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

25 de diciembre de 2023

Índice general

Índice de figuras

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. Será empleado para el desarrollo de esta práctica. 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 indicada a continuación.

`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 del entorno RStudio

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.

Figura 2: Logo de la compañía Posit

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.

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

1.1.1. $k-$means

Ejercicio 1.1.1. *El primer conjunto de datos, que se empleará para realizar el análisis de clasificación no supervisada con $k-$means, 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 clústers o grupos, creando de esta forma una clasificación del conjunto total. Estos clústers se determinarán en base a la muestra, obteniéndose durante el mismo proceso de clasificación.

El algoritmo $k-$means ofrece una técnica de clústerización en base a una muestra de datos y una cantidad n de clústers. Para empezar a realizar el algoritmo se necesita la ubicación de los centroides de cada clúster. Un centroide es el punto medio del grupo de sucesos que componen el clúster, por lo que habrá tantos centroides como n clústers haya. La cantidad de clústers 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 clúster, 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 clústers. Se introducen así los centroides:

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

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

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 clústers (`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 clústers que conforman la clasificación final. Como se puede apreciar, salen los dos centroides que salieron en teoría. El segundo parámetro indica a qué clúster pertenece cada suceso de la muestra introducida. Así el punto 1 pertenece al clúster 2, el punto 2 pertenece al clúster 2, el punto 3 al clúster 1... Este último se puede utilizar como entrada para realizar otras tareas. Puede ser muy útil para analizar cada clúster 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 clúster pertenece cada suceso.

```
(m = cbind(clasificacionns$clúster,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
```

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 clúster 1 y la porción restante que pertenece al clúster 2.

```
(mc1 = subset(m,m[,1]==1))
```

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

```
(mc2 = subset(m,m[,1]==2))
```

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

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

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

## [1] 2
```

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

1.1.2. Clusterización jerárquica algomerativa

Ejercicio 1.1.2. *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 clúster individual y luego, iterativamente, combina los clústers más cercanos hasta formar un único clúster. 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 clústers. 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 `$dendrogram`, en este caso 6.
- El tercer apartado es una lista que muestra las coordenadas de todos los clústers al finalizar la ejecución, los 6 primeros son los puntos introducidos y los 5 siguientes son los formados por cada una de las iteraciones uniendo dos clústers anteriores hasta tener todos unidos en el mismo clúster, momento en el que finaliza el algoritmo. Se puede ver en el atributo `$clústers`.

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

  1. Se unen los clústers 3 y 5, para formar el clúster 7.
  2. Se unen los clústers 2 y 6, para formar el clúster 8.
  3. Se unen los clústers 4 y 8, para formar el clúster 9.
  4. Se unen los clústers 7 y 9, para formar el clúster 10.
  5. Se unen los clústers 1 y 10, para formar el clúster 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):









##
##  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:




##
##  The new cluster is added to the solution.
##
## ------------------------------------------------------------------------------
## STEP =>2
##
##  Matrix Distance (distance type = EUC, approach type = MIN):









##
##  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.
```

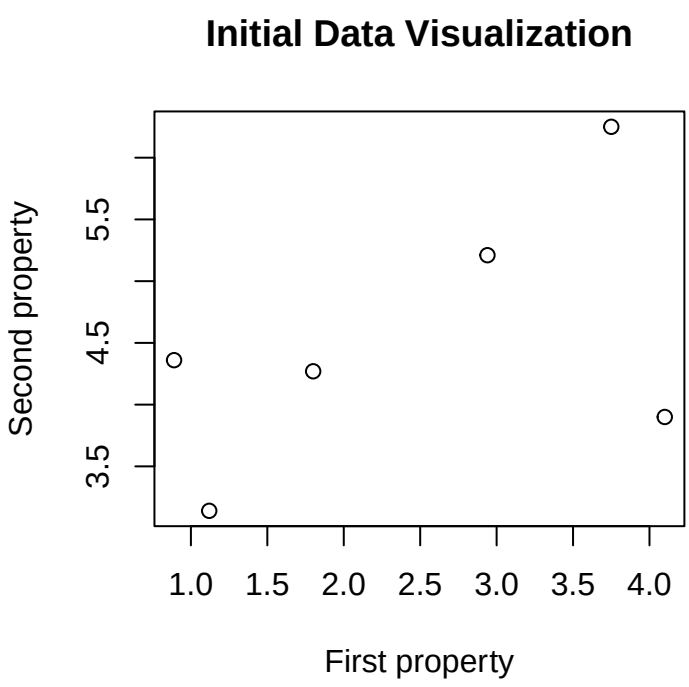


Figura 1.1: Diagrama de LearnClust

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

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

1.2. Clasificación supervisada

1.2.1. Árboles de decisión

Ejercicio 1.2.1. *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:

```
(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`.

1.2.2. Regresión lineal

Ejercicio 1.2.2. *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 de la recta $\hat{y} = a + bx$ 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 obtener 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 $d \cdot s_r$

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

2.1.1. $k-$means

Ejercicio 2.1.1. *El primer conjunto de datos, que se empleará para realizar el análisis de clasificación no supervisada con $k-$means, 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 clústers.*

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")))

##    Velocidad Temperatura
## 1       3.50        4.50
```

```
## 2       0.75        3.25
## 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
```

Posteriormente se lee el fichero con los centroides iniciales. En la clasificación final se tendrán tantos clústers como centroides se hayan introducido, ya que al final un centroide representa un clúster.

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

```
##     X   Y
## 1 0.5 0.5
## 2 1.0 2.0
## 3 2.0 1.0
```

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 clúster.

```
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 clúster que representa la fila es la mínima. Por ejemplo; si el punto 1 esta más cerca del clúster 2 que del resto de clústers, la posición a_{12} 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 clústers. 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 clúster que está a menos distancia, para poner un 1 en su celda correspondiente. El código que recoge esta funcionalidad es el siguiente:

```
create_assigment_matrix = function(dist_matrix) {
                                        n_pts = ncol(dist_matrix)
                                        n_cent = nrow(dist_matrix)
                                        assig_matrix = data.frame(matrix(0, ncol = n_pts, nrow
                                        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 clúster, 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_s
                                                                centroid_y = centroid_y + df_s
                                                                count = count + 1
                                                        }
                                                }
                                                if (count == 0){
                                                                new_centroids = append(append(n
            centroids[cent,1]),centroids[cent,2])
                                                }
                                                else {
                                                                new_centroids = append(append(n
            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 centroide, así como los puntos que conforman el clúster representado por ese centroide.

```
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("Centroide", 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 clústerización. 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 clúster 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
}
```

