K-means

K-means

Sitio: eGela 2017-18 UPV/EHU
Curso: Exploración y análisis de datos
Libro: K-means
Imprimido por: JESUS MARIA YURRAMENDI MENDIZABAL
Día: miércoles, 27 de septiembre de 2017, 08:17

Tabla de contenidos

1 Objetivo

K-means es uno de los algoritmos más importantes y más usados en el análisis de conglomerados (*clustering*). El objetivo es encontrar *k* elementos referidos a un conjunto de datos que en cierto sentido matemático son los que mejor representan dicho conjunto. Estos *k* elementos se conocen como centros de cluster, centroides, prototipos, etc.

Un objeto del conjunto de datos es asociado al centro de clase más cercano, y de esta manera se genera un *partición* del conjunto.

Una *partición* del conjunto de objetos $\boldsymbol{\Omega}$, es, formalmente, una familia de subconjuntos no vacíos del conjunto total que son incompatibles entre sí (cualesquiera dos subconjuntos cualesquiera de la familia tienen intersección vacía), y cuya unión es el conjunto total de objetos.

- $C_1, C_2, \ldots, C_k \subset \boldsymbol{\Omega}$
- $C_i \neq \emptyset \quad i=1,2,\ldots,k$
- $C_i \cap C_j = \emptyset, \; i,j = 1,2,\ldots,k \; i \neq j$
- $\bigcup_{i=1}^{k} C_i = \boldsymbol{\Omega}$

Desde un punto de vista matemático se trata de encontrar los *k* centros de clase que minimizan la *varianza intraclase* (*within*), es decir, la suma de distancias cuadradas desde cada punto a su centro de cluster asociado.

El problema es computacionalmente complejo (NP-hard), pero hay algoritmos heurísticos eficientes que se usan normalmente, y que convergen rápidamente a un *óptimo local*.

2 Algoritmo K-means

1. Fijar el número de clases o clústeres, *k*.
2. Elegir *k* semillas o centros iniciales en el espacio de representación de los objetos para realizar una clasificación con *k* clases o clústeres.
3. Fase de asignación. Asignar cada objeto al centro más cercano según la distancia previamente elegida. Se realiza una clasificación con *k* clases.
4. Fase de actualización. Se eligen nuevos centros en cada una de las *k* clases. Habitualmente suelen ser los centros de gravedad (medias) de las clases.
5. Si alguno de los *k* centros se han alterado respecto a los anteriores, ir al 3. paso, si no parar y dar dicha clasificación como solución del problema.

Comentarios a los distintos pasos:

1. No hay un procedimiento determinado para fijar *k*. Hay que hacer pruebas con distintas *k* para fijarlo. Una visualización del espacio de representación de los objetos pudiera servir de ayuda en dicho proceso de pruebas.
2. La elección de las *k* semillas puede hacerse de forma variada: al azar entre los objetos, al azar en el espacio de representación, sistemáticamente buscando previamente los objetos más densos, es decir, aquellos que más objetos tienen en un determinado entorno, centros de las clases de alguna otra clasificación predeterminada, arbitrariamente según cierta información externa del usuario, ... Pudiera pensarse, incluso, en fijar más de una semilla por cada clase.
3. Hay que definir la distancia empleada en el proceso. Habitualmente suele ser la *distancia euclideana cuadrática*. Por otra parte, hay versiones del algoritmo que actualizan los centros de gravedad después de cada asignación, lo cual hace depender el proceso de clasificación al orden de los objetos.
4. El concepto de semilla puede ser más amplio: un subconjunto de cada clase, una recta representante de la clase, un plano, ...
5. Se puede detener el proceso de mejora antes de cumplirse la condición, bien sea porque se rebasa un predeterminado número máximo de iteraciones, bien porque se rebasa un predeterminado tiempo máximo, bien porque la mejora del criterio es una cantidad insignificante.

El algoritmo K-means funciona bien cuando existen grupos de objetos cuyas representaciones geométricas se corresponden con elipsoides.

En la introducción se ha señalado que la calidad de un resultado es alta cuando la semejanza entre los objetos dentro de una misma clase (intra-cluster, ***W**ithin*) es alta (homogeneidad),es decir, cuando la no-homogeneidad (heterogeneidad) es baja

$$W_0 = \sum_{i=1}^{k} \sum_{a \in C_i} \sum_{b \in C_i} d(a,b)$$

y la semejanza entre los objetos de clases distintas (inter-cluster, ***B**etween*) es baja, es decir, cuando la heterogeneidad es alta

$$B_0 = \sum_{i=1}^{k} \sum_{j / = i} \sum_{a \in C_i} \sum_{b \in C_j} d(a,b)$$

Así, la homogeneidad/heterogeneidad total de Ω

$$T_0 = \sum_{a \in \Omega} \sum_{b \in \Omega} d(a,b)$$

se puede descomponer en esos dos términos:

$$\boldsymbol{T_o = B_o + W_o}$$

Puesto que $\boldsymbol{T_0}$ es fijo, si $\boldsymbol{B_0}$ es alta entonces $\boldsymbol{W_0}$ es baja, y viceversa.

Cuando la *distancia* es *la euclideana cuadrática* estos tres términos pueden expresarse mediante la media global y las medias locales de cada clase, y los términos cuadráticos se corresponden con la varianza total y las varianzas locales.

$$T = \sum_{h=1}^{n} \sum_{j} (x_{hj} - \bar{x}_j)^2$$

$$B = \sum_{i=1}^{k} \sum_{j} (\bar{x}_j^{(i} - \bar{x}_j)^2$$

$$W = \sum_{i=1}^{k} \sum_{h=1}^{n_i} \sum_{j} (x_{hj}^{(i} - \bar{x}_j^{(i)})^2 = \sum_{i=1}^{k} W^{(i}$$

$$\boldsymbol{T = B + W}$$

Las cantidades $\boldsymbol{T}$ y $\boldsymbol{T_o}$ son proporcionales, así como $\boldsymbol{B}$ y $\boldsymbol{B_o}$, y $\boldsymbol{W}$ y $\boldsymbol{W_0}$.

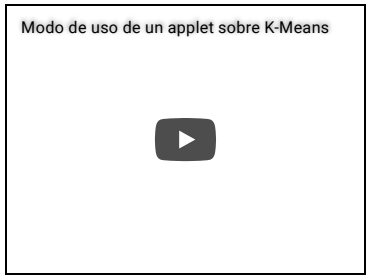
El algoritmo k-means procede de tal forma que minimiza a cada paso la cantidad $\boldsymbol{W}$ (se ponen los centros, y los objetos se asignan al más cercano), por lo que alcanza, en un número suficiente de pasos, un *mínimo local*.

Para una información más detallada de este proceso de minimización, se puede consultar el siguiente fichero.

3 Applet 'K-means Interactive demo'

Applet 'K-means Interactive demo'

3.1 Modo de uso del applet



Modo de uso

- Elige cuántos datos y clases quieres, y luego haz clic en el botón 'Initialize' para generar los puntos en posiciones aleatorias, o bien,
- Inserta manualmente los puntos y los centros iniciales o semillas utilizando los botones derecho e izquierdo del ratón. También puedes eliminar los puntos y las clases haciendo clic sobre ellos.
- Mueve los datos y las semillas como quieras, haciendo clic sobre ellos y arrastrándolos al lugar elegido.
- Elige en el menú de la derecha el tipo de distancia que el algoritmo debe usar.
- Haz clic en el botón 'Start' para empezar la simulación. Se hace una asignación de los puntos a las semillas.
- Haz clic en el botón de la derecha de 'Start', que muestra el número actual de pasos, para el siguiente paso del algoritmo. Se recalculan los centros de las clases, y se reasignan los puntos a dichos centros. Si haces clic sobre el botón 'Run' el proceso va hasta el final de la simulación.
- Utiliza el botón 'Reset' para volver a la configuración inicial de los puntos y las semillas. Ahora puedes mover los puntos y las semillas existentes, o generar otros nuevos y empezar luego otra simulación
- Cuando actives 'Show History' se muestran todos los pasos realizados por los centros de clases.

