Instalación y carga de paquetes de R packages

1. Instalación de paquetes

- **cluster**: para utilizar algoritmos de clustering
- **factoextra**: para visualización de clusters (http://www.sthda.com/english/rpkgs/factoextra/)

```
# Install factoextra
install.packages("factoextra")
# Install cluster package
install.packages("cluster")
```

Si el paquete factoextra no se instala correctamente,

```
if(!require(devtools)) install.packages("devtools")
devtools::install_github("kassambara/factoextra")
```

Carga de paquetes

```
library("cluster")
library("factoextra")
```

2. Preparación de Datos

Estadísticas Básicas de datos

```
Summary()
```

Normalización de datos

```
Scale()
```

3. Método de particionado: K-means

K-means se puede realizar SOLO con datos donde todas las variables sean continuas.

La función standard de R que permite utilizar el método kmeans es kmeans() (en paquete stats):

```
kmeans(x, centers, iter.max = 10, nstart = 1)
```

Necesita

- **x**: matriz, data frame o vector
- **centers**: Número de clusters (k)
- **iter.max**: Máximo número de iteraciones
- **nstart**: Número de particiones aleatorias iniciales. Recomendado *nstart > 1*.

Retorna:

- **cluster**: Un vector de enteros indicando el cluster asignado a cada ejemplo
- **centers**: Una matriz de centroides
- **totss:** Error Total
- **withinss:** Un vector de dimensión k con la distancia intra cluster
- **within-cluster:** Un vector de dimensión k con la distancia inter cluster
- **tot.withinss: sum(withinss)**
- **betweenss: totss−tot.withinss**
- **size:** El número de observaciones en cada cluster

Como k-means comienza con k centroides seleccionados aleatorios, se recomienda utilizar la función set.seed() para que los resultados sean reproducibles.

Ejemplo:

Generar datos aleatoriamente

```
set.seed(123)
# Two-dimensional data format
df <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),matrix(rnorm(100,
mean = 1, sd = 0.3), ncol = 2))
colnames(df) <- c("x", "y")
head(df)
```

Ver el tamaño de los clusters:

```
df$size
```

Agrupar datos

```
set.seed(123)
km.res <- kmeans(df, 2, nstart = 25)
# Cluster number for each of the observations
km.res$cluster
```

Visualizar

```
plot(df, col = km.res$cluster, pch = 19, frame = FALSE,
     main = "K-means con k = 2")
points(km.res$centers, col = 1:2, pch = 8, cex = 3)
```

- Visualiza la agrupación para k=4
- Comprueba el efecto del parámetro nstart en los resultados. Recuerda que este parámetro inicia kmeans con nstart configuraciones distintas (nstart=20 inicia kmeans con 20 centroides distintas y se queda con la mejor).
- Comprueba lo que ocurre con la distancia intra cluster con distintos valores nstart.

4. Método de Particionado: PAM y CLARA

PAM se puede realizar SOLO con datos donde todas las variables sean continuas.

La función standard de R que permite utilizar el método PAM es pam() (en paquete cluster):

```
pam(x, k, diss, metric)
```

Necesita

- x**:** Una matriz donde cada fila corresponde a una observación y cada columna a una variable o una matriz de disimilitud**.**
- k**:** número de clusters
- diss: Si TRUE la matriz de entrada es una matriz de distancias
- métrica usada: distancia euclidea o manhattan

Retorna:

- **medoids**: Una matriz donde cada fila i representa el medoide del cluster i
- **id.med**: Índice de los medoides
- **clustering:** Número de cluster asignado

- **clusinfo:** Información de cada cluster: Número de ejemplos de cada cluster, distancia máxima y media, diámetro y separación
- **silinfo:** Información sobre silueta

La función standard de R que permite utilizar el método CLARA es clara() (en paquete cluster):

```
clara(x, k, samples=5)
```

Necesita

- x**:** Una matriz donde cada fila corresponde a una observación y cada columna a una variable o una matriz de disimilitud**.**
- k**:** número de clusters
- samples: Número de muestras obtenidas del conjunto de datos. Es recomendable tomar un número grande

Retorna:

- **sample:** Muestra seleccionada
- **medoids**: Una matriz donde cada fila i representa el medoide del cluster i
- **i.med**: Índice de los medoides
- **clustering:** Número de cluster asignado
- **clusinfo:** Información de cada cluster: Número de ejemplos de cada cluster, distancia máxima y media, diámetro y separación
- **silinfo:** Información sobre silueta

Ejemplo:

Lectura de datos

```
library("cluster")
# Leer datos
data("USArrests")
# Escalar los datos y calcular pam con k=4
pam.res <- pam(scale(USArrests), 4)
```