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 clústerización. 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 clústerización.

```
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, centr
                                        assig_matrix = create_assigment_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,
                                                dist_matrix = create_distance_matrix(df_sample
                                                assig_matrix = create_assigment_matrix(dist_ma

                                                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 clústerización\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)
```

```
## PASO 1
## -------
## Matriz de distancias
##         X1       X2       X3        X4       X5       X6        X7       X8
## 1 5.000000 2.761340 2.549510 1.2747549 4.100305 5.153882 0.7905694 2.512469
## 2 3.535534 1.274755 1.414214 1.4577380 2.657536 3.716517 1.2747549 1.250000
## 3 3.807887 2.573908 2.828427 0.3535534 2.926175 3.913119 0.7905694 2.657536
##         X9       X10       X11      X12      X13      X14      X15
## 1 4.802343 1.0000000 0.7071068 4.301163 2.500000 1.520691 2.061553
## 2 3.363406 1.5811388 1.0000000 2.828427 1.118034 2.015564 1.118034
## 3 3.579455 0.7071068 1.0000000 3.162278 2.500000 0.750000 2.500000
## Matriz de asignaciones
##   X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15
## 1  0  0  0  0  0  0  1  0  0   0   1   0   0   0   0
## 2  1  1  1  0  1  1  0  1  1   0   0   1   1   0   1
## 3  0  0  0  1  0  0  0  0  0   1   0   0   0   1   0
## Los nuevos centroides son:
##       X     Y
## 1 1.125 0.875
## 2 1.825 3.575
## 3 1.750 0.500
## PASO 2
## -------
## Matriz de distancias
##         X1       X2       X3        X4       X5       X6        X7       X8
## 1 4.333734 2.404423 2.404423 0.6373774 3.432383 4.475628 0.1767767 2.298097
## 2 1.913439 1.123054 1.913439 2.8259954 1.187960 2.135708 2.8829239 1.676678
## 3 4.366062 2.926175 3.051639 0.2500000 3.482097 4.472136 0.5590170 2.915476
##         X9       X10       X11      X12      X13       X14      X15
## 1 4.126894 0.5303301 0.1767767 3.644345 2.215006 1.0752907 1.976424
## 2 1.805893 3.0921271 2.7039323 1.249500 1.444386 3.3296021 2.118077
## 3 4.138236 0.2500000 0.9013878 3.716517 2.795085 0.3535534 2.657536
## Matriz de asignaciones
##   X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15
## 1  0  0  0  0  0  0  1  0  0   0   1   0   0   0   1
## 2  1  1  1  0  1  1  0  1  1   0   0   1   1   0   0
## 3  0  0  0  1  0  0  0  0  0   1   0   0   0   1   0
## Los nuevos centroides son:
##          X        Y
## 1 0.750000 1.416667
## 2 2.027778 3.694444
## 3 1.750000 0.500000
## PASO 3
## -------
## Matriz de distancias
```

```
##         X1       X2       X3       X4        X5       X6        X7       X8
## 1 4.131518 1.833333 1.751983 1.201850 3.2414417 4.301970 0.8333333 1.660405
## 2 1.678201 1.352866 2.143394 2.957518 0.9738082 1.901307 3.0454378 1.908598
## 3 4.366062 2.926175 3.051639 0.250000 3.4820971 4.472136 0.5590170 2.915476
##         X9      X10       X11      X12      X13       X14      X15
## 1 3.948453 1.184389 0.4859127 3.425801 1.602949 1.7098570 1.317616
## 2 1.573557 3.237750 2.8838096 1.019108 1.678201 3.4445564 2.353419
## 3 4.138236 0.250000 0.9013878 3.716517 2.795085 0.3535534 2.657536
## Matriz de asignaciones
##   X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15
## 1  0  0  1  0  0  0  0  1  0   0   1   0   1   0   1
## 2  1  1  0  0  1  1  0  0  1   0   0   1   0   0   0
## 3  0  0  0  1  0  0  1  0  0   1   0   0   0   1   0
## Los nuevos centroides son:
##          X        Y
## 1 0.350000 2.500000
## 2 2.916667 4.041667
## 3 1.625000 0.562500
## PASO 4
## -------
## Matriz de distancias
##          X1       X2        X3       X4        X5        X6        X7       X8
## 1 3.7312866 0.850000 0.6103278 2.241093 2.9300171 3.9446166 1.9678669 0.509902
## 2 0.7418539 2.306768 3.0970977 3.492303 0.3033379 0.9510594 3.6895592 2.862897
## 3 4.3611388 2.826355 2.9295104 0.225347 3.4714235 4.4743191 0.4192627 2.798577
##          X9       X10       X11       X12       X13       X14      X15
## 1 3.6034705 2.3070544 1.6347783 3.0450780 0.5220153 2.7901613 0.350000
## 2 0.6194195 3.8144917 3.5951839 0.0931695 2.6316054 3.9008991 3.299042
## 3 4.1368202 0.1397542 0.7629097 3.7023008 2.6845914 0.4881406 2.528741
## Matriz de asignaciones
##   X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15
## 1  0  1  1  0  0  0  0  1  0   0   0   0   1   0   1
## 2  1  0  0  0  1  1  0  0  1   0   0   1   0   0   0
## 3  0  0  0  1  0  0  1  0  0   1   1   0   0   1   0
## Los nuevos centroides son:
##      X    Y
## 1 0.30 2.95
## 2 3.35 4.20
## 3 1.50 0.65
## PASO 5
## -------
## Matriz de distancias
##          X1        X2        X3        X4        X5       X6        X7
## 1 3.5556293 0.5408327 0.3041381 2.6348624 2.8160256 3.782195 2.3963514
## 2 0.3354102 2.7681221 3.5584407 3.8029594 0.5700877 0.500000 4.0388736
## 3 4.3384905 2.7060118 2.7879204 0.2692582 3.4438351 4.459260 0.2692582
##           X8        X9       X10       X11       X12       X13       X14
```

```
## 1 0.07071068 3.4539832 2.728095 2.0718349 2.8969812 0.2061553 3.1906112
## 2 3.32415403 0.1581139 4.136726 3.9702015 0.4031129 3.0923292 4.1743263
## 3 2.66176633 4.1182521 0.150000 0.6103278 3.6704904 2.5539186 0.6403124
##         X15
## 1 0.5408327
## 2 3.7566608
## 3 2.3817011
## Matriz de asignaciones
##   X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15
## 1  0  1  1  0  0  0  0  1  0   0   0   0   1   0   1
## 2  1  0  0  0  1  1  0  0  1   0   0   1   0   0   0
## 3  0  0  0  1  0  0  1  0  0   1   1   0   0   1   0
##
## --RESULTADOS--
## Centroide 1 ( 0.3 , 2.95 )
##   - Punto 2 ( 0.75 , 3.25 )
##   - Punto 3 ( 0 , 3 )
##   - Punto 8 ( 0.25 , 3 )
##   - Punto 13 ( 0.5 , 3 )
##   - Punto 15 ( 0 , 2.5 )
##
## Centroide 2 ( 3.35 , 4.2 )
##   - Punto 1 ( 3.5 , 4.5 )
##   - Punto 5 ( 3 , 3.75 )
##   - Punto 6 ( 3.75 , 4.5 )
##   - Punto 9 ( 3.5 , 4.25 )
##   - Punto 12 ( 3 , 4 )
##
## Centroide 3 ( 1.5 , 0.65 )
##   - Punto 4 ( 1.75 , 0.75 )
##   - Punto 7 ( 1.25 , 0.75 )
##   - Punto 10 ( 1.5 , 0.5 )
##   - Punto 11 ( 1 , 1 )
##   - Punto 14 ( 2 , 0.25 )
##
## Vector de clústerización
##  [1] 2 1 1 3 2 2 3 1 2 3 3 2 1 3 1
```

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 clústerización que también se puede ver en la salida de la función realizada.

2.1.2. Clusterización jerárquica algomerativa

Ejercicio 2.1.2. *El conjunto de datos de datos que se empleará para realizar el análisis de clasificación no supervisada con Clusterización Jerárquica Aglomerativa será el mismo que el utilizado en el Ejercicio 2.1.1, pero el este ejercicio le denominaremos segundo conjunto de datos.*

Para la resolución de este ejercicio se implementará una variación del algoritmo visto en clase, que ayudará a simplificar la codificación y el entendimiento del algoritmo. Se utilizarán algunas funciones ya explicadas en otros ejercicios y prácticas, como `len`, la media, la distancia euclídea, la desviación típica, y la covarianza. El primer paso del algoritmo consiste en calcular la distancia entre cada posible pareja de puntos.

```
create_distance_matrix = function(df) {
        n = len(df[,1])
        empty_matrix = matrix(0, ncol = n, nrow = n)
        distances = data.frame(empty_matrix)
        for (i in 1:n) {
                for (j in 1:i) {
                        distances[i, j] = euc_distance(df[i,], df[j,])
                        if (i != j) distances[j, i] = NA
                }
        }
        distances
}
```

Esto se logra mediante la función `create_distance_matrix` donde la entrada a_{ij} representa el valor de $\text{dist}(x_i, x_j)$. Nótese que $\text{dist}(x_i, x_j) = \text{dist}(x_j, x_i)$, por lo que $a_{ij} = a_{ji}$, y por tanto la matriz es simétrica. Para ahorrar cálculos y simplificar el problema, se ignorarán las entradas a_{ij} con $j > i$. Durante la explicación de este ejercicio se ejemplificará con los datos del ejercicio visto en teoría, pues los datos de laboratorio son mucho más extensos. De acuerdo a esto, la matriz de distancias se vería como la siguiente.

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{array} \\ \begin{pmatrix} 0 & \times & \times & \times & \times & \times \\ 4{,}15 & 0 & \times & \times & \times & \times \\ 3{,}39 & 4{,}13 & 0 & \times & \times & \times \\ 5{,}36 & 2{,}8 & 3{,}21 & 0 & \times & \times \\ 3{,}41 & 3{,}42 & 0{,}76 & 2{,}53 & 0 & \times \\ 3{,}29 & 1{,}05 & 3{,}15 & 2{,}61 & 2{,}48 & 0 \end{pmatrix} \end{array}$$

Además, en cada iteración del algoritmo se elegirán una/s serie de distancias de esta última matriz mostrada, necesitando saber cuáles han sido ya elegidas en iteraciones previas. Para ello se emplea una matriz de las mismas dimensiones que almacena ceros y unos indicando si dicha entrada ha sido seleccionada previamente. La creación de esta matriz inicial queda recogida en la función `create_chosen_matrix`, y un ejemplo de esta matriz sería el siguiente.