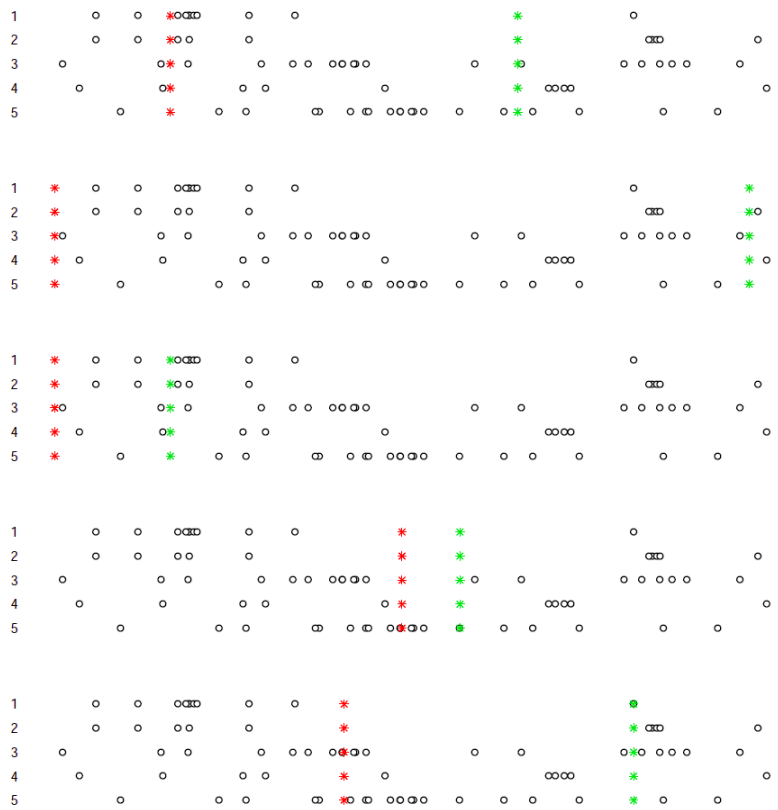
3.2 Ejercicios

3.2.1 Una variable y dos clases

En los siguientes ejercicios se plantean 5 conjuntos de puntos, y para cada uno de ellos se plantean 5 posibles inicializaciones del algoritmo K−means:Observa la disposición de los puntos Define visualmente dos clases

1. Observa los centros iniciales o semillas definidos para las dos clases
2. Trata de reproducir el algoritmo mentalmente, y pronostica el resultado
3. Usa el applet K−means, y dispón los puntos y semillas adecuadamente
4. Compara tus pronósticos con el comportamiento del algoritmo K−means
5. Haz diversas comparaciones con las distintas distancias ofrecidas

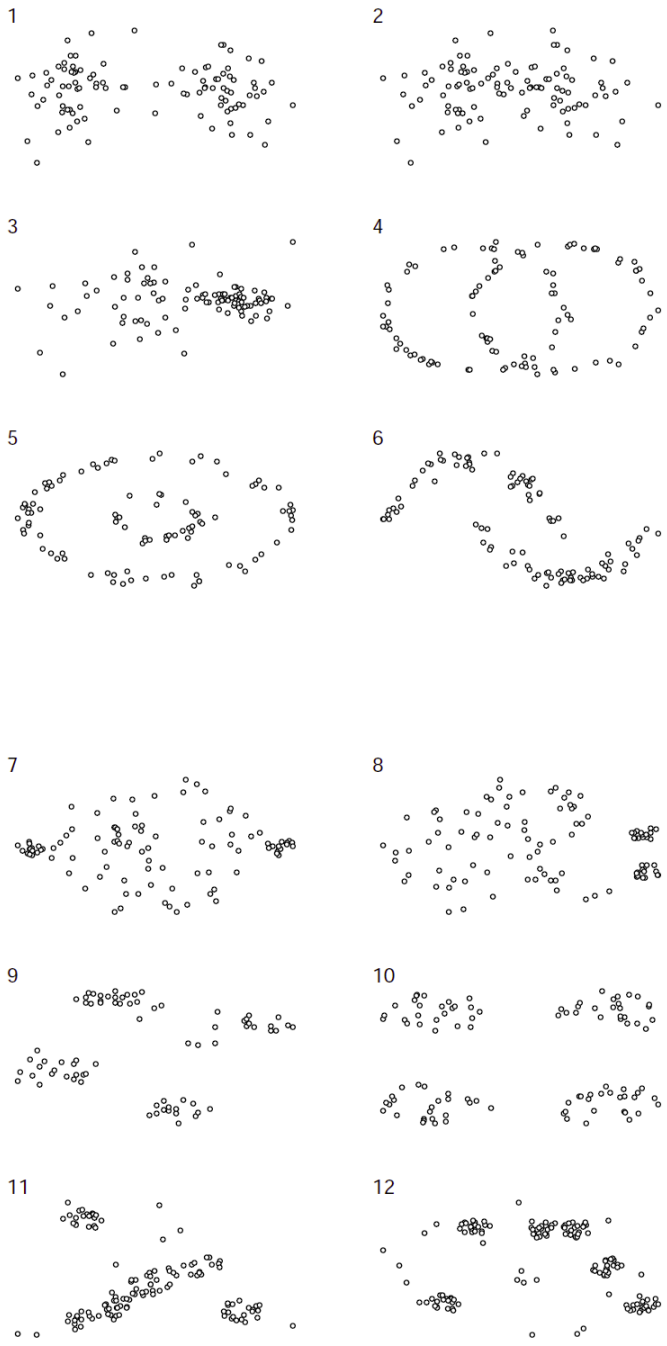
5 conjuntos de datos y 5 pares de centros iniciales o semillas (* y *)



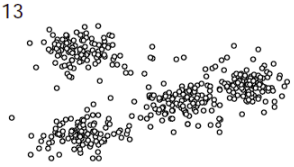
3.2.2 Dos variables

En el siguiente ejercicio se plantean 14 conjuntos de puntos, y para cada uno de ellos se pueden plantear muchas inicializaciones del algoritmo K−means en función del valor de K y de las posiciones los centros iniciales o semillas:

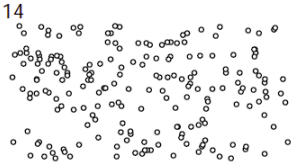
1. Observa la disposición de los puntos
2. Define las clases visualmente
3. Usa el applet K−means, y dispón los puntos adecuadamente
4. Dispón K semillas para la formación de las clases
5. Trata de reproducir el algoritmo mentalmente, y pronostica el resultado
6. Compara tus pronósticos con el comportamiento del algoritmo K−means
7. Experimenta con otras posiciones de las semillas y diferentes valores de K
8. Haz diversas comparaciones con las distintas distancias ofrecidas


1
2
3
4
5
6
7
8
9
10
11
12

13



14



4 Estudio de la función 'kmeans()'

Este apartado trata de enseñar a manejar los parámetros y características de la función '`kmeans()`' de ***R***.

Se puede encontrar información precisa en este enlace.