Extracción de medoides

```
pam.res$medoids
```

Asignación de clusters

```
head(pam.res$cluster)
```

5. Representación y Análisis de Clusters

Representación de resultados con la función *clusplot*()

```
clusplot(x, main = NULL, stand=TRUE, color = FALSE, labels=0)
```

Necesita

- x**:** Un objeto de clase partition, es decir la variable que retorna un algoritmo de clustering (PAM, CLARA)
- clus: Un vector que contiene el número de cluster asignado a cada observación.
- stand: Datos normalizados (TRUE)
- color: Los clusters están coloreados
- labels: Etiquetas de los puntos (valores 0 a 4)

Ejemplo

```
clusplot(pam.res, main = "Cluster plot, k = 4", color = TRUE)
```

También se puede utilizar la función *fviz_cluster*()

```
fviz_cluster(x, data = NULL, stand = TRUE, geom = c("point",
"text"),frame = TRUE, frame.type = "convex")
```

Necesita

- x**:** Un objeto de clase partition, es decir la variable que retorna un algoritmo de clustering (PAM, CLARA o KMEANS)
- data: Si el algoritmo es K-Means, datos de entrada al clustering. Innecesario en otro caso.
- stand: Datos normalizados (TRUE)
- geom: Especificación de la geometría utilizada, “point”, “text” o c(“point”, “text”)
- frame: Marco alrededor de clusters (TRUE)

Ejemplo

```
fviz_cluster(pam.res)
```

6. Métodos Jerárquicos

- Clustering Aglomerativo: función hclust (paquete stats) o agnes (paquete cluster)
- Clustering Divisivo: función diana (paquete cluster)

```
hclust(x, method="complete")
```

Necesita

- x: Matriz de distancias
- method: Método aglomerativo. Posibilidades: "Ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" o "centroid".

```
data("USArrests")
df <- na.omit(USArrests)
# Dissimilarity matrix
d <- dist(df, method = "euclidean")
# Hierarchical clustering using Ward's method
res.hc <- hclust(d, method = "ward.D2" )
# Plot the obtained dendrogram
plot(res.hc, cex = 0.6, hang = -1)
```

- Prueba como afectan las distintas distancias en el método de agrupamiento
- Prueba los distintos métodos de realizar los aglomerados.

```
# Agglomerative Nesting (Hierarchical Clustering)
agnes(x, metric = "euclidean", stand = FALSE, method = "average")
# DIvisive ANAlysis Clustering
diana(x, metric = "euclidean", stand = FALSE)
```

Necesitan

- x: Matriz de distancias
- metric: Distancia usada ("euclidean", "manhattan")
- stand: TRUE si se normaliza x

- method: Método aglomerativo. Posibilidades: “Ward.D”, “ward.D2”, “single”, “complete”, “average”, “mcquitty”, “median” o “centroid”.

Ejemplo

```
res.agnes <- agnes(df, method = "ward")
```

- Comprueba que valores retornan las funciones hclust(), agnes(), diana()

7. Representación Métodos Jerárquicos

- Representación de resultados con la función *pltree*()

```
pltree(x, main = NULL, cex)
```

Necesita

- x**:** Un objeto de clase partition, es decir la variable que retorna un algoritmo de clustering (PAM, CLARA)
- clus: Un vector que contiene el número de cluster asignado a cada observación.
- stand: Datos normalizados (TRUE)
- color: Los clusters están coloreados
- labels: Etiquetas de los puntos (valores 0 a 4)

Ejemplo

```
pltree(res.agnes, cex = 0.6, hang = -1, main = "Dendrograma  de
agnes")
```

- Generación de grupos a partir de un dendograma *cutree()*

```
cutree(x, k)
```

Necesita

- x: objeto de tipo árbol jerárquico
- k: Número de clusters

Ejemplo

```
grp <- cutree(res.agnes, k = 4)
```

En la variable grp se almacena el número de cluster asignado a cada ejemplo (en este caso un número entre 1 y 4).

Si queremos conocer los elementos de un cluster:

```
rownames(df)[grp == 1]
```

Nos da los nombres de las filas cuyo cluster según este método es el 1.

Por último, se pueden representar los grupos gráficamente

```
pltree(res.agnes, cex = 0.6, hang = -1, main = "Dendrograma  de
                                agnes")
            rect.hclust(res.hc, k = 4, border = 2:5)
```

- Representación de resultados con la función *fviz_cluster*() vista anteriormente

Tenemos que decirle cuales son los cluster

Ejemplo

```
fviz_cluster(list(data = df, cluster = grp))
```

8. Comparación de dendogramas

- Instalar el paquete *dendextend()*

```
library(dendextend)
```

- Carga de datos y generación de clusters con 2 métodos diferentes

```
data("USArrests")
df <- na.omit(USArrests)
```

```
#Construir matriz de distancias
d <- dist(df, method = "euclidean")

#Métodos de clustering
res.hc <- hclust(d, method = "ward.D2" )
res.agnes<-agnes(df, metric = "euclidean", stand = TRUE, method
= "average")
res.diana<-diana(df, metric = "euclidean", stand = TRUE)
```