$$
\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{array} \\
\begin{pmatrix}
0 & \times & \times & \times & \times & \times \\
4{,}15 & 0 & \times & \times & \times & \times \\
3{,}39 & 4{,}13 & 0 & \times & \times & \times \\
5{,}36 & 2{,}8 & 3{,}21 & 0 & \times & \times \\
3{,}41 & 3{,}42 & \boxed{0{,}76} & 2{,}53 & 0 & \times \\
3{,}29 & 1{,}05 & 3{,}15 & 2{,}61 & 2{,}48 & 0
\end{pmatrix}
\end{array}
\qquad
\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{array} \\
\begin{pmatrix}
\times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times \\
0 & 0 & \times & \times & \times & \times \\
0 & 0 & 0 & \times & \times & \times \\
0 & 0 & 1 & 0 & \times & \times \\
0 & 0 & 0 & 0 & 0 & \times
\end{pmatrix}
\end{array}
$$

```
create_chosen_matrix = function(df) {
        n = len(df[, 1])
        empty_matrix = matrix(0, ncol = n, nrow = n)
        chosen = data.frame(empty_matrix)
        for (i in 1:n) {
                for (j in 1:i) {
                        chosen[j, i] = NA
                }
        }
        chosen
}
```

Finalmente, el último elemento que falta para poder dar comienzo al algoritmo es buscar una forma de llevar un control de la estructura de los clústers, mostrado en el siguiente ejemplo como etiquetas en las filas y las columnas de las matrices.

$$
\begin{array}{c} \\ x_1 \\ C_3 \supset C_2 \supset x_2 \\ C_1 \supset x_3 \\ C_3 \supset x_4 \\ C_1 \supset x_5 \\ C_3 \supset C_2 \supset x_6 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} x_1 & C_3 \supset C_2 \supset x_2 & C_1 \supset x_3 & C_3 \supset x_4 & C_1 \supset x_5 & C_3 \supset C_2 \supset x_6 \end{array} \\
\begin{pmatrix}
0 & \times & \times & \times & \times & \times \\
4{,}27 & 0 & \times & \times & \times & \times \\
3{,}4 & 3{,}15 & 0 & \times & \times & \times \\
4{,}27 & \boxed{2{,}71} & 3{,}15 & 0 & \times & \times \\
3{,}4 & 3{,}15 & \boxed{0{,}76} & 3{,}15 & 0 & \times \\
4{,}27 & \boxed{1{,}05} & 3{,}15 & \boxed{2{,}71} & 3{,}15 & 0
\end{pmatrix}
\end{array}
\tag{2.1}
$$

Se puede observar que aunque los clústers se comporten como conjuntos, la estructura de datos más adecuada para almacenar cada una de las etiquetas es un árbol. Como se tiene que trabajar con un conjunto de n árboles, se estará trabajando con un bosque (vector de árboles). Para hacer esto posible se hará uso del paquete `data.tree`. Al ser la matriz simétrica, el bosque de las filas será igual al de las columnas. Para el último ejemplo mostrado, el bosque tendría el aspecto del mostrado a continuación. Obsérvese que al finalizar el algoritmo, todos los árboles del bosque deberán ser el mismo.

$$\texttt{forest} = \left[x_1, \; \begin{array}{c} C_3 \\ x_4 \quad C_2 \\ \quad\; x_2 \quad x_6 \end{array}, \; \begin{array}{c} C_1 \\ x_3 \quad x_5 \end{array}, \; \begin{array}{c} C_3 \\ x_4 \quad C_2 \\ \quad\; x_2 \quad x_6 \end{array}, \; \begin{array}{c} C_1 \\ x_3 \quad x_5 \end{array}, \; \begin{array}{c} C_3 \\ x_4 \quad C_2 \\ \quad\; x_2 \quad x_6 \end{array} \right] \tag{2.2}$$

Una vez se han presentado las estructuras necesarias para realizar el algoritmo, se explicará el funcionamiento de este. El flujo principal queda recogido en la función `fcd_ahc`.

```
fcd_ahc = function(data, criteria, details = FALSE) {
        dist_matrix = create_distance_matrix(data)
        chosen_matrix = create_chosen_matrix(data)
        first_dist_matrix = dist_matrix
        next_cluster = 1
        forest = sapply(colnames(dist_matrix), function(root){Node$new(root)})
        while(sum(chosen_matrix, na.rm = TRUE) !=
                (len(dist_matrix) * (len(dist_matrix) - 1) / 2)){
                min_index = min_ahc(dist_matrix, chosen_matrix)
                chosen_matrix = choose_distance(dist_matrix, chosen_matrix,
                         dist_matrix[min_index[1], min_index[2]])
                new_tree_tag = Node$new(paste0("C", next_cluster))
                new_tree_tag$AddChildNode(forest[[min_index[1]]])
                new_tree_tag$AddChildNode(forest[[min_index[2]]])
                trees_2_update = sapply(Traverse(new_tree_tag,
                filterFun = isLeaf), function(x){
                        as.integer(substring(x$name, 2))
                        })
                for(i in 1:len(forest)){
                        if(i %in% trees_2_update){
                                forest[[i]] = new_tree_tag
                        }
                }
                next_cluster = next_cluster + 1
                criteria_name = switch(
                        criteria, "MIN" = min, "MAX" = max, "AVG" = fcd_mean)
                dist_matrix = update_distance_matrix(first_dist_matrix,
                        dist_matrix, forest, trees_2_update, criteria_name)
                if (details) {
                        cat("\nIteración", next_cluster - 1)
                        cat("============\n")
                        cat("\n\nMatriz de distancias\n")
```

```
                        print(dist_matrix)
                        cat("\n\nMatriz de distancias elegidas\n")
                        print(chosen_matrix)
                        cat("\n\nDendrogramas\n")
                        print(forest)
                }
        }
        cat("\n\nMatriz cofenética\n")
        print(dist_matrix)
        cat("\nDendrograma final\n")
        print(forest[[1]])
        cpcc = calculate_cpcc(first_dist_matrix, dist_matrix)
        cat("\n\nCoeficiente de CPCC: ", cpcc)
        plot(forest[[1]])
        plot(as.dendrogram(forest[[1]]), center = TRUE, yaxt='n')
}
```

A continuación se irá explicando esta función paso a paso de manera que se entienda cómo funciona el algoritmo. En primer lugar se inician las estructuras de datos comentadas anteriormente. Se observa que el instante inicial, el bosque debe ser de la forma $[x_1, x_2, \ldots, x_n]$. Además, en la variable `next_cluster` se almacena el índice del siguiente clúster que se creará.

El cuerpo del algoritmo transcurre en un bucle. El algoritmo finaliza cuando todos los elementos de la triangular inferior de la matriz han sido seleccionados. Como tanto la matriz de distancias, como la de elegidos son matrices $n \times n$ y solo se trabaja con la triangular inferior, una forma sencilla de comprobar si el algoritmo se encuentra en este punto es comprobar si la suma de la triangular inferior de la matriz de elegidos es igual al número de elementos de la triangular inferior de la matriz de distancias. Como las matrices son cuadradas, la primera columna tendrá n elementos, la segunda $n-1$, y así hasta la última que tendrá un único elemento. Haciendo uso de la igualdad

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2},$$

se puede deducir que el algoritmo finaliza cuando la suma de los elementos de la triangular inferior de la matriz de elegidos sea igual a $\frac{n(n-1)}{2}$, pues se ignoran los elementos de la diagonal principal.

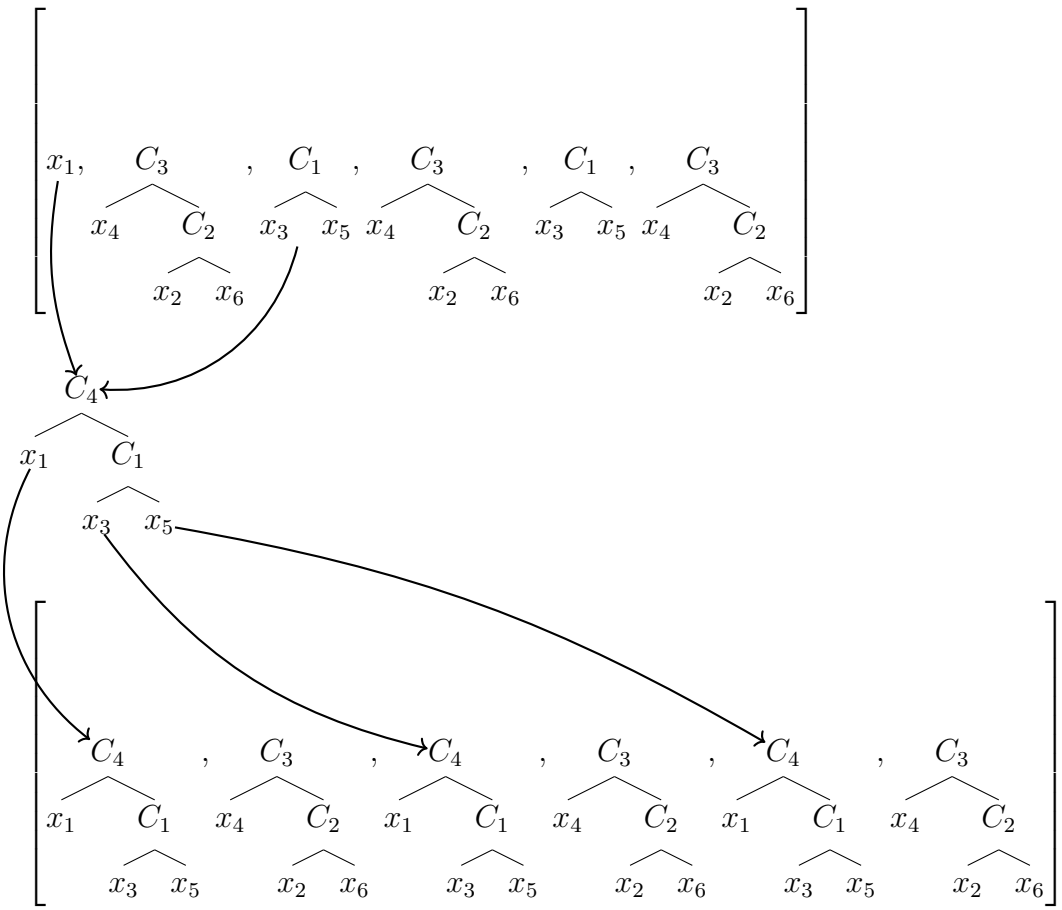
En cada iteración del algoritmo se elige la menor de las distancias que no haya sido elegida previamente y se marca. Si la misma distancia aparece más de una vez, se marcan todas sus apariciones en una misma iteración. Las funciones `min_ahc` y `choose_distance` realizan esta tarea.

```
min_ahc = function(distances, chosen) {
        min_d = Inf
        min_c = c()
        for(i in 1:nrow(distances)) {
                for(j in 1:i) {
                        if (distances[i, j] < min_d
                         && distances[i, j] != 0 && chosen[i, j] == 0) {
                                min_d = distances[i, j]
                                min_c = c(i, j)
                        }
                }
        }
        min_c
}

choose_distance = function(distances, chosen, distance){
        for(i in 1:nrow(distances)) {
                for(j in 1:i) {
                        if (distances[i, j] == distance) {
                                chosen[i, j] = 1
                        }
                }
        }
        chosen
}
```