* Los apartados señalados con * no son imprescindibles para manejar '`kmeans()`', pero ayudan a profundizar en el conocimiento de su funcionamiento.

```
#######################################################################
#
# 'k-means', 'pam' ('Partitioning Around Medoids') y 'clara' (CLustering
# LARge Applications) son algoritmos de clustering que definen particiones
# sobre un número predefinido 'k' de clases.
#
# #####################################################################
#
# Introducción de los datos: 'iris'
#
?iris
#
# Para una clasificación no supervisada no se toma en cuenta
# la variable 'Species' de la planta (5a. columna)
#
datos0 <- iris[,1:4]
#
# Características generales del conjunto de datos
#
dim(datos0); nrow(datos0); ncol(datos0)
#
head.matrix(datos0)
#
summary(datos0)
#
# Preprocesamiento
#
# ¿Estandarizar las variables?
#    Son homogéneas (cm.); es razonable tratarlas tal cual.
#    Sin embargo, algunas medidas son más grandes ('*.Length') que otras,
#    y tratarlas tal cual equivaldría, en cierto sentido,
#    a analizarlas por su apariencia visual.
#
#    Estandarizar las variables equivale, en principio, a
#    poner todas las variables a un mismo nivel en el análisis.
#    Esta es la opción que se hace en el siguiente análisis
#
# Para estandarizar las variables (media 0, varianza 1)
#
datos <- scale(datos0)
#
# si no fueran estandarizadas:
#
#                  datos <- datos0
#
# Verificación:
#
for(j in 1:ncol(datos)) print(c(mean(datos[,j]), var(datos[,j])))
#
# En lenguaje R es más propio verificarlo así, sin bucles explícitos:
#
apply(datos,2,mean); apply(datos,2,var)
#
#######################################################################
#
# Estudio de la función 'kmeans()':
#
#   http://en.wikipedia.org/wiki/K-means
#   http://es.wikipedia.org/wiki/K-means
#   http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/
#                                                          K-Means
#   Ejecutar el siguiente comando:
#
#   > example(kmeans)
#
?kmeans
#
#
# #####################################################################
#
# Primer ensayo con 'kmeans()'
#
# Fijación de las semillas iniciales para obtener los mismos resultados
#          en distintas ejecuciones del siguiente código
#
set.seed(2011)
#
# 'k' arbitrario
#
k <- 5
#
km <- kmeans(datos, k)
#
km
#
# Características generales del objeto 'km'
#
class(km)
mode(km)
attributes(km)
#
# Tamaño de las 'k' clases resultantes
#
km$size
#
# Pertenencia de los objetos a las clases
#
km$cluster
#
# Composición de las clases
#
compocl <- vector("list", k)
for(kcl in 1:k) compocl[[kcl]] <- which(km$cluster == kcl)
compocl
#
# #####################################################################
#
# Índice de la bondad de la partición resultante: 'IB'
#
#   'km$tot.withinss': Suma de los cuadrados ('squared sum')
```

```
#                     de las diferencias de todos ('tot') los objetos
#                     respecto al centro de su respectiva clase
#                     Expresa la variabilidad dentro ('within') de las clases
#
#   'km$betweenss'   : Suma de los cuadrados ('squared sum')
#                     de las diferencias de los centros de cada clase
#                     respecto al centro del conjunto de datos
#                     Expresa la variabilidad entre ('between') las clases
#
#   'km$totss'       : Suma de los cuadrados ('squared sum')
#                     de las diferencias de todos ('tot') los objetos
#                     respecto al centro del conjunto de datos
#                     Expresa la variabilidad total del conjunto de datos
#
#   Se cumple la siguiente igualdad:
#
km$totss == km$tot.withinss + km$betweenss
#
#   'IB' expresa la proporción de variabilidad debida a
#        la formación de las clases
#
IB <- km$betweenss/km$totss # IB <- 1-km$tot.withinss/km$totss
IB
#
#     0 <= IB <= 1
#
#          Si IB=0, entonces las clases formadas no tienen sentido
#                   pues los centros de clases coinciden
#                   (km$betweenss=0)
#          Si IB=1, entonces las clases formadas son homogéneas,
#                   pues los elementos de la misma clase coinciden
#                   (km$tot.withinss=0)
#
#     Un valor de 'IB' cercano a 1
#     indica que las clases son homogéneas y separadas
#
######################################################################
#
# Contraste de la adecuación de las clases:
#
# * Representación gráfica de las clases por pares de variables
#
pairs(datos0, col=km$cluster+1, las=1,
      main=paste("Iris data set\nk-means, k=", k, ", IB=", round(IB,4)),
      font.main=4)
#
#   Se observan dos clases bien diferenciadas,
#   una de las cuales es alargada
#
#
# * Imagen coloreada del conjunto de datos
#   en función de las clases formadas y de las distancias
#
#   Reordenamiento de los elementos en función de las clases resultantes
#
sortdatos <- sort(km$cluster, index.return=TRUE)$ix
datosort <- datos[sortdatos, ]
#
#   Distancia euclidiana:
#          http://es.wikipedia.org/wiki/Distancia_euclidiana
#
distancias <- dist(datosort, method="euclidean")
#
#   Las distancias euclidianas cuadráticas 'distancias^2'
#   están en consonancia con la noción de varianza generalizada:
#
#   sum(distancias^2)/nrow(datosort)^2 == km$totss/nrow(datosort)
#
#   Los valores de 'distancias^2' están implícitamente en
#   la función objetivo de 'kmeans()':
#
#   km$totss == km$tot.withinss + km$betweenss
#
#   IB <- km$betweenss/km$totss # IB <- 1-km$tot.withinss/km$totss
#
#   Construcción de la imagen: 'image()'
#
#   Propuesta: ejecutar los siguientes comandos
#
#   > example(image)
#   > ?image
#
#   Cuanto más cerca están los objetos entre sí
#   mayor es la intensidad del color rojo (mapa de calor)
#
image(as.matrix(distancias^2), col=heat.colors(12), axes=FALSE)
#
#   Conviene completar la imagen para su mejor interpretación
#
#   Marcas de los bordes por clases
#
atpoints <- c(0,cumsum(km$size))/max(cumsum(km$size))
for(side in 1:4) axis(side, at=atpoints, col.axis="transparent")
atpoints2 <- c(atpoints[-1],0)
atpoints2 <- (atpoints+atpoints2)/2
atpoints2 <- atpoints2[-length(atpoints2)]
#
etclases <- sort(unique(km$cluster))
for(side in 1:4)
 for(etcl in 1:length(etclases))
  axis(side,
       at=atpoints2[etcl], labels=etclases[etcl], col.axis=etclases[etcl],
       las=1, tick=FALSE, cex.axis=0.7)
#
#   Tamaño de las clases
#
text(atpoints2, atpoints2, km$size, col="white")
#
######################################################################
#
#   Se observan dos clases, en una de las cuales
#   se podrían considerar algunas subclases
#
```

```
#########################################################################
# Contraste de la adecuación de las clases
#
selec <- c(1,2) # cualquier par de variables
#
plot(datos0[,selec], col=km$cluster, las=1,
     main=paste("Iris data set\nk-means, k=", k, ", IB=", round(IB,4)),
     font.main=4, xlab=names(datos0)[selec[1]], ylab=names(datos0)[selec[2]])
#
#                      Ubicación de los centros de clase
#
kmcenters <- matrix(rep(0, k*ncol(datos0)), ncol=ncol(datos0))
for(j in 1:ncol(datos0))
      kmcenters[,j] <- tapply(datos0[,j], km$cluster, mean)
rownames(kmcenters) <- 1:k; colnames(kmcenters) <- colnames(kmcenters)
#
points(kmcenters[etclases,selec],
       col=etclases, pch=19, cex=2)
#
legend("topleft", bty="n",
       x.intersp=1, y.intersp=1,
       legend=km$size[etclases], text.col=etclases,
       pch=19, col=etclases, ncol=1, cex=1)
#
#########################################################################
```

