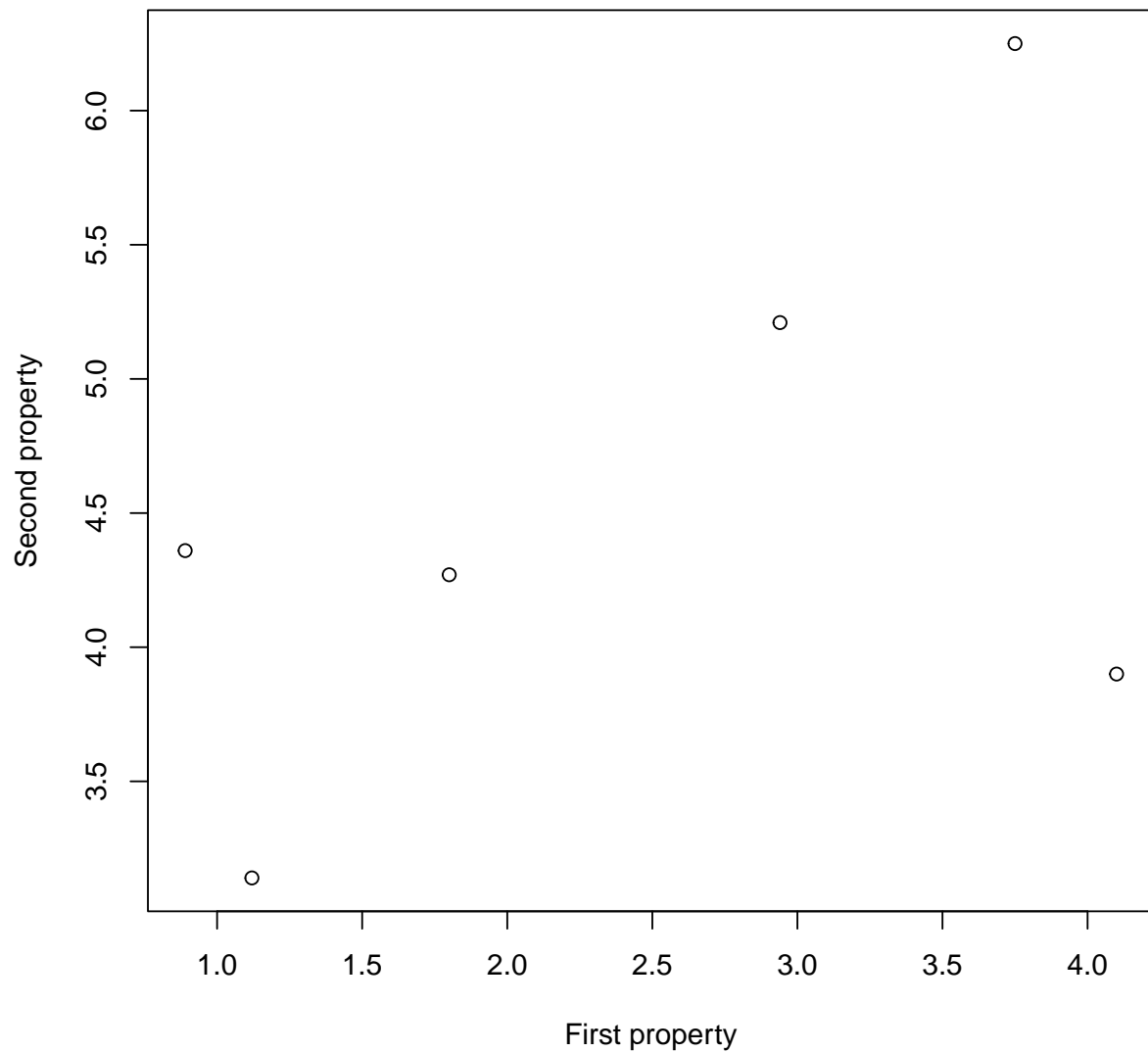
##      [,1] [,2]
```

```
## [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')
```

### Initial Data Visualization