La primera de las funciones devuelve el índice de fila y columna de la menor de las distancias, y la siguiente, marca en la matriz de elegidos todas las apariciones de esta distancia. Que se haya seleccionado el valor a_{ij} indica que el nuevo clúster C_i, contendrá a las raíces de los árboles A_i y A_j, sustituyéndose los árboles A_i y A_j por este.

Sin embargo, esto sólo sucede así en el caso base de que A_i y A_j son árboles que solo contienen a un único x_i. En caso contrario, se puede observar que deberán ser actualizadas todas las etiquetas de la matriz cuya columna esté en el árbol de la entrada seleccionada (es decir, si se elige a_{35} pero x_2 estaba involucrado, también deberá actualizarse la etiqueta de la segunda columna además de la de la tercera y la quinta). Veamos en el ejemplo de la Ecuación (2.2) qué sucedería si se selecciona la entrada a_{13} (ignorando si esto es correcto). En primer lugar, los árboles A_1 y A_3 se sustituyen por aquel formado por ambos. Sin embargo, también deberá asignarse este nuevo árbol a A_5, pues x_5 está también en este nuevo árbol. En general, para saber qué etiquetas han de actualizarse, si se marca a_{ij}, se crea un nuevo árbol con A_i y A_j, se recorre, y para cada una de los nodos n_k si n_k es hoja y contiene a x_m, el nuevo árbol es asignado a A_m.



Una vez definida la estructura de los clústers en este punto, es momento de actualizar los valores de la matriz de distancias. Se puede observar que los valores marcados nunca deberán cambiar, y que deberán ser actualizadas aquellas entradas cuya etiqueta de fila o columna haya sido actualizada, pues de lo contrario, se tratará de un punto que no ha intervenido en la última fusión de clústers. Esta actualización se realiza en la función `update_distance_matrix`.

```
update_distance_matrix = function(
        initial_dm, actual_dm, forest, new_cluster_points, criteria_name) {
        updated_cols = c(rep(0, times = len(forest)))
        updated_cols[new_cluster_points] = 1

        for (i in 1:len(forest)) {
                if (!updated_cols[i]) {
                        tree_points = sapply(Traverse(forest[[i]],
                                filterFun = isLeaf), function(x){
                                as.integer(substring(x$name, 2))})
                        new_distance = func_criteria(initial_dm,
                                new_cluster_points, tree_points,
```

```
                                criteria_name)

                        for (i in new_cluster_points) {
                                for (j in tree_points) {
                                        minor = min(i, j)
                                        mayor = max(i, j)
                                        actual_dm[mayor, minor] =
                                                new_distance
                                }
                        }
                        updated_cols[tree_points] = 1
                                        }
                                }
        actual_dm
}
```

Esta función calcula las entradas cuyas distancias deberán ser actualizadas, y de entre ellas, va asignando a cada una aquellas cuyas distancias deberán ser iguales. A la hora de calcular las nuevas distancias se puede elegir entre tres criterios:

- `MIN`:
$$\text{dist}(C_i, C_j) = \min_{(x,y)\in C_i \times C_j} \{\text{dist}(x, y)\}$$
- `MAX`:
$$\text{dist}(C_i, C_j) = \max_{(x,y)\in C_i \times C_j} \{\text{dist}(x, y)\}$$
- `AVG`:
$$\text{dist}(C_i, C_j) = \frac{\sum_{x\in C_i} \sum_{y\in C_j} \text{dist}(x, y)}{|C_i \times C_j|}$$

El cálculo según diferentes criterios se realiza en la siguiente función.

```
func_criteria = function(initial_dm, new_cluster_points,
        tree_points, criteria_name) {
        distances = c()
        for (i in new_cluster_points) {
                for (j in tree_points) {
                        minor = min(i, j)
                        mayor = max(i, j)
                        distances = c(distances, initial_dm[mayor, minor])
                }
        }
        criteria_name(distances)
}
```

Adicionalmente, en caso de que el usuario lo desee, se va mostrando la traza del algoritmo para ver paso a paso qué sucede. Cuando acaba el bucle `while` en la variable que almacenaba la matriz de distancias, queda la matriz cofenética, muy útil para calcular el CPCC, que indica cómo de buena ha sido la clusterización. Se calcula mediante las ecuaciones mostradas a continuación y dichos cálculos se realizan en la función `calculate_cpcc`.

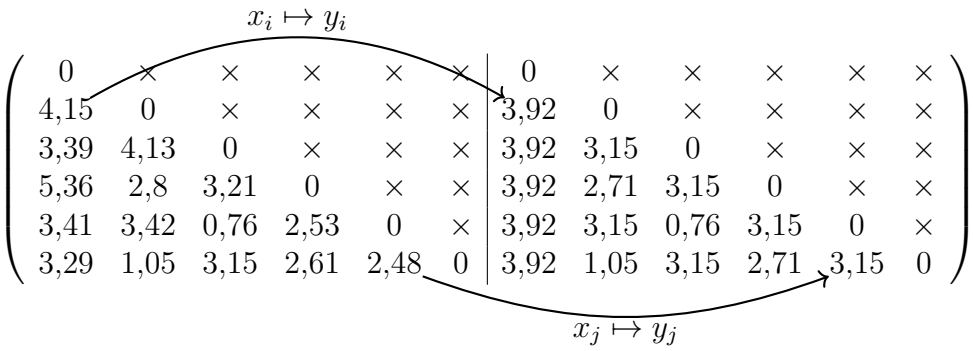
$$CPCC = \frac{S_{xy}}{S_x S_y}$$

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

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

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

Queda por definir quiénes son las variables x e y. Estas representan los valores de la matriz de distancias inicial y final. Cada x_i tiene un y_i asociado, por ejemplo, supongamos que estas son la matriz inicial y final de una ejecución del algoritmo:



$$\left(\begin{array}{cccccc|cccccc}
0 & \times & \times & \times & \times & \times & 0 & \times & \times & \times & \times & \times \\
4{,}15 & 0 & \times & \times & \times & \times & 3{,}92 & 0 & \times & \times & \times & \times \\
3{,}39 & 4{,}13 & 0 & \times & \times & \times & 3{,}92 & 3{,}15 & 0 & \times & \times & \times \\
5{,}36 & 2{,}8 & 3{,}21 & 0 & \times & \times & 3{,}92 & 2{,}71 & 3{,}15 & 0 & \times & \times \\
3{,}41 & 3{,}42 & 0{,}76 & 2{,}53 & 0 & \times & 3{,}92 & 3{,}15 & 0{,}76 & 3{,}15 & 0 & \times \\
3{,}29 & 1{,}05 & 3{,}15 & 2{,}61 & 2{,}48 & 0 & 3{,}92 & 1{,}05 & 3{,}15 & 2{,}71 & 3{,}15 & 0
\end{array}\right)$$

```
calculate_cpcc = function(first_dist_matrix, dist_matrix) {
        x = unlist(first_dist_matrix)
        x = x[!is.na(x) & x != 0]

        y = unlist(dist_matrix)
        y = y[!is.na(y) & y != 0]

        sx = standard_dev(x)
        sy = standard_dev(y)
        sxy = covariance(x, y)

        sxy / (sx * sy)
}
```