Youtube

Clustering - k-means - R - iris - 1

How to Perform K-Means Clustering in R Statistical Computing

K Means/ Hierarchical Clustering in R

```
mp4
#########################################################################
#
# Verificación de los resultados de 'km()'
#
# #######################################################################
#
# Recuperación del ejemplo tratado anteriormente
#
# Introducción de los datos: 'iris'
#
datos0 <- iris[,1:4]
#
# Estandarización de las variables (media 0, varianza 1)
#
datos <- scale(datos0)
#
# Fijación de las semillas iniciales para obtener los mismos resultados
#          en distintas ejecuciones del siguiente código
#
set.seed(2011)
#
# 'k' arbitrario
#
k <- 5
km <- kmeans(datos, k)
#
# #######################################################################
#
attributes(km)
#
# Verificación de los cálculos:
#                               km$size
#                               km$centers
#                               km$withinss
#                               km$tot.withinss
#                               km$betweenss
#                               km$totss
#
# ##########
#
# km$size
#
km$size
sum(km$size) == nrow(datos)
#
# ##########
#
# km$centers
#
# Cálculo de las medias de las variables por cada clase
#
medias.cl <- matrix(rep(0, k*ncol(datos)), ncol=ncol(datos))
for(j in 1:ncol(datos))
   medias.cl[,j] <- tapply(datos[,j], km$cluster, mean)
#
rownames(medias.cl) <- 1:k; colnames(medias.cl) <- colnames(datos)
medias.cl
km$centers
medias.cl == km$centers # FALSE!!!
#
# Las medias (ponderadas) de las medias parciales de cada clase
# coinciden con las medias totales (0, si las variables son estandarizadas)
#
medias.pond <- vector()
for(j in 1:ncol(datos))
   medias.pond[j] <- sum(km$size*km$centers[,j])/nrow(datos)
names(medias.pond) <- colnames(datos)
medias.pond
#
medias.tot <- apply(datos, 2, mean)  # colMeans(datos)
medias.tot
medias.pond == medias.tot # FALSE!!!
#
# ##########
#
# km$withinss
#
# Cálculo de las varianzas de las variables
# dentro de cada clase
#
varcl <- matrix(rep(0, k*ncol(datos)), ncol=ncol(datos))
for(j in 1:ncol(datos))
   varcl[,j] <- tapply(datos[,j], km$cluster, var)
rownames(varcl) <- 1:k; colnames(varcl) <- colnames(datos)
varcl
#
# Cálculo de las sumas de las diferencias cuadráticas
# de todas las variables dentro de cada clase
#
SSWcl <- vector()
for(kcl in 1:k)
   SSWcl[kcl] <- sum((km$size[kcl]-1)*varcl[kcl,])
SSWcl
km$withinss
km$withinss == SSWcl # FALSE!!!
#
# ##########
#
# km$tot.withinss
#
# Cálculo de las sumas de las diferencias cuadráticas
# de todas las variables dentro de todas las clases
#
SSW <- sum(SSWcl)
SSW
km$tot.withinss
km$tot.withinss == SSW # FALSE!!!
km$tot.withinss == sum(km$withinss)
#
```

```
# ##########
# km$betweenss
#
# Cálculo de varianzas de las variables
# siendo cada clase representada por su centro (media)
#
varB.pond <- vector() # varB.pond[j], varianza ponderada de cada variable
for(j in 1:ncol(datos))
   varB.pond[j] <- sum(km$size*(km$centers[,j]-medias.tot[j])^2)/nrow(datos)
names(varB.pond) <- colnames(datos)
varB.pond
#
# Cálculo de sumas de las diferencias cuadráticas de las variables
# siendo cada clase representada por su centro (media)
#
SSBj <- nrow(datos)*varB.pond
SSBj
#
# Cálculo de las sumas de las diferencias cuadráticas
# de todas las variables y de todas las clases,
# siendo cada clase representada por su centro (media)
#
SSB <- sum(SSBj)
SSB
km$betweenss
km$betweenss == SSB # FALSE !!!
#
# ##########
#
# km$totss
#
# Cálculo de las sumas de las diferencias cuadráticas
# de todas las variables
#
SST <- 0
for(j in 1:ncol(datos))
   SST <- SST + sum((nrow(datos)-1)*var(datos[,j]))
SST
#
SST == SSW + SSB
#
km$totss
km$totss == SST
km$totss == km$tot.withinss + km$betweenss
#
# ############################################################################
```

```
#########################################################################
#
# Interpretación de una clasificación
#
# #######################################################################
#
# Recuperación del ejemplo tratado anteriormente
#
# Introducción de los datos: 'iris'
#
datos0 <- iris[,1:4]
#
# Cálculo de los estadísticos principales por variables
#
medias.tot <- apply(datos0, 2, mean); medias.tot
var.tot  <- apply(datos0, 2, var);    var.tot
desv.st.tot  <- apply(datos0, 2, sd); desv.st.tot
#
# Estandarización de las variables (media 0, varianza 1)
#
datos <- scale(datos0)
#
# Fijar las semillas iniciales para obtener los mismos resultados
#
set.seed(2011)
#
# 'k' arbitrario
#
k <- 5
km <- kmeans(datos, k)
attributes(km)
#
# #######################################################################
#
# Interpretación de la clasificación:
#                                     - Índice de bondad
#                                     - Análisis de estadísticos
#                                     - Representación gráfica
#
#
# Índice de la bondad de las clases formadas: 'IB'
#
IB <- km$betweenss/km$totss # IB <- 1-km$tot.withinss/km$totss
IB
#
#     0 <= IB <= 1
#          Si IB=0, entonces las clases formadas no tienen sentido
#                   pues los centros de clases coinciden
#                   (km$betweenss=0)
#          Si IB=1, entonces las clases formadas son homogéneas,
#                   pues los elementos de la misma clase coinciden
#                   (km$tot.withinss=0)
#
#
# Análisis de estadísticos de cada variable por clases
#
#              Media de cada variable por clases
km$centers
#                   Se observa que los centros de las clases:
#                     '1' y '2' se separan de los de '3', '4' y '5'
#                     '1'<'2' y '5'<'3'<'4'
#
#              Varianza de cada variable por clases
#
varcl <- matrix(rep(0, k*ncol(datos)), ncol=ncol(datos))
for(j in 1:ncol(datos))
   varcl[,j] <- tapply(datos[,j], km$cluster, var)
rownames(varcl) <- 1:k; colnames(varcl) <- colnames(datos)
#
varcl
#                 Se observa que las variables
#                   'Petal.Length' y 'Petal.Width'
#                   son las de menor varianza en todas las clases
#                   por tanto, las más características de cada clase
#
#              Varianza de cada variable entre clases
#
medias.pond <- vector()
for(j in 1:ncol(datos))
   medias.pond[j] <- sum(km$size*km$centers[,j])/sum(km$size)
names(medias.pond) <- colnames(datos)
medias.pond # si ha habido estandarización, todas nulas
#
varB.pond <- vector() # varB.pond[j], varianza ponderada por clases
for(j in 1:ncol(datos))
   varB.pond[j] <-
       sum(km$size*(km$centers[,j]-medias.pond[j])^2)/sum(km$size)
names(varB.pond) <- colnames(datos)
#
varB.pond
#
sum(varB.pond)*nrow(datos); km$betweenss # Verificación de la descomposición de B
#
#                 Se observa que las variables
#                   'Petal.Length' y 'Petal.Width' son las de mayor
#                   varianza, por tanto, las más características de
#                   la constitución de las clases
#
#
# Representación gráfica de las clases:
#
#              Por cada par de variables
#
pairs(datos0, col=km$cluster, las=1,
      main=paste("Iris data set\nk-means, k=", k, ", IB=", round(IB,4)),
      font.main=4)
#              Por un par de variables características
#
elecx <- 3; elecy <- 4         # 'Petal.Length' y 'Petal.Width'
```

```
plot(datos0[,c(elecx,elecy)], col=km$cluster, las=1,
     main=paste("Iris data set\nk-means, k=", k, ", IB=", round(IB,4)),
     font.main=4, xlab=names(datos0)[elecx], ylab=names(datos0)[elecy])
#
#                      Ubicación de los centros de clase
#
kmcenters <- matrix(rep(0, k*ncol(datos0)), ncol=ncol(datos0))
