Clustering y heatmaps: aprendizaje no supervisado

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Septiembre, 2017

Tabla de contenidos

Versión PDF: https://github.com/JoaquinAmatRodrigo/Estadistica-con-R

Introducción

El término *clustering* hace referencia a un amplio abanico de técnicas *unsupervised* cuya finalidad es encontrar patrones o grupos (*clusters*) dentro de un conjunto de observaciones. Las particiones se establecen de forma que las observaciones que están dentro de un mismo grupo son similares entre ellas y distintas a las observaciones de otros grupos. Se trata de un método *unsupervised,* ya que el proceso ignora la variable respuesta que indica a que grupo pertenece realmente cada observación (si es que existe tal variable). Esta característica diferencia al *clustering* de las técnicas estadísticas conocidas como análisis discriminante, que emplean un set de entrenamiento en el que se conoce la verdadera clasificación.

Dada la popularidad del *clustering* en disciplinas muy distintas (genómica, marketing...), se han desarrollado multitud de variantes y adaptaciones de sus métodos y algoritmos. Pueden diferenciarse tres grupos:

- *Partitioning Clustering*: Este tipo de algoritmos requieren que el usuario especifique de antemano el número de *clusters* que se van a crear (*K-means*, *K-medoids*, *CLARA*).
- *Hierarchical Clustering*: Este tipo de algoritmos no requieren que el usuario especifique de antemano el número de *clusters*. (*agglomerative clustering*, *divisive clusterig*).
- Métodos que combinan o modifican los anteriores (*hierarchical K-means*, *fuzzy clustering*, *model based clustering* y *density based clustering*).

En el entorno de programación R existen múltiples funciones y paquetes desarrollados para la realización de *clustering*. En este capítulo se emplean los siguientes:

- `stats`: contiene las funciones `dist()` para calcular distancias y `kmeans()`, `hclust()`, `cuttree()` y `plot.hclust()` para la identificación de *clusters*.
- `cluster`, `mclust`: algoritmos para *clustering*.
- `factoextra`: extensión basada en ggplot2 para crear visualizaciones elegantes de los resultados de *clustering*.
- `dendextend`: extensión para la customización de dendrogramas.

Medidas de distancia

Todos los métodos de *clustering* tienen una cosa en común, para poder llevar a cabo las agrupaciones necesitan definir y cuantificar la similitud entre las observaciones. El término distancia se emplea dentro del contexto del *clustering* como cuantificación de la similitud o diferencia entre observaciones. Si se representan las observaciones en un espacio p dimensional, siendo p el número de variables asociadas a cada observación, cuando más se asemejen dos observaciones más próximas estarán, de ahí que se emplee el término distancia. La característica que hace del *clustering* un método adaptable a escenarios muy diversos es que puede emplear cualquier tipo de distancia, lo que permite al investigador escoger la más adecuada para el estudio en cuestión. A continuación, se describen algunas de las más utilizadas.

Distancia euclídea

La distancia euclídea entre dos puntos p y q se define como la longitud del segmento que une ambos puntos. En coordenadas cartesianas, la distancia euclídea se calcula empleando el teorema de Pitágoras. Por ejemplo, en un espacio de dos dimensiones en el que cada punto está definido por las coordenadas (x, y), la distancia euclídea entre p y q viene dada por la ecuación:

$$d_{euc}(p,q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

Esta ecuación puede generalizarse para un espacio euclídeo n-dimensional donde cada punto está definido por un vector de n coordenadas: $p = (p_1, p_2, p_3, \ldots, p_n)$ y $q = (q_1, q_2, q_3, \ldots, q_n)$.