Para finalizar, se muestra un gráfico del árbol final (los n árboles acaban igual) y un dendrograma de la clusterización resultante.

```
(sample = read.xlsx("data/datosKMeans.xlsx"))
```

```
##    Velocidad Temperatura
## 1       3.50        4.50
## 2       0.75        3.25
## 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
```

```
fcd_ahc(sample, "MIN")
```

```
##
##
## Matriz cofenética
##          X1        X2       X3        X4       X5       X6        X7       X8
## 1  0.000000        NA       NA        NA       NA       NA        NA       NA
## 2  2.304886 0.0000000       NA        NA       NA       NA        NA       NA
## 3  2.304886 0.3535534 0.000000        NA       NA       NA        NA       NA
## 4  2.304886 1.8027756 1.802776 0.0000000       NA       NA        NA       NA
## 5  0.250000 2.3048861 2.304886 2.3048861 0.000000       NA        NA       NA
## 6  0.250000 2.3048861 2.304886 2.3048861 0.250000 0.000000        NA       NA
## 7  2.304886 1.8027756 1.802776 0.3535534 2.304886 2.304886 0.0000000       NA
## 8  2.304886 0.2500000 0.250000 1.8027756 2.304886 2.304886 1.8027756 0.000000
## 9  0.250000 2.3048861 2.304886 2.3048861 0.250000 0.250000 2.3048861 2.304886
## 10 2.304886 1.8027756 1.802776 0.3535534 2.304886 2.304886 0.3535534 1.802776
## 11 2.304886 1.8027756 1.802776 0.3535534 2.304886 2.304886 0.3535534 1.802776
## 12 0.559017 2.3048861 2.304886 2.3048861 0.250000 0.559017 2.3048861 2.304886
## 13 2.304886 0.3535534 0.500000 1.8027756 2.304886 2.304886 1.8027756 0.250000
## 14 2.304886 0.5590170 0.559017 0.5590170 2.304886 2.304886 0.5590170 0.559017
## 15 2.304886 0.5000000 0.500000 1.8027756 2.304886 2.304886 1.8027756 0.500000
##          X9       X10      X11      X12      X13      X14 X15
## 1        NA        NA       NA       NA       NA       NA  NA
## 2        NA        NA       NA       NA       NA       NA  NA
## 3        NA        NA       NA       NA       NA       NA  NA
## 4        NA        NA       NA       NA       NA       NA  NA
```

```
## 5        NA        NA       NA       NA       NA       NA  NA
## 6        NA        NA       NA       NA       NA       NA  NA
## 7        NA        NA       NA       NA       NA       NA  NA
## 8        NA        NA       NA       NA       NA       NA  NA
## 9  0.000000        NA       NA       NA       NA       NA  NA
## 10 2.304886 0.0000000       NA       NA       NA       NA  NA
## 11 2.304886 0.3535534 0.000000       NA       NA       NA  NA
## 12 0.559017 2.3048861 2.304886 0.000000       NA       NA  NA
## 13 2.304886 1.8027756 1.802776 2.304886 0.000000       NA  NA
## 14 2.304886 0.5590170 0.559017 2.304886 0.559017 0.000000  NA
## 15 2.304886 1.8027756 1.802776 2.304886 0.500000 0.559017   0
##
## Dendrograma final
##                          levelName
## 1  C14
## 2   ¦--C13
## 3   ¦   ¦--X14
## 4   ¦   °--C12
## 5   ¦       ¦--C5
## 6   ¦       ¦   ¦--X11
## 7   ¦       ¦   °--C4
## 8   ¦       ¦       ¦--X7
## 9   ¦       ¦       °--C3
## 10  ¦       ¦           ¦--X10
## 11  ¦       ¦           °--X4
## 12  ¦       °--C9
## 13  ¦           ¦--X15
## 14  ¦           °--C8
## 15  ¦               ¦--X8
## 16  ¦               °--C7
## 17  ¦                   ¦--C6
## 18  ¦                   ¦   ¦--X13
## 19  ¦                   ¦   °--X3
## 20  ¦                   °--X2
## 21  °--C11
## 22      ¦--X5
## 23      °--C10
## 24          ¦--X12
## 25          °--C2
## 26              ¦--X9
## 27              °--C1
## 28                  ¦--X6
## 29                  °--X1
##
##
## Coeficiente de CPCC:  0.8701407
```

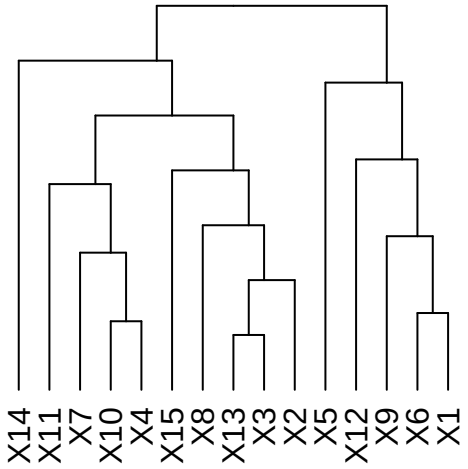


Figura 2.1: Clusterización con `MIN`

```
fcd_ahc(sample, "MAX")
```

```
##
##
## Matriz cofenética
##           X1        X2        X3       X4       X5        X6       X7       X8
## 1  0.0000000        NA        NA       NA       NA        NA       NA       NA
## 2  4.5961941 0.0000000        NA       NA       NA        NA       NA       NA
## 3  4.5961941 0.7905694 0.0000000       NA       NA        NA       NA       NA
## 4  4.5961941 2.8504386 2.8504386 0.000000       NA        NA       NA       NA
## 5  1.0606602 4.5961941 4.5961941 4.596194 0.000000        NA       NA       NA
## 6  0.2500000 4.5961941 4.5961941 4.596194 1.060660 0.0000000       NA       NA
## 7  4.5961941 2.8504386 2.8504386 0.500000 4.596194 4.5961941 0.000000       NA
## 8  4.5961941 0.5590170 0.7905694 2.850439 4.596194 4.5961941 2.850439 0.000000
## 9  0.3535534 4.5961941 4.5961941 4.596194 1.060660 0.3535534 4.596194 4.596194
## 10 4.5961941 3.4003676 3.4003676 3.400368 4.596194 4.5961941 3.400368 3.400368
## 11 4.5961941 3.4003676 3.4003676 3.400368 4.596194 4.5961941 3.400368 3.400368
## 12 0.9013878 4.5961941 4.5961941 4.596194 1.060660 0.9013878 4.596194 4.596194
## 13 4.5961941 2.8504386 2.8504386 2.573908 4.596194 4.5961941 2.573908 2.850439
## 14 4.5961941 3.4003676 3.4003676 3.400368 4.596194 4.5961941 3.400368 3.400368
## 15 4.5961941 2.8504386 2.8504386 2.474874 4.596194 4.5961941 2.474874 2.850439
##           X9       X10      X11      X12      X13      X14 X15
## 1         NA        NA       NA       NA       NA       NA  NA
```

```
## 2         NA        NA       NA       NA       NA       NA  NA
## 3         NA        NA       NA       NA       NA       NA  NA
## 4         NA        NA       NA       NA       NA       NA  NA
## 5         NA        NA       NA       NA       NA       NA  NA
## 6         NA        NA       NA       NA       NA       NA  NA
## 7         NA        NA       NA       NA       NA       NA  NA
## 8         NA        NA       NA       NA       NA       NA  NA
## 9  0.0000000        NA       NA       NA       NA       NA  NA
## 10 4.5961941 0.0000000       NA       NA       NA       NA  NA
## 11 4.5961941 0.7071068 0.000000       NA       NA       NA  NA
## 12 0.9013878 4.5961941 4.596194 0.000000       NA       NA  NA
## 13 4.5961941 3.4003676 3.400368 4.596194 0.000000       NA  NA
## 14 4.5961941 1.2500000 1.250000 4.596194 3.400368 0.000000  NA
## 15 4.5961941 3.4003676 3.400368 4.596194 2.573908 3.400368   0
##
## Dendrograma final
##                    levelName
## 1  C14
## 2   ¦--C13
## 3   ¦   ¦--C9
## 4   ¦   ¦   ¦--X14
## 5   ¦   ¦   °--C5
## 6   ¦   ¦       ¦--X11
## 7   ¦   ¦       °--X10
## 8   ¦   °--C12
## 9   ¦       ¦--C11
## 10  ¦       ¦   ¦--X13
## 11  ¦       ¦   °--C10
## 12  ¦       ¦       ¦--X15
## 13  ¦       ¦       °--C3
## 14  ¦       ¦           ¦--X7
## 15  ¦       ¦           °--X4
## 16  ¦       °--C6
## 17  ¦           ¦--X3
## 18  ¦           °--C4
## 19  ¦               ¦--X8
## 20  ¦               °--X2
## 21  °--C8
## 22      ¦--X5
## 23      °--C7
## 24          ¦--X12
## 25          °--C2
## 26              ¦--X9
## 27              °--C1
## 28                  ¦--X6
## 29                  °--X1
##
```