for(j in 1:ncol(datos0))
      kmcenters[,j] <- tapply(datos0[,j], km$cluster, mean)
rownames(kmcenters) <- 1:k; colnames(kmcenters) <- colnames(kmcenters)
#
kmcenters
#
etclases <- unique(km$cluster) # etiqueta de clases
#
points(kmcenters[etclases,c(elecx,elecy)],
       col=etclases, pch=19, cex=2)
#
legend("bottomright", bty="n",
       x.intersp=1, y.intersp=1,
       legend=km$size[etclases], text.col=etclases,
       pch=19, col=etclases, ncol=1, cex=1)
#
# ##########################################################################
```

```
########################################################################
#
# El resultado depende de las semillas iniciales
#
# ######################################################################
#
# Recuperación del ejemplo tratado anteriormente
#
# Introducción de los datos: 'iris'
#
datos0 <- iris[,1:4]
#
# Estandarización de las variables (media 0, varianza 1)
#
datos <- scale(datos0)
#
# Fijar las semillas iniciales para obtener los mismos resultados
#
set.seed(2011)
#
# 'k' arbitrario
#
k <- 5
km <- kmeans(datos, k)
attributes(km)
#
km$size
#
# Índice de la bondad de las clases formadas: 'IB'
#
IB <- km$betweenss/km$totss # IB <- 1-km$tot.withinss/km$totss
IB
#
# ######################################################################
#
# El resultado depende de las semillas iniciales
#
#       Si no se fija la semilla  'set.seed(xxxx)'
#       los resultados son diversos
#
k <- 5
tabla <- NULL
for(i in 1:20) {
   km <- kmeans(datos, k)
   tabla <- rbind(tabla,
                  c(km$size, round(km$betweenss/km$totss, digits=6)))
}
nombrecl <- vector()
for(kcl in 1:k) nombrecl[kcl] <- paste("cl", kcl, sep="")
# nombrecl
colnames(tabla) <- c(nombrecl, "IB")
#
tabla
#
#         Se observa que los resultados son diversos
#
# Para evitar esta diversidad y aproximarse a un máximo
# se ejecutar el algoritmo 'kmeans()' 'nstart' veces.
# 'kmeans()' selecciona automáticamente el mejor resultado
#
# Dependiendo del tamaño de 'datos'
# 'nstart' no asegura la consecución de un máximo total.
#
# También hay que asegurar que el algoritmo se detenga,
# pues la convergencia a un máximo local no está asegurada
# en un predeterminado número de iteraciones:
# Fijar 'iter.max' pasos
#
k <- 5
tabla <- NULL
for(i in 1:20) {
  km <- kmeans(datos, k, nstart=100, iter.max=1000)
   tabla <- rbind(tabla,
                  c(km$size, round(km$betweenss/km$totss, digits=6)))
}
nombrecl <- vector()
for(kcl in 1:k) nombrecl[kcl] <- paste("cl", kcl, sep="")
colnames(tabla) <- c(nombrecl, "IB")
#
tabla
#
#         Se observa que los resultados son los mismos,
#         salvo la etiquetación u orden de las clases
#         (se podría comprobar que
#         no sólo los tamaños de las clases coinciden,
#         sino también la composición de las mismas)
#
# ######################################################################
```

```
#########################################################################
#
# Siluetas
#
# #######################################################################
#
# Recuperación del ejemplo tratado anteriormente
#
# Introducción de los datos: 'iris'
#
datos0 <- iris[,1:4]
#
# Estandarización de las variables (media 0, varianza 1)
#
datos <- scale(datos0)
#
# Construcción de una partición
#
k <- 3
km <- kmeans(datos, k, nstart=100, iter.max=10000)
#
# Composición de las clases
#
compocl <- vector("list", k)
for(kcl in 1:k) compocl[[kcl]] <- which(km$cluster == kcl)
compocl
#
# #######################################################################
#
# Construcción de siluetas mediante la función 's(i)'
#          asignado a cada objeto 'i'.
#
#          http://en.wikipedia.org/wiki/Silhouette_%28clustering%29
#
#   s(i) = (b(i)-a(i))/max(b(i),a(i))
#
#   a(i) = disimilitud media del objeto 'i' con los de su clase
#   b(i) = mínima disimilitud media del objeto 'i' con los de otras clases
#
# Hay que definir una distancia (disimilitud) entre los objetos 'i'
#
# La distancia debe estar en consonancia
# con la función objetivo de 'kmeans()'
#
distancias <- dist(datos, method="euclidean", upper=TRUE, diag=TRUE)^2
distancias.m <- as.matrix(distancias)
#
# Cálculo de los valores 'a(i)', 'b(i)', y 's(i)'
#
dmean <- matrix(rep(NA, nrow(datos)*k), ncol=k)
a <- rep(NA, nrow(datos))
b <- rep(NA, nrow(datos))
s <- rep(NA, nrow(datos))
#
for(i in 1:nrow(datos)){
  for(kcl in 1:k) dmean[i,kcl] <- mean(distancias.m[i, compocl[[kcl]]])
  kcli <- km$cluster[i]
  a[i] <- dmean[i,kcli]*km$size[kcli]/(km$size[kcli]-1)
  b[i] <- min(dmean[i,-kcli])
  s[i] <- (b[i]-a[i])/max(b[i],a[i])
}
#
# Cálculo de los valores 's(i)' ('width'),
#         mediante la función 'silhouette()'
#
library(cluster)
#
silk <- silhouette(km$cluster, distancias)
#
silk
#
# Características generales del objeto 'silk'
#
class(silk)
mode(silk)
attributes(silk)
#
cluster <- silk[,1] # clase del objeto 'i'
table(cluster)
#
neighbor <- silk[,2] # clase vecina o más próxima al objeto 'i'
table(neighbor)
#
width <- silk[,3] # valor del índice de silueta 's(i)'
summary(width)
hist(width)
#
summary(silk)
attributes(summary(silk))
#
summary(silk)$clus.avg.widths
mediascl <- tapply(width, cluster, mean)
#
summary(silk)$avg.width
mean(s)
#
# Verificación de que
# los valores 'sil_width' (3a. columna) son los 's(i)'
#
cbind(s, silk[,3])
s == width # FALSE !!!
#
# El valor medio del índice 's(i)' ('avg.width'),
# se usa para evaluar la bondad de la partición
#
# Una evaluación más detallada de la bondad de la partición
# se puede hacer mediante las siluetas de las clases
#
# El análisis de los valores medios por clase
# y la negatividad de algunos valores
```

```
# se puede usar para determinar el número apropiado de clases 'k'
etclases <- sort(unique(km$cluster))
plot(silk, col=etclases)
#
#
# Las siluetas sólo dependen de la partición,
# no dependen del algoritmo para obtenerlas
#
# Así, las siluetas valen para comparar los resultados de diferentes clustering
# aplicados a los mismo conjunto de datos
#
# Las siluetas valen para interpretar y validar los resultados
# 'k' pequeño, clases naturales mezcladas:
#             Si 'SSW' grande, 'a(i)' grande, s(i) pequeños.
# 'k' grande, clases naturales separadas:
#             Si 'SSB' pequeño, 'b(i)' pequeño, s(i) pequeños.
#
# Valor medio global puede valer para seleccionar el valor de 'k'
#
# Las siluetas pueden servir para mejorar los resultados del análisis,
# por ejemplo, moviendo un objeto con valor de s(i)<0 a su clase vecina
#
mean(silk[,3])
# [1] 0.6507398
#
which(silk[,3] < 0)
# [1] 112 128
silk[which(silk[,3] < 0), 2]
# [1] 3 3      las calses vecinas
km$cluster.new <- km$cluster
km$cluster.new[which(silk[,3] < 0)] <- silk[which(silk[,3] < 0), 2]
#
silk <- silhouette(km$cluster.new, distancias)
etclases <- sort(unique(km$cluster))
plot(silk, col=etclases+1)
mean(silk[,3])
# [1] 0.6529184 # se ha mejorado el valor del indice
#
which(silk[,3] < 0)
# [1] 109
silk[which(silk[,3] < 0), 2]
km$cluster.new2 <- km$cluster.new
km$cluster.new2[which(silk[,3] < 0)] <- silk[which(silk[,3] < 0), 2]
kmsize.new2 <- as.vector(table(km$cluster.new2))
#
silk <- silhouette(km$cluster.new, distancias)
etclases <- sort(unique(km$cluster))
plot(silk, col=etclases+1)
mean(silk[,3])
# [1] 0.6529184 # no se ha mejorado el valor del indice
#
#########################################################################
```

```
#########################################################################
#
# Selección del valor de 'k' (el número de clases)
#
# #######################################################################
#
# Recuperación del ejemplo tratado anteriormente
#
# Introducción de los datos: 'iris'
#
datos0 <- iris[,1:4]