```
## $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

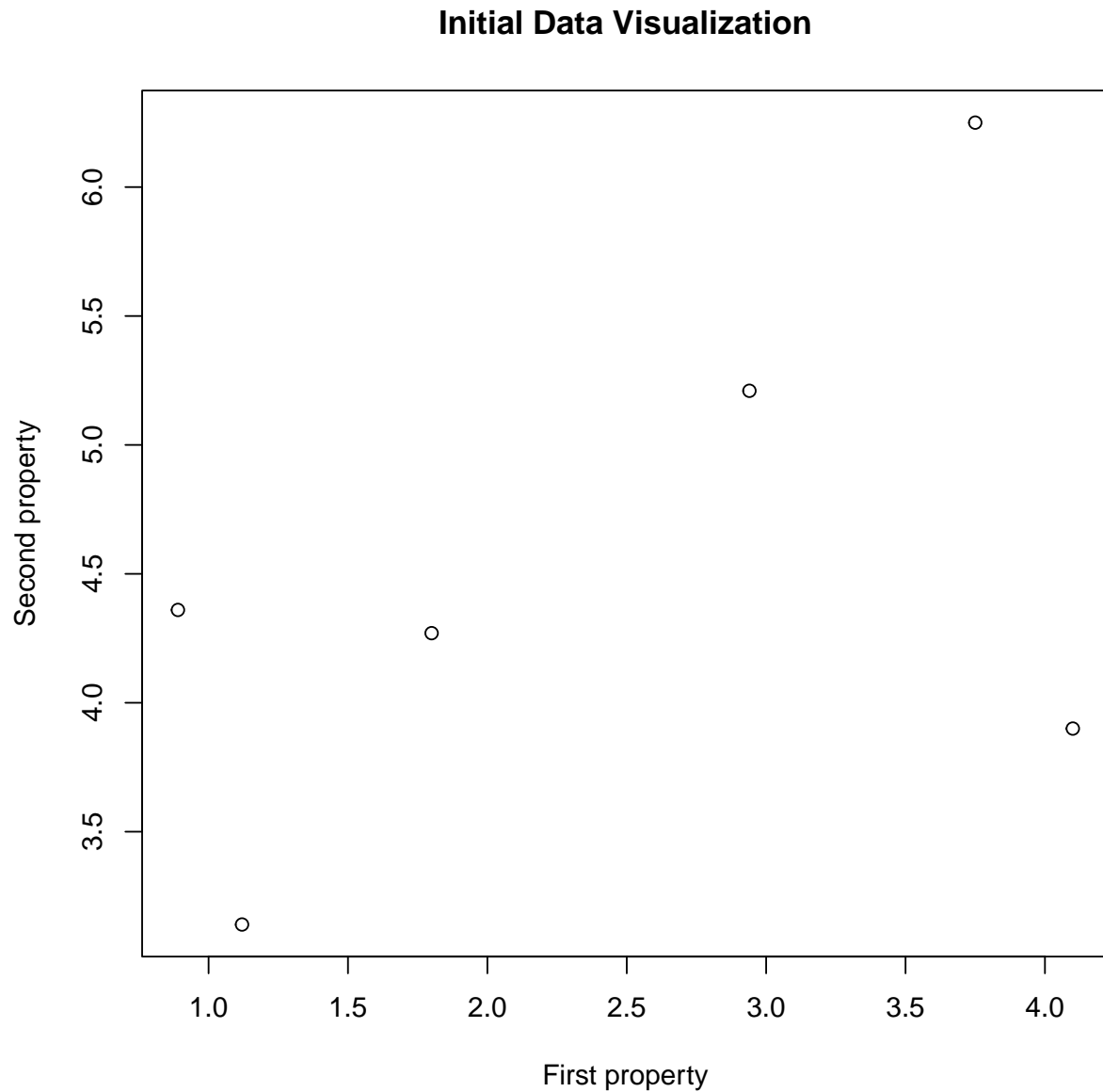
```



```

##
## 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}.*

El primer paso a ejecutar para poder realizar la clasificación supervisada con árboles de decisión es introducir los datos en un archivo de texto (.txt) que será posteriormente leído con el lenguaje R. Para ello, es necesario que el archivo de texto cumpla con un formato en concreto:

- La primera fila contendrá los nombres de las columnas, iniciando la fila con un tabulado o espacio e introduciendo otro entre cada una de las columnas.
- La primera columna contendrá los nombres de cada una de las filas del dataframe.
- Cada elemento de una fila estará separado del resto con un tabulado o espacio.
- El archivo de texto debe finalizar con una fila vacía.

Este archivo de texto se llamará `aprobados.txt` y será guardado en la carpeta `data`. El archivo es cargado en R de la siguiente forma:

```
(calificaciones=read.table("data/aprobados.txt"))

##      T L P CG
## 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
```

Para la implementación del algoritmo, se emplea el paquete `rpart`, el cual se encuentra disponible en el repositorio `CRAN`. Este paquete ofrece un método homónimo que recibe por parámetro, entre otros, los datos en forma de dataframe. Por ello, es necesario poseer los datos en esta forma en lugar de en una tabla:

```
library(rpart)

## Warning: package 'rpart' was built under R version 4.3.2
```

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

##      T L P CG
## 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

clasificacion = rpart(CG~T+L+P, data=muestra, method="class", minsplit=1)
```

El primer argumento que se le pasa es `CG T+L+P`, que contiene como primer elemento (a la izquierda de `~`) el criterio que se desea clasificar, y a la derecha de la virgulilla, aparecen las columnas sobre las que se desea clasificar. También es posible poner un punto (`.`) a la derecha, lo cual significa que se clasificaría en base a todas las columnas que no fueran el criterio de clasificación. El segundo parámetro es, como ya se ha mencionado, la muestra en un dataframe, el tercer parámetro `method="class"` indica que se está abordando un problema de clasificación, y el último parámetro `minsplit=1` es necesario ya que se está trabajando sobre un conjunto de datos pequeño.

La salida de la función es la siguiente:

```
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) *
```

Se puede apreciar que el árbol consta de dos niveles además de la raíz. El primero de ellos clasifica según la nota en laboratorio, en caso de ser `C` o `D`, la calificación final es `Ss`, en caso de ser `A` o `B` es necesario observar el siguiente nivel. Este segundo nivel clasifica según la nota de prácticas, si la nota es `A` o `B`, la calificación es `Ap` y si es `C` o `D`, la calificación final es `Ss`.

## 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:

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

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.

## Parte 2

### Ejercicios autónomos

En esta segunda parte de la práctica se implementarán los algoritmos vistos anteriormente, plasmando en el documento la explicación del código realizado, así como su aplicación a los enunciados propuestos para esta parte de la práctica.

#### 2.1. Clasificación No Supervisada con K-Means

**Ejercicio 2.1.1.** *El primer conjunto de datos, que se empleará para realizar el análisis de clasificación no supervisada con Kmeans, estará formado por los siguientes 15 valores de velocidades de respuesta y temperaturas normalizadas de un microprocesador {Velocidad, Temperatura}: 1.{3.5, 4.5}; 2.{0.75, 3.25}; 3.{0, 3}; 4.{1.75, 0.75}; 5.{3, 3.75}; 6.{3.75, 4.5}; 7.{1.25, 0.75}; 8.{0.25, 3}; 9.{3.5, 4.25}; 10.{1.5, 0.5}; 11.{1, 1}; 12.{3, 4}; 13.{0.5, 3}; 14.{2, 0.25}; 15.{0, 2.5}. Del análisis visual de los datos se ha concluido que hay una alta probabilidad que sean tres clusters.*

Los datos que se proponen en el enunciado del ejercicio van a ser introducidos en un fichero Excel (`.xlsx`), de tal forma que estos se van a obtener leyendo este fichero desde R. Para ello, se hará uso de la función `read.xlsx` contenida en el paquete `openxlsx` que se ha visto previamente. Con esta función se puede pasar como parámetro la ruta del archivo que contiene los datos, devolviendo los mismos en un dataframe.

Para este ejercicio en concreto se hará uso de dos ficheros, uno con los datos de velocidad y temperatura (dispuestos cada uno en una columna del Excel), y otro fichero que contenga las coordenadas de los centroides iniciales que se van a utilizar en el algoritmo. En la primera fila de ambos ficheros deberá ir el título de cada columna.

Primeramente se leen los datos del ejercicio. A partir de ahora, a cada uno de estos datos se les conocerán como puntos.