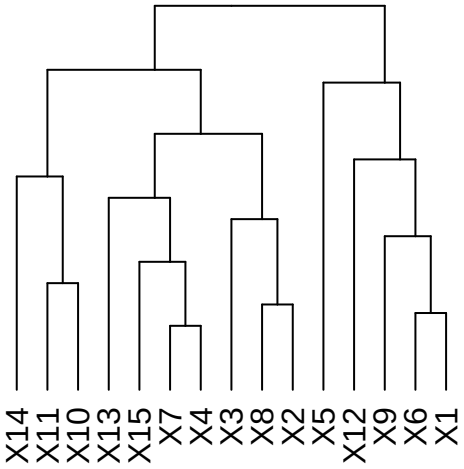


Figura 2.2: Clusterización con `MAX`

```
##
## Coeficiente de CPCC:  0.8157924
```

```
fcd_ahc(sample, "AVG")

##
##
## Matriz cofenética
##           X1        X2        X3        X4        X5        X6        X7
## 1  0.0000000        NA        NA        NA        NA        NA        NA
## 2  3.6662879 0.0000000        NA        NA        NA        NA        NA
## 3  3.6662879 0.4110307 0.0000000        NA        NA        NA        NA
## 4  3.6662879 2.6284228 2.6284228 0.0000000        NA        NA        NA
## 5  0.7297887 3.6662879 3.6662879 3.6662879 0.0000000        NA        NA
## 6  0.2500000 3.6662879 3.6662879 3.6662879 0.7297887 0.0000000        NA
## 7  3.6662879 2.6284228 2.6284228 0.4267767 3.6662879 3.6662879 0.0000000
## 8  3.6662879 0.5590170 0.4110307 2.6284228 3.6662879 3.6662879 2.6284228
## 9  0.3017767 3.6662879 3.6662879 3.6662879 0.7297887 0.3017767 3.6662879
## 10 3.6662879 2.6284228 2.6284228 0.3535534 3.6662879 3.6662879 0.4267767
## 11 3.6662879 2.6284228 2.6284228 0.6170765 3.6662879 3.6662879 0.6170765
## 12 0.7225039 3.6662879 3.6662879 3.6662879 0.7297887 0.7225039 3.6662879
## 13 3.6662879 0.4110307 0.5000000 2.6284228 3.6662879 3.6662879 2.6284228
```

```
## 14 3.6662879 2.6284228 2.6284228 0.8173555 3.6662879 3.6662879 0.8173555
## 15 3.6662879 0.7066960 0.7066960 2.6284228 3.6662879 3.6662879 2.6284228
##           X8        X9       X10       X11      X12      X13      X14 X15
## 1         NA        NA        NA        NA       NA       NA       NA  NA
## 2         NA        NA        NA        NA       NA       NA       NA  NA
## 3         NA        NA        NA        NA       NA       NA       NA  NA
## 4         NA        NA        NA        NA       NA       NA       NA  NA
## 5         NA        NA        NA        NA       NA       NA       NA  NA
## 6         NA        NA        NA        NA       NA       NA       NA  NA
## 7         NA        NA        NA        NA       NA       NA       NA  NA
## 8  0.0000000        NA        NA        NA       NA       NA       NA  NA
## 9  3.6662879 0.0000000        NA        NA       NA       NA       NA  NA
## 10 2.6284228 3.6662879 0.0000000        NA       NA       NA       NA  NA
## 11 2.6284228 3.6662879 0.6170765 0.0000000       NA       NA       NA  NA
## 12 3.6662879 0.7225039 3.6662879 3.6662879 0.000000       NA       NA  NA
## 13 0.4110307 3.6662879 2.6284228 2.6284228 3.666288 0.000000       NA  NA
## 14 2.6284228 3.6662879 0.8173555 0.8173555 3.666288 2.628423 0.000000  NA
## 15 0.7066960 3.6662879 2.6284228 2.6284228 3.666288 0.706696 2.628423   0
##
## Dendrograma final
##                      levelName
## 1  C14
## 2   ¦--C13
## 3   ¦   ¦--C12
## 4   ¦   ¦   ¦--X14
## 5   ¦   ¦   °--C8
## 6   ¦   ¦       ¦--X11
## 7   ¦   ¦       °--C4
## 8   ¦   ¦           ¦--X7
## 9   ¦   ¦           °--C3
## 10  ¦   ¦               ¦--X10
## 11  ¦   ¦               °--X4
## 12  ¦   °--C9
## 13  ¦       ¦--X15
## 14  ¦       °--C7
## 15  ¦           ¦--C5
## 16  ¦           ¦   ¦--X13
## 17  ¦           ¦   °--X3
## 18  ¦           °--C6
## 19  ¦               ¦--X8
## 20  ¦               °--X2
## 21  °--C11
## 22      ¦--X5
## 23      °--C10
## 24          ¦--X12
## 25          °--C2
## 26              ¦--X9
```

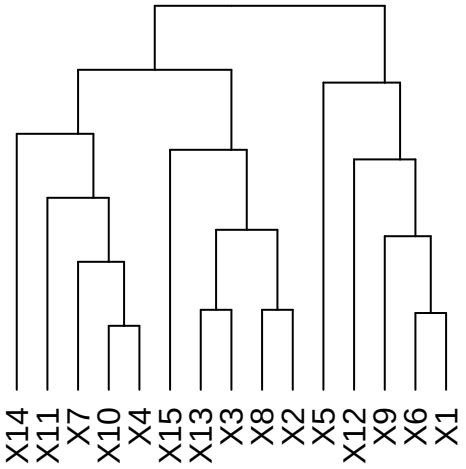


Figura 2.3: Clusterización con `AVG`

```
## 27              °--C1
## 28                 ¦--X6
## 29                 °--X1
##
##
## Coeficiente de CPCC:  0.9423605
```

2.2. Clasificación supervisada

2.2.1. Árboles de decisión

Ejercicio 2.2.1. *El tercer conjunto de datos, que se empleará para realizar el análisis de clasificación supervisada utilizando árboles de decisión, estará formado por el siguiente conjunto de 10 sucesos constituidos por los valores de cuatro características de vehículos: 1. {B, 4, 5, Coche}; 2. {A, 2, 2, Moto}; 3. {N, 2, 1, Bicicleta}; 4. {B, 6, 4, Camión}; 5. {B, 4, 6, Coche}; 6. {B, 4, 4, Coche}; 7.{N, 2, 2, Bicicleta}; 8. {B, 2, 1, Moto}; 9. {B, 6, 2, Camión}; 10. {N, 2, 1, Bicicleta}, donde las características de cada suceso son: {TipoCarnet, NúmeroRuedas, NúmeroPasajeros, TipoVehículo}. Se debe clasificar el tipo de vehículo en función del resto de características. TipoCarnet, es el tipo de carnet necesario para conducir el vehículo.*

Para este conjunto de datos se va a plantear un estudio de clasificación supervisada utilizando como técnica los árboles de decisión. Se buscará definir, a partir de la muestra, un modelo de clasificación que permita obtener el valor de una característica a partir del valor del resto de características.

Como se necesita una muestra, lo primero que se va a realizar es la lectura del fichero Excel (`.xlsx`) por medio de la función `read.xlsx` del paquete `openxlsx`. Al igual que en el resto de ejercicios, se deberá pasar la ruta del fichero. En él se encuentra contenido el conjunto de daos propuesto para el ejercicio.

```
(sample = read.xlsx("data/vehiculos.xlsx"))
```

```
##    tCarnet nRuedas nPasajeros tVehículo
## 1        B       4          5     Coche
## 2        A       2          2      Moto
## 3        N       2          1 Bicicleta
## 4        B       6          4    Camión
## 5        B       4          6     Coche
## 6        B       4          4     Coche
## 7        N       2          2 Bicicleta
## 8        B       2          1      Moto
## 9        B       6          2    Camión
## 10       N       2          1 Bicicleta
```

Una vez tenemos los datos leídos hay que plantear el problema. En este caso, se pretende sacar el tipo de vehículo (`tVehiculo`) a partir del resto de características (`tCarnet`, `nRuedas` y `nPasajeros`). Al primero le llamaremos a partir de ahora clasificador. Se va a hacer uso del algoritmo de Hunt. Este algoritmo de clasificación supervisada basada en árboles de decisión se compone de dos pasos fundamentales:

1. **Selección de nodo**: Para el primer paso se elige una característica, que compondrá la raíz del árbol. En el resto de pasos, esta selección formará los nodos intermedios del mismo. La característica que forme el nodo nunca podrá ser el clasificador (`tVehiculo` en este caso). El objetivo de este árbol, una vez formado, es ver en cada nodo una característica. De cada nodo saldrán hijos cuya diferencia será el valor de la característica del nodo padre. Los hijos contendrán, o bien un valor del clasificador (lo que permitirá clasificar directamente un suceso de la muestra), o bien un subárbol del que dependerá la clasificación del resto de la muestra (habrá que recorrerlo para poder clasificar un suceso de la muestra).