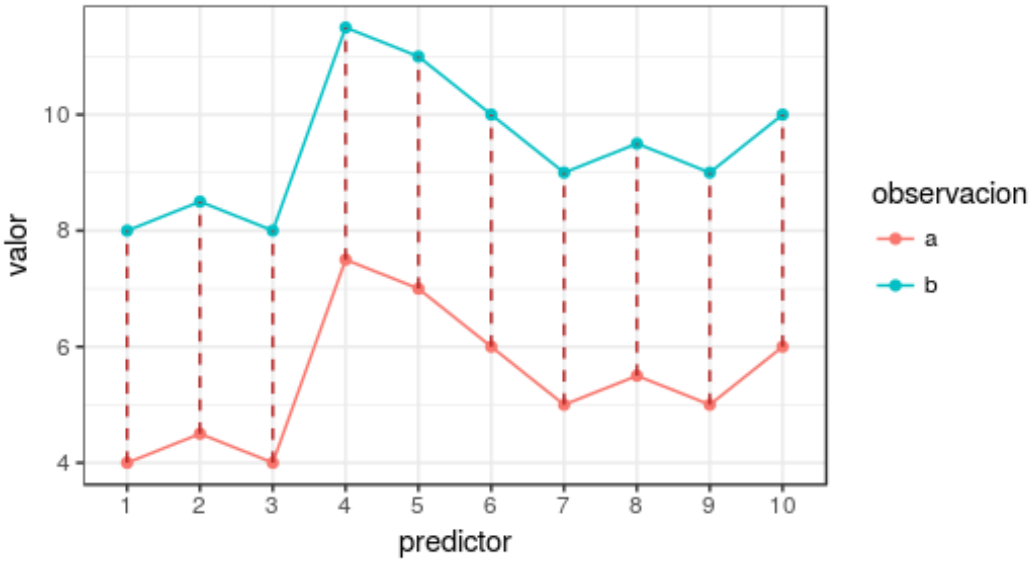
$$d_{euc}(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} =$$

$$\sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Una forma de dar mayor peso a aquellas observaciones que están más alejadas es emplear la distancia euclídea al cuadrado. En el caso del *clustering*, donde se busca agrupar observaciones que minimicen la distancia, esto se traduce en una mayor influencia de aquellas observaciones que están más distantes.

La siguiente imagen muestra el perfil de dos observaciones definidas por 10 variables (espacio con 10 dimensiones).

```
library(ggplot2)
observacion_a <- c(4, 4.5, 4, 7.5, 7, 6, 5, 5.5, 5, 6)
observacion_b <- c(4, 4.5, 4, 7.5, 7, 6, 5, 5.5, 5, 6) + 4
datos <- data.frame(observacion = rep(c("a", "b"), each = 10),
                    valor = c(observacion_a, observacion_b),
                    predictor = 1:10)
ggplot(data = datos, aes(x = as.factor(predictor), y = valor,
                         colour = observacion)) +
  geom_path(aes(group = observacion)) +
  geom_point() +
  geom_line(aes(group = predictor), colour = "firebrick", linetype = "dashed") +
  labs(x = "predictor") +
  theme_bw()
```



La distancia euclídea entre las dos observaciones equivale a la raíz cuadrada de la suma de las longitudes de los segmentos rojos que unen cada par de puntos. Tiene en cuenta por lo tanto el desplazamiento individual de cada una de las variables.