```
(data = data.frame(read.xlsx("data/datosKMeans.xlsx")))  
  
## Error in read.xlsx("data/datosKMeans.xlsx"): no se pudo encontrar la función  
"read.xlsx"
```

Y posteriormente se lee el fichero con los centroides iniciales. En la clasificación final se tendrán tantos clusters como centroides se hayan introducido, ya que al final un centroide representa un cluster.

```
(centroides = data.frame(read.xlsx("data/centroides.xlsx")))
```

```
## Error in read.xlsx("data/centroides.xlsx"): no se pudo encontrar la función
"read.xlsx"
```

Una vez se tienen todos los datos en el entorno de trabajo se va a proceder a explicar la implementación del algoritmo. Previo a ello se necesitan dos funciones auxiliares que ya se han visto en la anterior práctica. Estas son la función `len` para determinar la longitud de una lista y `euc_distance`, a la cual se pasan dos puntos y devuelve la distancia euclídea entre los mismos.

```
len = function(list) {
  count = 0
  for (element in list) {
    count = count + 1
  }
  count
}

euc_distance = function(p1, p2) {
  sqrt(((p1[1] - p2[1])^2) + ((p1[2] - p2[2])^2))
}
```

El primer paso del algoritmo consiste en realizar una matriz de distancias. Esta matriz tendrá tantas filas como centroides se hayan declarado y tantas columnas como puntos se hayan introducido. Esta matriz recoge las distancias euclídeas de todos los puntos a todos los centroides. Para realizar esta tabla se ha programado la siguiente función, la cual va rellenando una matriz con las distancias euclídeas de cada punto a cada cluster.

```
create_distance_matrix = function(points, centroids) {
  n_pts = nrow(points)
  n_cent = nrow(centroids)
  distances = data.frame(matrix(ncol = n_pts, nrow = n_cent))
  for (cent in 1:n_cent) {
    for (pt in 1:n_pts) {
      distances[cent, pt] =
        euc_distance(points[pt,], centroids[cent,])
    }
  }
  distances
}
```

Una vez se tiene la matriz de distancias se debe realizar una matriz de asignaciones. Esta matriz será de las mismas dimensiones que la anterior, pero ahora no contendrá las distancias euclídeas. Por cada punto (columna) tendremos un 0 o un 1. Si aparece un 1 en una posición, la distancia del punto a ese cluster que representa la fila es la mínima. Por ejemplo; si el punto 1 esta más cerca del cluster 2 que del resto de clusters, la posición 1x2 tendrá un 1, y el resto de la columna un 0.

Para realizar esta tabla se itera punto por punto, y por cada uno se observa la distancia del mismo a todos los clusters. Se debe poner un 1 en el lugar donde se encuentra la distancia euclídea mínima y un 0 en el resto de celdas de la columna. Es por ello que primero se inicializa a 0 toda la matriz y se va buscando por cada punto el cluster que está a menos distancia, para poner un 1 en su celda correspondiente. El código que recoge esta funcionalidad es el siguiente:

```
create_assignment_matrix = function(dist_matrix) {
  n_pts = ncol(dist_matrix)
  n_cent = nrow(dist_matrix)
  assig_matrix = data.frame(matrix(0, ncol = n_pts, nrow = n_cent))
  for (col in 1:n_pts) {
    minimum_centroid = 1
    for (cent in 2:n_cent) {
      if (dist_matrix[cent, col] <
          dist_matrix[minimum_centroid, col]) {
        minimum_centroid = cent
      }
    }
    assig_matrix[minimum_centroid, col] = 1
  }
  assig_matrix
}
```

Por último, una vez se han asignado los puntos más próximos a cada cluster, se deben recalcular los centroides del mismo. Para recalcular un centroide en específico se deben tener en cuenta todos los puntos asociados al mismo. Las nuevas coordenadas del centroide se calculan realizando la media de los puntos que tiene asignados. El resultado de este algoritmo resulta en un nuevo dataframe donde, por cada fila, se van a tener las nuevas componentes del centroide correspondiente, y que serán usados para continuar con la ejecución del algoritmo.

Antes de ver el código, se ha de considerar que un centroide puede no tener ningún punto asociado, por lo que el cálculo que se propone arriba no sería posible (ya que se tendría que dividir entre cero a la hora de calcular la media). Es por ello que si un centroide no tiene ningún punto asociado, este se quedará igual (sus componentes no se verán alteradas). Esto resultará en que el centroide propuesto no se ha definido muy bien, y desemboca en una clasificación diferente a la esperada.

```

update_cent = function(df_sample, centroids, assig_matrix) {
  n_pts = ncol(assig_matrix)
  n_cent = nrow(assig_matrix)
  new_centroids = c()
  for (cent in 1:n_cent) {
    centroid_x = 0
    centroid_y = 0
    count = 0
    for (pt in 1:n_pts) {
      if (assig_matrix[cent, pt] == 1) {
        centroid_x = centroid_x + df_sample[pt, 1]
        centroid_y = centroid_y + df_sample[pt, 2]
        count = count + 1
      }
    }
    if (count == 0){
      new_centroids = append(append(new_centroids,
        centroids[cent,1]),centroids[cent,2])
    }
    else {
      new_centroids = append(append(new_centroids,
        centroid_x/count), centroid_y/count)
    }
  }
  data.frame(t(matrix(new_centroids, 2, n_cent,
    dimnames=list(c("X","Y")))))
}

```

El algoritmo se ejecutará hasta que la matriz de asignaciones sea la misma en una iteración con respecto a su anterior. Para visualizar el resultado o clasificación final se propone una función que tomará la matriz de asignaciones final y el valor de los centroides finales. Este método imprimirá las coordenadas de cada centroe, así como los puntos que conforman el cluster representado por ese centroe.

```

printCentroids = function(df_sample, centroids, assig_matrix){
  n_cents = nrow(assig_matrix)
  n_pts = ncol(assig_matrix)
  cat("\n--RESULTADOS--\n")
  for (cent in 1:n_cents){
    cat("Centroe", cent, "(", centroids[cent,1], ", ", centroids[cent,2], ") \n")
    for(pt in 1:n_pts){
      if (assig_matrix[cent,pt] == 1){
        cat(" - Punto", pt, "(", df_sample[pt, 1], ", ", df_sample[pt,2], ") \n")
      }
    }
    cat("\n")
  }
}

```

De todo este algoritmo, se ha considerado que la salida mas útil para el analista es un vector de clusterización. Este consiste en un vector de tantos elementos como puntos se hayan introducido, de tal forma que la posición  $i$ -ésima del mismo corresponda con el cluster asociado al punto  $i$ . Se itera la matriz de asignaciones final por puntos, y por cada uno de

ellos se devuelve el valor de la fila que contiene un 1 (el número del centroide al que se ha asociado en la clasificación final)