   Con el fin de poder optimizar estos árboles, se ha de encontrar la característica que mejor clasifique en base a la muestra con la que se está trabajando. Esto se va a realizar con la medida de ganancia de información (Δ_I). Esta medida mide de una división, la diferencia de impureza entre el nodo padre y los nodos hijos una vez realizada esa división. A mayor ganancia de información, mejor división se habrá realizado, por lo que

el objetivo será implementar a la clasificación la división con mayor Δ_I. La ganancia de información se calcula de la siguiente forma:

$$\Delta_i = I_{\text{padre}} - \sum_{j=1}^{k} \frac{N(n_j)}{N} \cdot I(n_j)$$

- I_{padre}: Impureza del nodo padre
- k: Número total de hijos (en esta implementación siempre será 2)
- $N(n_j)$: Número de sucesos de la muestra asociados al nodo hijo j
- N: Número total de sucesos de la muestra actual
- $I(n_j)$: Impureza del nodo hijo j

Como se puede ver, para calcular la ganancia de información, se necesita la impureza I del nodo padre y de sus hijos. Esta puede ser calculada de diferentes formas en función del método que se utilice. Todas ellas utilizan las frecuencias relativas (f_i) de las clases de equivalencia c del clasificador. Se distinguen tres:

- **Entropía**
$$I = -\sum_{i=1}^{c} f_i \log_c(f_i)$$
- **Error**
$$I = 1 - \text{máx}\{f_i\}_{i=1}^{c}$$
- **Gini**
$$I = 1 - \sum_{i=1}^{c} f_i^2$$

2. **Clasificación de los sucesos:** Una vez se ha elegido la característica que va a conformar el nodo, se clasifica. Si se pueden clasificar completamente los sucesos de la muestra (que en cada hijo solo haya un valor del clasificador posible) se trata de un nodo final y termina la clasificación. Si no lo permite, se trata de un nodo interno, por lo que hay que volver al primer paso.

   Una observación a tener en cuenta es que un nodo ha clasificado completamente cuando la impureza del padre es igual a la ganancia de información ($\Delta_I = I_{\text{padre}}$), ya que significa que las impurezas de los hijos son 0. Cada vez que se clasifique con un criterio determinado, todas las filas de la muestra que contengan ese o esos valores en la característica del nodo se deben eliminar (ya que se considera que han sido clasificadas).

   Puede darse el caso en el que el algoritmo no pueda seguir clasificando porque la ganancia de información sea 0 en todos los escenarios de clasificación posibles. En ese caso, el algoritmo finalizará habiendo clasificado todo lo anterior pero dejando el nodo final abierto, es decir, sin una clasificación concreta.

En esta implementación se utilizarán árboles binarios (cada nodo con 2 hijos) tal y como se ha mencionado previamente. Esto condiciona la forma de actuar en el algoritmo, así que la metodología a seguir se resume en los siguientes pasos:

1. **Selección del mejor nodo**: Se realizará la clasificación en base a
2. **Unión de la clasificación actual al resto de clasificación**:
3. **Recorte de la muestra**:

Estos paso se realizarán hasta que el algoritmo finalice. Este finalizará atendiendo a las siguientes condiciones:

Con esta base teórica se procede a explicar la implementación del algoritmo en R. En primer lugar...

2.2.2. Regresión lineal

Ejercicio 2.2.2. *El cuarto conjunto de datos, que se empleará para realizar el análisis de clasificación supervisada utilizando regresión, estará formado por los siguientes 4 subconjuntos de datos: 1.* $\{10, 8.04; 8, 6.95; 13, 7.58; 9, 8.81; 11, 8.33; 14, 9.96; 6, 7.24; 4, 4.26; 12, 10.84; 7, 4.82; 5, 5.68\}$*; 2.* $\{10, 9.14; 8, 8.14; 13, 8.74; 9, 8.77; 11, 9.26; 14, 8.1; 6, 6.13; 4, 3.1; 12, 9.13; 7, 7.26; 5, 4.74\}$*; 3.* $\{10, 7.46; 8, 6.77; 13, 12.74; 9, 7.11; 11, 7.81; 14, 8.84; 6, 6.08; 4, 5.39; 12, 8.15; 7, 6.42; 5, 5.73\}$*; 4.* $\{8, 6.58; 8, 5.76; 8, 7.71; 8, 8.84; 8, 8.47; 8, 7.04; 8, 5.25; 19, 12.5; 8, 5.56; 8, 7.91; 8, 6.89\}$*. Se deben calcular las rectas de regresión de los cuatro subconjuntos y sus parámetros de ajuste.*

Para la resolución de este ejercicio, se hará uso de la recta de regresión lineal de la forma $\hat{y} = a + bx$ donde los parámetros a y b obedecen a las siguiente ecuaciones.

$$\begin{aligned} b &= \frac{S_{xy}}{S_x^2} \\ a &= \bar{y} - b\bar{x} \end{aligned} \tag{2.3}$$

Para el cálculo de estos parámetros se necesita hacer uso de funciones ya implementadas en la práctica anterior como la media aritmética, la desviación típica y la varianza, se recuerda el código a continuación.

```
fcd_mean = function(list) {
        add = 0
        for (i in 1:len(list)) {
                add = add + list[i]
        }
        add / len(list)
}

standard_dev = function(list) {
```

```
        mean = fcd_mean(list)
        n = len(list)
        add = 0
        for (i in 1:n) {
                add = add + ((list[i] - mean)^2)
        }
        sqrt(add/n)
}

variance = function(list) {
        dev = standard_dev(list)
        var = dev^2
        var
}
```

El siguiente cálculo será hallar el valor de la covarianza de la siguiente manera

$$S_{xy} = \frac{\sum_{i=1}^{n} x_i y_i}{n} - \bar{x}\bar{y}, \tag{2.4}$$

y quedará reflejado en la función `covariance` que recibe dos vectores que representan los datos y sus valores correspondientes:

$$\mathbf{X} = (x_1, x_2, \ldots, x_n)$$
$$\mathbf{Y} = (y_1, y_2, \ldots, y_n)$$

La función realiza el cálculo de la Ecuación (2.4) usando `%*%`, que en este caso realiza el producto escalar de $\mathbf{X}$ e $\mathbf{Y}$. También se verifica que estos vectores compartan dimensiones, pues de lo contrario no se podrá calcular.

```
covariance = function(x, y) {
        if (len(x) != len(y)) {
                stop("X e Y deben tener la misma dimensión")
        }
        else {
                sum = x %*% y
                (sum/len(x))-(fcd_mean(x)*fcd_mean(y))
        }
}
```

Finalmente, la función `regression_line` calcula los parámetros a y b mediante la Ecuación (2.3). Devuelve el resultado de forma matricial para más tarde trabajar de forma más cómoda.

$$\mathbf{P} = \begin{pmatrix} a \\ b \end{pmatrix}$$

```
regression_line = function(x, y) {
        b = covariance(x, y) / variance(x)
        a = fcd_mean(y) - b * fcd_mean(x)
        matrix(c(a, b), ncol = 1)
}
```

Una vez se tienen calculados los parámetros de la recta de regresión lineal, es momento de evaluar cómo de buena es la aproximación. Para ello se emplea la métrica R^2. Su valor depende de otros dos, SSR y SSY, que obedecen a las siguientes ecuaciones.