#
# Estandarización de las variables (media 0, varianza 1)
#
datos <- scale(datos0)
#
# 'k' arbitrario
#
k <- 5
km <- kmeans(datos, k, nstart=100, iter.max=10000)
#
attributes(km)
#
#########################################################################
#
# Selección del valor de 'k':
#                            Índice de bondad, 'IB'
#                            Índice Davies-Bouldin
#                            Valor medio de índice de siluetas
#
# ##########
#
# Índice de la bondad de la partición: 'IB'
#
IB <- km$betweenss/km$totss # IB <- 1-km$tot.withinss/km$totss
IB
#
# Cálculo del valor de 'IB' para cada valor de 'k'
# buscando el máximo para cada uno de ellos (con 'nstart'=100)
#
IB <- vector()
for(k in 1:20)
{
  km <- kmeans(datos, k, nstart=100, iter.max=10000)
  IB[k] <- km$betweenss/km$totss
}
#
cbind(k=1:length(IB), IB=round(IB, digits=6))
#
# Representación gráfica de la variación de 'IB' en función de 'k'
#
plot(IB, type="b", las=1, xlab="k")
#
# Selección del valor de 'k'
# tras el análisis de los valores de 'IB'
#
# El valor de 'IB' crece, lógicamente, con 'k',
# pues a mayor 'k' mayor 'km$betweenss'
#
# Así, analizar los valores de 'IB'
# consiste en analizar su crecimiento respecto a 'k':
# Si hay un salto grande entre dos consecutivos,
# entonces es señal de que hay dos clases ciertamente separadas
#
selectk <- NULL
for(k in 3:length(IB))
  selectk <- rbind(selectk, c(k,  IB[k]-IB[k-1],  IB[k]/IB[k-1]))
colnames(selectk) <- c("k",  "IB[k]-IB[k-1]",  "IB[k]/IB[k-1]")
selectk
#
# Observados los saltos ("IB[k]-IB[k-1]",  "IB[k]/IB[k-1]"),
# k=2 parece una buena elección,
# y k=3 podría serla también
#
# ##########
#
# Índice de Davies-Bouldin
#           http://en.wikipedia.org/wiki/Davies-Bouldin_index
#
# La distancia debe estar en consonancia
# con la función objetivo de 'kmeans()'
#
distancias <- dist(datos, method="euclidean")^2 # distancia cuadrática
#
# Adecuación del objeto 'distancias' a un tipo de objeto manejable para lo que sigue
#
distancias.m <- as.matrix(distancias, ncol=nrow(datos)) # formato necesario
#
k <- 3
km <- kmeans(datos, k, nstart=100, iter.max=10000)
#
# Composición de las clases
#
compocl <- vector("list", k)
for(kcl in 1:k) compocl[[kcl]] <- which(km$cluster == kcl)
#
# Definición del índice 'DB' mediante una función
#
DB <- function(distancias.m, compocl, k){
#
#   Distancias medias 'dentro de' y 'entre' las clases: 'Mcl'
#                     'dentro de': diagonal de 'Mcl'
#                     'entre'    : fuera de la diagonal de 'Mcl'
 Mcl <- matrix(rep(0,k*k), ncol=k) # inicialización
 for(kcli in 1:k){
   for(kclj in 1:k){
     Mcl[kcli,kclj] <- mean(distancias.m[compocl[[kcli]], compocl[[kclj]]])
   }
 }
#
```

```
  # Cálculo de ratios
   R <- matrix(rep(0,k*k), ncol=k)
   for(kcli in 1:k){
     for(kclj in 1:k){
       R[kcli,kclj] <- (Mcl[kcli,kcli]+Mcl[kclj,kclj])/Mcl[kcli,kclj]
     }
   }
  #
  # Cuanto menor sea R, mayor será la separación entre clases
  #
  # Se adopta el peor de los casos para definir el índice
  #
   D <- rep(0,k)
   for(kcl in 1:k){
      D[kcl] <- max(R[kcl,-kcl])
   }
  #
  # Cálculo del valor del índice de Davies-Bouldin
  #
   DB <- mean(D)
  #
  # Cuanto menor es el índice, mejor es la partición
  #
   return(DB)
  #
  }
  #
  # Cálculo del valor de 'DB' para cada valor de 'k'
  # buscando el mínimo
  #
  db <- NA # db[1] no tiene sentido
  for(k in 2:20)
  {
    km <- kmeans(datos, k, nstart=100, iter.max=10000)
    compocl <- vector("list", k)
    for(kcl in 1:k) compocl[[kcl]] <- which(km$cluster == kcl)
    db[k] <- DB(distancias.m, compocl,k)
  }
  #
  cbind(k=1:length(db), DB=round(db, digits=6))
  #
  # Representación gráfica de la variación de 'db' en función de 'k'
  #
  plot(db, type="b", las=1, xlab="k")
  #
  # Observados los saltos, k=2 parece una buena elección
  #
  # ##########
  #
  # Valor medio del índice de siluetas 's(i)'
  #
  k <- 3
  km <- kmeans(datos, k, nstart=100, iter.max=10000)
  #
  # La distancia debe estar en consonancia
  # con la función objetivo de 'kmeans()'
  #
  distancias <- dist(datos, method="euclidean")^2
  #
  # Cálculo del valor del índice 's(i)' ('widths'),
  #         asignado a cada objeto 'i',
  #         mediante la función 'silhouette()'
  #         http://en.wikipedia.org/wiki/Silhouette_%28clustering%29
  #
  library(cluster)
  #
  silk <- silhouette(km$cluster, distancias)
  #
  class(silk)
  mode(silk)
  attributes(silk)
  #
  summary(silk)
  attributes(summary(silk))
  #
  # El valor medio del índice 's(i)' ('avg.width'),
  # es también un índice de la bondad de la partición
  #
  summary(silk)$avg.width
  #
  # Se analiza este valor para distintos valores de 'k'
  #
  avg.width <- NA # avg.width[1] no tiene sentido
  for(k in 2:20)
  {
    km <- kmeans(datos, k, nstart=100, iter.max=10000)
    silk <- silhouette(km$cluster, distancias)
    avg.width[k] <- summary(silk)$avg.width
  }
  #
  cbind(k=1:length(avg.width), avg.width=round(avg.width, digits=6))
  #
  #
  # Representación gráfica de la variación de 'avg.width' en función de 'k'
  #
  plot(avg.width, type="b", las=1, xlab="k")
  #
  selectk <- NULL
  for(k in 3:length(IB))
    selectk <- rbind(selectk, c(k,  avg.width[k]-avg.width[k-1],
                                    avg.width[k]/avg.width[k-1]))
  colnames(selectk) <- c("k",  "avg.width[k]-avg.width[k-1]",
                               "avg.width[k]/avg.width[k-1]")
  selectk
  #
  # Observados los saltos "avg.width[k]-avg.width[k-1]",
  #                       "avg.width[k]/avg.width[k-1]",
  # k=3 parece una buena elección,
  #
  ###########################################################################
```

```
#########################################################################
#
# Introducción de los datos: 'iris'
#
datos0 <- iris[,1:4]
#
# Estandarización de las variables (media 0, varianza 1)
#
datos <- scale(datos0)
#
#########################################################################
#
# Selección del valor del parámetro 'k': k=2
#
k <- 2
km <- kmeans(datos, k, nstart=100, iter.max=10000)
#
# La distancia debe estar en consonancia
# con la función objetivo de 'kmeans()'
#
distancias <- dist(datos, method="euclidean")^2
#
# Siluetas
#
library(cluster)
#
silk <- silhouette(km$cluster, distancias)
#
etclases <- sort(unique(km$cluster))
plot(silk, col=etclases+1)
#
# Cálculo de medias por clase
#
medias.cl <- matrix(rep(0, k*ncol(datos)), ncol=ncol(datos))
for(j in 1:ncol(datos))
   medias.cl[,j] <- tapply(datos[,j], km$cluster, mean)
#
# Cálculo de medias total. Si variables son estandarizadas, 0
#
medias.tot <- apply(datos, 2, mean)  # colMeans(datos)
medias.tot
#
# Cálculo de varianzas entre clases por variable
#
varB.pond <- vector() # varB.pond[j], varianza ponderada de cada variable
for(j in 1:ncol(datos))
   varB.pond[j] <- sum(km$size*(medias.cl[,j]-medias.tot[j])^2)/nrow(datos)
names(varB.pond) <- colnames(datos)
varB.pond
#
# Sepal.Length  Sepal.Width Petal.Length  Petal.Width
#    0.5112540    0.3616017    0.8458193    0.7821296
#
# Las dos variables del Sépalo son las que más explican
# la consideración de las dos clases o conglomerados
#
# Cálculo de sumas de errores cuadráticos entre clases
#
SSBj <- nrow(datos)*varB.pond; SSBj
#
SSB <- sum(SSBj) # km$betweenss
IB <- km$betweenss/km$totss
IB
#
# La consideración de estas dos clases o conglomerados (clusters)
# explica casi el 63% de la varianza inicial (bastante alto)
# y se debe, principalmente, a los variables del sépalo
#
pairs(datos, pch=21, col="black", bg=km$cluster+1)
#
#########################################################################
```