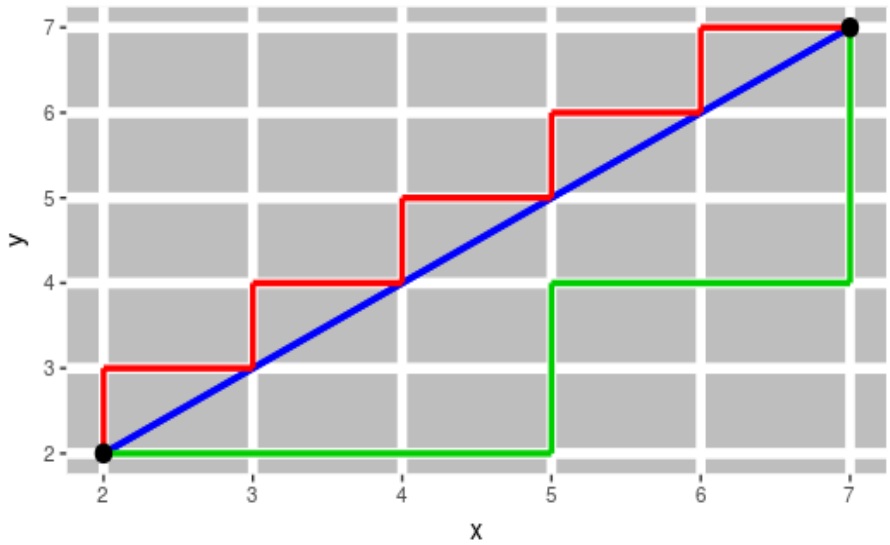
Distancia de Manhattan

La distancia de Manhattan, también conocida como *taxicab metric*, *rectilinear distance* o *L1 distance*, define la distancia entre dos puntos p y q como el sumatorio de las diferencias absolutas entre cada dimensión. Esta medida se ve menos afectada por *outliers* (es más robusta) que la distancia euclídea debido a que no eleva al cuadrado las diferencias.

$$d_{man}(p,q) = \sum_{i=1}^{n} |\,(p_i - q_i)|$$

La siguiente imagen muestra una comparación entre la distancia euclídea (segmento azul) y la distancia de manhattan (segmento rojo y verde) en un espacio bidimensional. Existen múltiples caminos para unir dos puntos con el mismo valor de distancia de manhattan, ya que su valor es igual al desplazamiento total en cada una de las dimensiones.

```
datos <- data.frame(observacion = c("a", "b"), x = c(2,7), y = c(2,7))
manhattan <- data.frame(
              x = rep(2:6, each = 2),
              y = rep(2:6, each = 2) + rep(c(0,1), 5),
              xend = rep(2:6, each = 2) + rep(c(0,1), 5),
              yend = rep(3:7, each = 2))

manhattan_2 <- data.frame(
                x = c(2, 5, 5, 7),
                y = c(2, 2, 4, 4),
                xend = c(5, 5, 7, 7),
                yend = c(2, 4, 4, 7))

ggplot(data = datos, aes(x = x, y = y)) +
geom_segment(aes(x = 2, y = 2, xend = 7, yend = 7), color = "blue", size = 1.2) +
geom_segment(data = manhattan, aes(x = x, y = y, xend = xend, yend = yend),
             color = "red", size = 1.2) +
geom_segment(data = manhattan_2, aes(x = x, y = y, xend = xend, yend = yend),
             color = "green3", size = 1.2) +
geom_point(size = 3) +
theme(panel.grid.minor = element_blank(),
      panel.grid.major = element_line(size = 2),
      panel.background = element_rect(fill = "gray",
                                      colour = "white",
                                      size = 0.5, linetype = "solid"))
```



Correlación

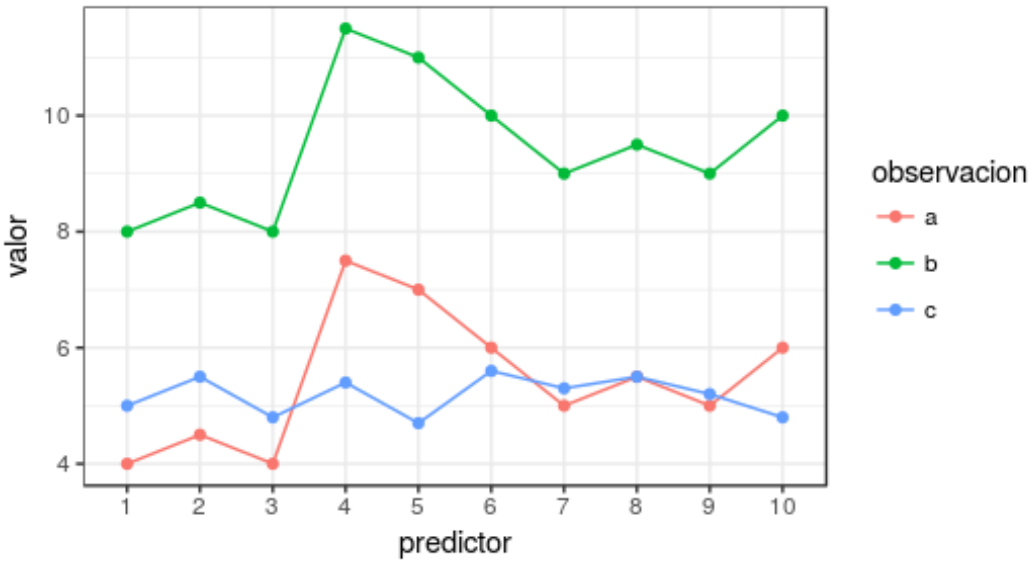
La correlación es una medida de distancia muy útil cuando la definición de similitud se hace en términos de patrón o forma y no de desplazamiento o magnitud. ¿Qué quiere decir esto? En la imagen del apartado de la distancia euclídea, las dos observaciones tienen exactamente el mismo patrón, la única diferencia es que una de ellas está desplazada 4 unidades por encima de la otra. Si se emplea como medida de similitud 1 menos el valor de la correlación, ambas observaciones se consideran idénticas (su distancia es 0).