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

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

Estos cálculos quedan recogidos en las funciones `ssr` y `ssy`. La primera de ellas calcula el valor requerido de forma matricial, recibiendo $\mathbf{P}$, $\mathbf{X}$, e $\mathbf{Y}$ como parámetros, y realiza las siguientes operaciones, donde el cuadrado se realiza elemento a elemento, en vez de forma matricial. En este caso, `%*%` funciona como el producto de matrices usual.

$$\hat{\mathbf{Y}} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

$$SSR = \sum_{i=1}^{n} (\hat{\mathbf{Y}}_i - \bar{\mathbf{Y}}_i)^2$$

De la manera similar, se calcula el SSY como $\sum_{i=1}^{n} (\mathbf{Y}_i - \bar{\mathbf{Y}}_i)^2$. Además se conoce que la relación entre estos dos parámetros y R^2 es la siguiente.

$$R^2 = \frac{SSR}{SSY}$$

```
ssr = function(p, x, y) {
        X = matrix(c(rep(1, times = len(x)), x), ncol = 2)
        y_hat = X %*% p
        sum((y_hat - rep(fcd_mean(y)))^2)
}

ssy = function(y) {
        sum((y - rep(fcd_mean(y)))^2)
}

r2 = function(sr, sy) {
```

```
        sr/sy
}
```

Finalmente se encapsula todo en una única función llamada `fcd_regression` que muestra por consola la ecuación de $\hat{y}$ y el valor de R^2, además de una gráfica con la recta y los puntos.

```
fcd_regression = function(sample) {
        X = sample[, 1]
        Y = sample[, 2]

        param = regression_line(X, Y)
        r = r2(ssr(param, X, Y), ssy(Y))

        a = param[1, 1]
        b = param[2, 1]

        print(sprintf("y = %.3fx + %.3f", b, a))
        print(sprintf("R2 = %.3f", r))

        plot(X, Y, col = "blue", main = "Recta de regresión",
                xlab = "Eje X", ylab = "Eje Y")
        abline(a=a, b=b, col="red")
}
```

De la siguiente forma, se prueba para los casos del profesor. Se observa que las aproximaciones no son buenas, pues los valores de R^2 no se aproximan a 1. En la Figura 2.4 se debe a que los puntos se encuentran de manera un tanto dispersa, dando a entender que no existe relación entre las variables. En la Figura 2.5 se ve claramente que la relación es cuadrática, por lo que es inadecuado utilizar una recta para aproximar esos puntos. En la Figura 2.6 el problema viene dado por un outlier, y en la Figura 2.7 se dan diferentes valores a y para un mismo x lo que hace que la pendiente de esta recta tienda a infinito.

```
sample1 = read.xlsx("../Memoria/data/conj1.xlsx", colNames=FALSE)
fcd_regression(sample1)

## [1] "y = 0.500x + 3.000"
## [1] "R2 = 0.667"
```

```
sample2 = read.xlsx("../Memoria/data/conj2.xlsx", colNames=FALSE)
fcd_regression(sample2)

## [1] "y = 0.500x + 3.001"
## [1] "R2 = 0.666"
```

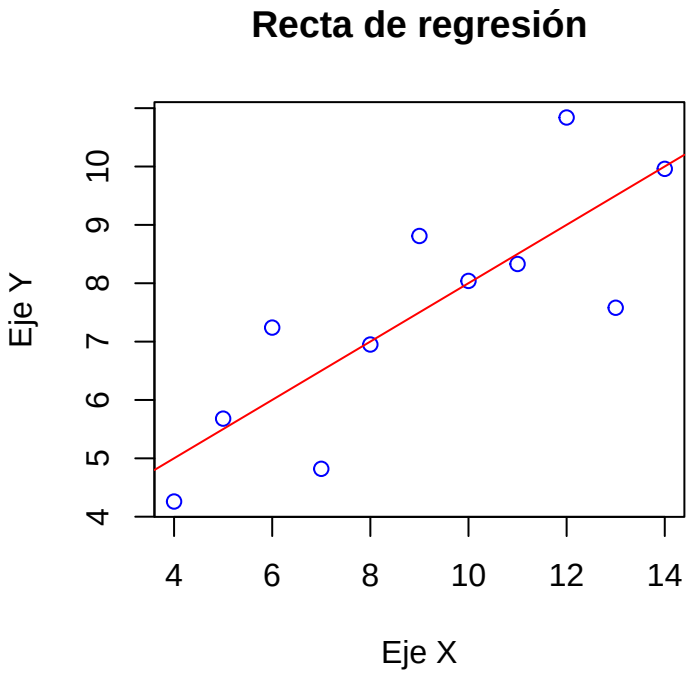


Figura 2.4: Datos no correlacionados

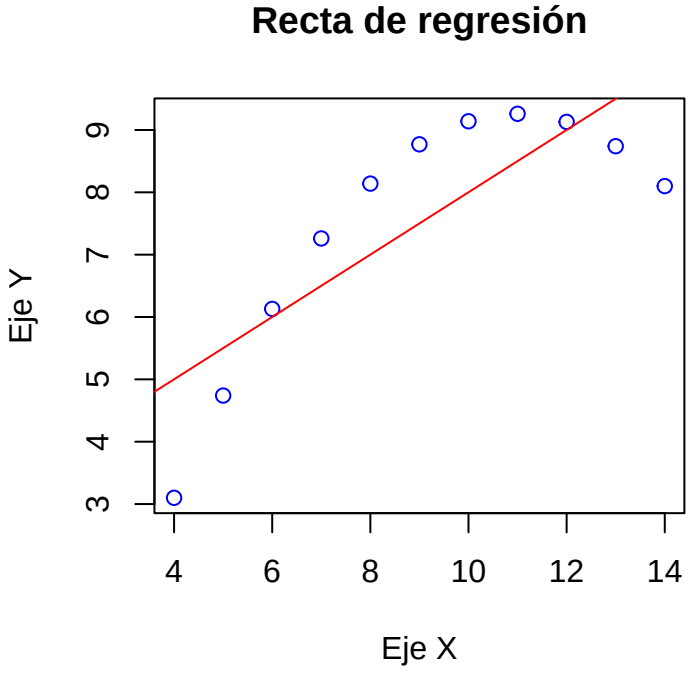


Figura 2.5: Relación cuadrática

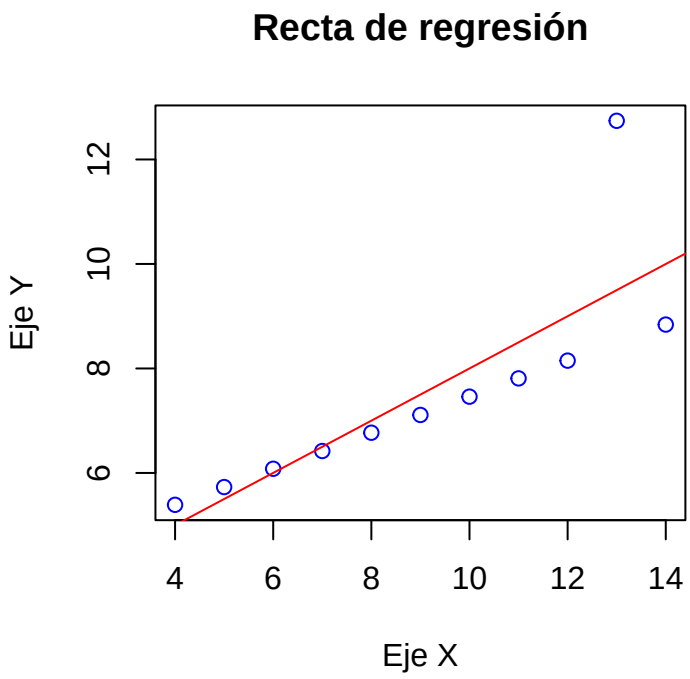


Figura 2.6: Relación lineal con outlier

```
sample3 = read.xlsx("../Memoria/data/conj3.xlsx", colNames=FALSE)
fcd_regression(sample3)

## [1] "y = 0.500x + 3.002"
## [1] "R2 = 0.666"
```

```
sample4 = read.xlsx("../Memoria/data/conj4.xlsx", colNames=FALSE)
fcd_regression(sample4)

## [1] "y = NaNx + NaN"
## [1] "R2 = NaN"

## Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): 'a' y 'b'
deben ser finitos
```

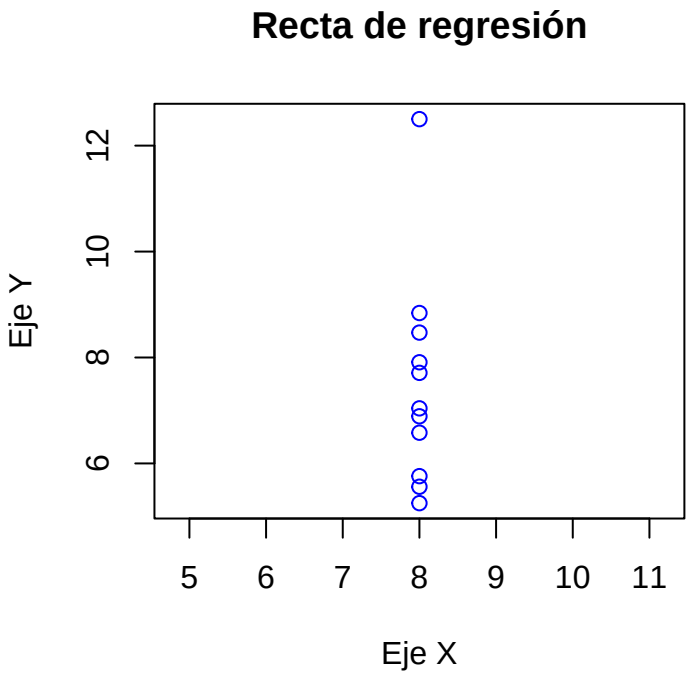


Figura 2.7: Recta con pendiente infinita