5 Ejercicios

5.1 Una variable y dos clases

En los siguientes ejercicios se plantean 5 conjuntos de puntos, que puedes encontrarlos en el fichero ejercicios1.RData. Su representación gráfica se encuentra en el punto 3.2.1 del libro de 'K-means'

Para cada uno de ellos se plantea un par de semillas o inicializaciones del algoritmo K−means (*k=2*):

```
#
centers <- rbind(   c(10  ,  16)   # semillas para el conjunto 1
                  , c( 8  ,  20)   # semillas para el conjunto 2
                  , c( 8.3,  10)   # semillas para el conjunto 3
                  , c(14  ,  15)   # semillas para el conjunto 4
                  , c(13  ,  18) ) # semillas para el conjunto 5
#
```

Se trata de encontrar las soluciones mediante **código *R*** en lugar de hacerlo con el applet.

Usando la función '`kmeans()`', escribe el código ***R*** que dé una solución a cada uno de ellos, y representa la solución correspondiente gráficamente.

5.2 Dos variables y dos clases

Se plantean los 14 conjuntos de objetos del punto 3.2.2 del libro de 'K-means', que pueden descargarse desde el fichero ejercicios2.RData.

Usando '`kmeans()`' y *k=2*, escribe el código ***R*** que dé una solución a cada uno de ellos, y representa la solución correspondiente gráficamente.

5.3 Múltiples variables

Mediante la función '`kmeans()`' haz un análisis de conglomerados (clustering) del fichero de datos '`swiss`' que está en el paquete '`datasets`' de ***R***, y que se carga automáticamente al inicio.

```
> ?swiss
```

```
################################################################################
#
# Ejercicios de 'kmeans()' con una única variable
#
################################################################################
#
opar <- par()
#
################################################################################
#
# Función para representar gráficamente los ejercicios
#
ejercicios1.dat <- function(data, titulo="", subtitulo="")
{
  #
  par(mar=c(5,4,8,2)+0.1)
  stripchart(x~example, data=data, method="overplot"
             , pch=21, cex=0.8, col="black", bg="white"
             , axes=FALSE, xlab="", ylab="", frame.plot=FALSE)
  title(titulo)
  mtext(subtitulo, side=3, line=4, font=2, cex=1)
  nexamples <- nlevels(data$example)
  axis(2, at=1:nexamples, labels=nexamples:1
       , las=1, tick=FALSE)
  par(mar=c(5,4,4,2)+0.1)
  #
}
#
################################################################################
#
load("ejercicios1.RData")
#
################################################################################
#
# Definicion de los centros iniciales o semillas
#
centers <- rbind(   c(10  ,  16)
                  , c( 8  ,  20)
                  , c( 8.3,  10)
                  , c(14  ,  15)
                  , c(13  ,  18) )
#
################################################################################
#
# Representación gráfica de los ejercicios
#
# Conjuntos de datos
#
ejercicios1.dat(data, subtitulo="Conjuntos de datos")
#
# Conjuntos de datos y semillas
#
nexamples <- nlevels(data$example)
#
for(ic in 1:nrow(centers))
{
  ejercicios1.dat(data, subtitulo="Conjuntos de datos y semillas")
  for(kex in 1:nexamples)
    points(centers[ic,], c(kex, kex), col=(1:ncol(centers))+1, pch=8, cex=1)
}
#
################################################################################
#
# Edición de las soluciones a los ejercicios
#
pdf("ejercicios1solkm.pdf")
#
for(ic in 1:nrow(centers)){
  ejercicios1.dat(data)
  for(kex in 1:nexamples){
    x.m <- data[data$example==kex, 1]
    km <- kmeans(x.m, centers=centers[ic,], nstart=1)
    #
    points(x.m, rep(kex, length(x.m)), pch=21, cex=0.8, col="black", bg=km$cluster+1)
    points(centers[ic,], c(kex, kex), col=(1:ncol(centers))+1, pch=8, cex=0.5)
    points(km$centers, c(kex, kex), col=(1:ncol(centers))+1, pch=8, cex=1.5)
  }
}
#
dev.off()
#
################################################################################
```