```
create_clust_vector = function(assig_matrix){
  n_pts = ncol(assig_matrix)
  n_cents = nrow(assig_matrix)
  clust_vector = c()
  for(pt in 1:n_pts){
    for(cent in 1:n_cents){
      if (assig_matrix[cent,pt] == 1){
        clust_vector = append(clust_vector, cent)
      }
    }
  }
  clust_vector
}
```

Todas estas funciones se recogen en la función `fcd_kmeans`. Esta función será la que implemente el algoritmo y a la que se deberá llamar. Recibirá como parámetros un dataframe con los datos de la muestra, otro dataframe con los centroides iniciales y un booleano `details` el cual se usará para recibir una salida más o menos detallada. En caso de que se quiera una salida detallada, se imprimirán paso por paso la matriz de distancias, la matriz de asignaciones y los valores de los nuevos centroides en ese paso concreto. En caso contrario simplemente se imprimirá la clasificación final (se llama a la función `printCentroids`) y se devolverá el vector de clusterización. Por defecto, `details` se encuentra a `FALSE`, ofreciendo una salida no detallada en caso de no indicar este parámetro en la llamada. El usuario podrá asignar a una variable la llamada a la función, recibiendo en la misma el vector de clusterización.

```
fcd_kmeans = function(df_sample, centroids, details = FALSE) {
  n_pts = nrow(df_sample)
  n_cent = nrow(centroids)

  prev_assig_matrix = data.frame(
    matrix(0, ncol = n_pts, nrow = n_cent))

  step = 1
  dist_matrix = create_distance_matrix(df_sample, centroids)
  assig_matrix = create_assignment_matrix(dist_matrix)
  if(details){
    cat("PASO", step, "\n----- \n")
    cat("Matriz de distancias \n")
    print(dist_matrix)
    cat("Matriz de asignaciones \n")
    print(assig_matrix)
  }
}
```

```

while (!identical(prev_assig_matrix, assig_matrix)) {
  prev_assig_matrix = assig_matrix
  centroids = update_cent(df_sample, centroids, assig_matrix)
  dist_matrix = create_distance_matrix(df_sample, centroids)
  assig_matrix = create_assignment_matrix(dist_matrix)

  if(details) {
    cat("Los nuevos centroides son:\n")
    print(centroids)
    step = step + 1
    cat("PASO", step, "\n----- \n")
    cat("Matriz de distancias \n")
    print(dist_matrix)
    cat("Matriz de asignaciones \n")
    print(assig_matrix)
  }
}
printCentroids(df_sample, centroids, assig_matrix)
cat("Vector de clusterización\n")
create_clust_vector(assig_matrix)
}

```

Un aspecto a destacar es el uso de la función `identical` contenida en el paquete `base`. Esta función permite comparar si dos dataframes son iguales. Esto sirve en el algoritmo para ver si la matriz de asignaciones de la anterior iteración es igual que la actual. Esta condición será la que se deba cumplir para que el algoritmo finalice.

Con esto ya estaría el algoritmo implementado. Ahora se va a llamar a la función `fcd_kmeans` con los datos leídos al principio. Se va a optar por una salida detallada.

```

fcd_kmeans(data, centroides, TRUE)

## Error in eval(expr, envir, enclos): objeto 'centroides' no encontrado

```

Se ha optado por tres centroides que más o menos se encuentran entre el conjunto total de puntos. Han sido necesarios 4 pasos para llegar a una solución. Se puede ver cómo se han ido calculando las matrices de distancias y asignaciones, así como los nuevos centroides, hasta que se ha llegado a los resultados finales. En estos se aprecian las coordenadas de los tres centroides así como los puntos que contienen cada uno. Por último, se devuelve el vector de clusterización que también se puede ver en la salida de la función realizada.