- Crear una lista de dendogramas

```
den.hc<-as.dendrogram(res.hc)
den.diana<-as.dendrogram(res.diana)
den.agnes<-as.dendrogram(res.agnes)
dend_list <- dendlist(den.hc, den.diana)
```

- Uso de la función entanglement() para comparar dos dendogramas. Cuanto más próximo a 0 más se parecen.

```
tanglegram(den.hc, den.diana ,main = paste("entanglement =",
round(entanglement(dend_list), 2)))
```

- Diferencia relativa entre dendogramas con función *all.equal()*. Si son iguales devuelve TRUE. En caso contrario retorna la distancia relativa.

```
all.equal(den.hc,den.diana)
```

9. Validación de clusters: Silueta

```
Sil.pam<- silhouette(pam.res)
Sil.km <- silhouette(km.res$cluster, dist(df))

plot(silhouette(pam.res),  col = 2:5)
```

En caso de que algún ejemplo esté en el cluster incorrecto, se puede buscar cual es el cluster vecino más próximo

```
# Compute silhouette
sil <- silhouette(pam.res)[, 1:3]
# Objects with negative silhouette
neg_sil_index <- which(sil[, 'sil_width'] < 0)
sil[neg_sil_index, , drop = FALSE]
```

```
# Summary of silhouette analysis
si.sum <- summary(sil)
# Average silhouette width of each cluster
si.sum$clus.avg.widths
##         1         2         3
## 0.6363162 0.3473922 0.3933772
# The total average (mean of all individual silhouette widths)
si.sum$avg.width
## [1] 0.4599482
# The size of each clusters
si.sum$clus.sizes
## cl
##  1  2  3
## 50 47 53
```

10. Cálculo del número de clusters óptimo

Existen muchos métodos para intentar buscar el número cluster óptimo. Todos ellos se basan en aplicar el algoritmo de clustering con distintos valores de *k*.

Una de las posibilidades es calcular la suma de distancias intra cluster (parámetro *wss*), representar esta cantidad según los valores de *k* y elegir como óptimo el k que corresponda con un punto de inflexión.

```
fviz_nbclust(x, FUNcluster, method = c("silhouette", "wss"))
```

Necesita

- **x**: matriz, data frame o vector
- **FUNcluster**: Nombre del algoritmo de particionado, kmeans, pam, etc.
- **method**: Método para determinar el número de clusters

Ejemplo:

```
#kmeans con k entre 1 y 10.
set.seed(123)
fviz_nbclust(df, kmeans, method = "wss")
```

Otra posibilidad

Esta función computa un método de clustering con k clusters entre min.nc y max.nc, utilizando como distancia la especificada en distance.

```
NbClust(data = NULL, diss = NULL, distance = "euclidean", min.nc = 2,
max.nc = 15, method = NULL, index = "all")
```

Necesita

- data: **matriz**
- diss: matriz de disimilitud. Por defecto NULL. Si no es NULL, entonces distance tiene que ser NULL
- distance: distancia usada para calcular la matriz de disimilitud. **Toma los siguientes valores: "euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski" o "NULL". Por defecto, "euclidean"**
- min.nc: Número mínimo de clusters
- max.nc: Número máximo de clusters
- method: Método de cluster usado: **"ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median", "centroid", "kmeans".**
- Index: Índice de validación usado: **"kl", "ch", "hartigan", "ccc", "scott", "marriot", "trcovw", "tracew", "friedman", "rubin", "cindex", "db", "silhouette", "duda", "pseudot2", "beale", "ratkowsky", "ball", "ptbiserial", "gap", "frey", "mcclain", "gamma", "gplus", "tau", "dunn", "hubert", "sdindex", "dindex", "sdbw", "all" (todos menos GAP, Gamma, Gplus and Tau), "alllong" (todos.**

Ejemplo:

```
res.nbclust <- NbClust(df, distance = "euclidean", min.nc = 2, max.nc
= 10, method = "complete", index ="all")
```

- ¿Qué valores retorna?
- ¿Cuál es el número óptimo de clusters sugerido?
- Comprueba lo que ocurre cuando el método es "silhouette". ¿Propone el mismo número de clusters cómo óptimo?
- Realiza la agrupación con el número de clusters óptimo

```
# k-means clustering con k = 2
set.seed(123)
km.res <- kmeans(df, 2, nstart = 25)
```

```
print(km.res)
```

Calculamos la media de cada una de las variables en el cluster (los centroides reales)

```
aggregate(df,by=list(cluster=km.res$cluster),mean)
```