5.2 Representación gráfica de las soluciones

ejercicios1solkm.pdf

```
################################################################################
#
# Ejercicios de 'kmeans()' con dos variables y dos clases
#
################################################################################
#
opar <- par()
#
################################################################################
#
# Función para representar gráficamente todos los ejercicios
#
# Para representar solo un ejercicio,
# basta con fijar el valor de 'kex',
# en lugar de hacerlo correr por todo 1:nexamples
#
ejercicios2.dat <- function(data, titulo="", subtitulo="")
{
  #
  par(oma=c(4,4,10,4))
  par(mfrow=c(3,2))
  par(mar=rep(2,4)+0.1)
  nexamples <- nlevels(data$example)
  #
  for(kex in 1:nexamples)
  {
    data1 <- data[data[,3]==kex, 1:2]
    plot(data1, pch=21, cex=0.75, col="black", bg="white"
         , axes=FALSE, xlab="", ylab="", frame.plot=FALSE)
    mtext(kex, side=3, adj=0, cex=1)
    if(kex==1) title("Conjuntos de datos", line=5, outer=TRUE)
  }
  par(mar=c(5,4,4,2)+0.1)
  par(mfrow=c(1,1))
  par(oma=rep(0,4))
  #
}
#
################################################################################
#
load("ejercicios2.RData")
#
################################################################################
#
pdf("ejercicios2.pdf")
#
ejercicios2.dat(data)
#
dev.off()
#
################################################################################################################################################################################
#
# Edición de las soluciones a los ejercicios
#
pdf("ejercicio2solkm.pdf")
#
nexamples <- nlevels(data$example)
#
for(kex in 1:nexamples)
{
  data1 <- data[data[,3]==kex, 1:2]
  x.m <- data1
  km <- kmeans(x.m, centers=2, nstart=10)
#
  plot(data1, pch=21, cex=0.75, col="black", bg="white"
       , axes=FALSE, xlab="", ylab="", frame.plot=FALSE)
  points(x.m, pch=21, cex=0.8, col="black", bg=km$cluster+1)
  points(km$centers, col=(1:nrow(km$centers))+1, pch=8, cex=1.5)
  mtext(kex, side=3, adj=0, cex=1)
  if(kex==1) title("Conjuntos de datos", line=5, outer=TRUE)
}
#
dev.off()
#
################################################################################
```

5.4 Representación gráfica de las soluciones

ejercicios2solkm.pdf

```
#########################################################################
#
nombredatos <- "Swiss data set"
#
head(swiss)
#
datos0 <- swiss
#
# Características generales del conjunto de datos
#
dim(datos0); nrow(datos0); ncol(datos0)
#
summary(datos0)
#
#########################################################################
#
# Los datos son homogéneos: porcentajes
# (luego se analiza el caso de 'heterogéneos')
#
datos <- datos0
#
# Análisis gráfico univariante
#
par(mfrow=c(3,2))
#
for(j in 1:ncol(datos)) hist(datos[,j])
#
par(mfrow=c(1,1))
#
# Análisis bivariante
#
pairs(datos)
#
cor(datos)
#
#########################################################################
#
# k-means
#
# Tanteo para buscar el valor de k
#
ib <- 0
for(k in 2:10)
{
  set.seed(2014)
  km <- kmeans(datos, k, nstart=10)
  print(km$size)
  ib[k] <- km$betweenss/km$totss
}
#
names(ib) <- 1:10
round(ib[2:10], digits=4); plot(2:10, ib[2:10], xlab="k", ylab="IB", las=1)
#      2      3      4      5      6      7      8      9     10
# 0.7011 0.8176 0.8681 0.8986 0.9182 0.9318 0.9426 0.9488 0.9525
#
# Por un equilibrio (subjetivo)
# entre calidad, salto de la funcion 'ib' y número de clases
#
k <- 4
set.seed(2014)
km <- kmeans(datos, k, nstart=10)
km$size
#
###################################
#
# Validación interna:
# Coeficiente IB
#
km$tot.withinss; km$betweenss; km$totss
# [1] 15622.49
# [1] 102860.8
# [1] 118483.3
(IB <- km$betweenss/km$totss)
# [1] 0.868146
#
###################################
#
# Validación interna:
# Siluetas
#
distancias <- dist(datos, method="euclidean", upper=TRUE, diag=TRUE)^2 # cuadratica
distancias.m <- as.matrix(distancias)
#
library(cluster)
#
avg.width <- 0
for(k in 2:10)
{
  set.seed(2014)
  km <- kmeans(datos, k, nstart=10)
  print(km$size)
  silk <- silhouette(km$cluster, distancias)
  avg.width[k] <- summary(silk)$avg.width
}
#
names(avg.width) <- 1:10
round(avg.width[2:10], digits=4); plot(2:10, avg.width[2:10], xlab="k", ylab="avg.width", las=1)
#
# Según el índice, 2 o 4 clases
# podrían constituir una buena solución.
#
k <- 4
set.seed(2014)
km <- kmeans(datos, k, nstart=10)
kmclusthmg <- km$cluster
silk <- silhouette(km$cluster, distancias)
plot(silk, col="blue")
#
# Las clases parecen consistentes
#
```

```
# Representación gráfica de la solución
pdf("swisssolkm1.pdf")
pairs(datos, pch=21, cex=0.75, col="black", bg=km$cluster+1)
dev.off()
#
########################################################################
#
# Los datos no son homogéneos: porcentajes
# ¿significa lo mismo (0.92-0.88) que (0.05-0.01)?
# (0.92-0.88)/0.88 <<< (0.05-0.01)/0.01
#
datos <- scale(datos0, center=TRUE, scale=TRUE)
#
# Análisis gráfico univariante
#
par(mfrow=c(3,2))
#
for(j in 1:ncol(datos)) hist(datos[,j])
#
par(mfrow=c(1,1))
#
# Análisis bivariante
#
pairs(datos)
#
cor(datos)
#
########################################################################
#
# k-means
#
# Tanteo para buscar el valor de k
#
ib <- 0
for(k in 2:10)
{
  set.seed(2014)
  km <- kmeans(datos, k, nstart=10)
  print(km$size)
  ib[k] <- km$betweenss/km$totss
}
#
names(ib) <- 1:10
round(ib[2:10], digits=4); plot(2:10, ib[2:10], xlab="k", ylab="IB", las=1)
#
# Por un equilibrio cualitativo (subjetivo)
# salto de la función ib y número de clases
#
k <- 6
set.seed(2014)
km <- kmeans(datos, k, nstart=10)
km$size
#
###################################
#
# Validación interna:
# Coeficiente IB
#
km$tot.withinss; km$betweenss; km$totss
# [1] 72.55739
# [1] 203.4426
# [1] 276
(IB <- km$betweenss/km$totss)
# [1] 0.7371109
#
###################################
#
# Validación interna:
# Siluetas
#
distancias <- dist(datos, method="euclidean", upper=TRUE, diag=TRUE)^2 # cuadratica
distancias.m <- as.matrix(distancias)
#
library(cluster)
#
avg.width <- 0
for(k in 2:10)
{
  set.seed(2014)
  km <- kmeans(datos, k, nstart=10)
  print(km$size)
  silk <- silhouette(km$cluster, distancias)
  avg.width[k] <- summary(silk)$avg.width
}
#
names(avg.width) <- 1:10
round(avg.width[2:10], digits=4); plot(2:10, avg.width[2:10], xlab="k", ylab="avg.width", las=1)
#
# Según el índice, 3, 4 o 10 clases
# podrían constituir una buena solución.
#
k <- 4
set.seed(2014)
km <- kmeans(datos, k, nstart=10)
kmclusthtg <- km$cluster
silk <- silhouette(km$cluster, distancias)
plot(silk, col="blue")
#
# Las clases son bastante consistentes,
# salvo, quizás, la clase 3
#
# Representación gráfica de la solución
#
pdf("swisssolkm2.pdf")
pairs(datos, pch=21, cex=0.75, col="black", bg=km$cluster+1)
dev.off()
#
########################################################################
#
# Comparación de particiones con k=4
#
table(kmclusthmg,kmclusthtg)
```

```
#           kmclusthtg
#  hmg      1  2  3  4
#          1 16  0  0  0
#          2  0  8  0  8
#          3  7  0  5  0
#          4  0  0  3  0
#
# La decisión de cuál de las dos particiones es mejor
# debe tomar en cuenta otras consideraciones externas
#
###############################################################################
```

5.6 Representación gráfica de la solución

swisssolkm1.pdf

swisssolkm2.pdf