$$d_{cor}(p, q) = 1 - correlacion(p, q)$$

donde la correlación puede ser de distintos tipos (*Pearson*, *Spearman*, *Kendall*...).

En la siguiente imagen se muestra el perfil de 3 observaciones. Acorde a la distancia euclídea, las observaciones *b* y *c* son las más similares, mientras que acorde a la correlación de Pearson, las observaciones más similares son *a* y *b*.

```
library(ggplot2)
observacion_a <- c(4, 4.5, 4, 7.5, 7, 6, 5, 5.5, 5, 6)
observacion_b <- c(4, 4.5, 4, 7.5, 7, 6, 5, 5.5, 5, 6) + 4
observacion_c <- c(5, 5.5, 4.8, 5.4, 4.7, 5.6, 5.3, 5.5, 5.2, 4.8)
datos <- data.frame(observacion = rep(c("a", "b", "c"), each = 10),
                    valor = c(observacion_a, observacion_b, observacion_c),
                    predictor = 1:10)
ggplot(data = datos, aes(x = as.factor(predictor), y = valor,
                         colour = observacion)) +
geom_path(aes(group = observacion)) +
geom_point() +
labs(x = "predictor") +
theme_bw()
```



```
dist(x = rbind(observacion_a, observacion_b, observacion_c), method = "euclidean")
```

```
##               observacion_a observacion_b
## observacion_b      12.64911
## observacion_c       3.75100      13.98821
```

```
1 - cor(x = cbind(observacion_a, observacion_b, observacion_c), method = "pearson")
```

```
##               observacion_a observacion_b observacion_c
## observacion_a     0.0000000     0.0000000     0.9466303
## observacion_b     0.0000000     0.0000000     0.9466303
## observacion_c     0.9466303     0.9466303     0.0000000
```

Este ejemplo pone de manifiesto que no existe una única medida de distancia que sea mejor que las demás, sino que, dependiendo del contexto, una será más adecuada que otra. Para información más detallada sobre correlación consultar Correlación lineal y Regresión lineal simple.

Jackknife correlation

El coeficiente de correlación de *Pearson* resulta efectivo en ámbitos muy diversos. Sin embargo, tiene la desventaja de no ser robusto frente a *outliers* a pesar de que se cumpla la condición de normalidad. Si dos variables tienen un pico o un valle común en una única observación, por ejemplo, por un error de lectura, la correlación va a estar dominada por este registro a pesar de que entre las dos variables no haya correlación real alguna. Lo mismo puede ocurrir en la dirección opuesta. Si dos variables están altamente correlacionadas excepto para una observación en la que los valores son muy dispares, entonces la correlación existente quedará enmascarada. Una forma de evitarlo es recurrir a la *Jackknife correlation*, que consiste en calcular todos los posibles coeficientes de correlación entre dos variables si se excluye cada vez una de las observaciones. El promedio de todas las *Jackknife correlations* calculadas atenuará en cierta medida el efecto del *outlier*.

$$\overline{\theta}_{(A,B)} = Promedio\ Jackknife\ correlation(A,B) = \frac{1}{n}\sum_{i=1}^{n}\hat{r}_i$$

donde n es el número de observaciones y $\hat{r}_i$ es el coeficiente de correlación entre las variables A y B, habiendo excluido la observación i.

Además del promedio, se puede estimar su error estándar (SE) y así obtener intervalos de confianza para la *Jackknife correlation* y su correspondiente *p-value*.

$$SE = \sqrt{\frac{n-1}{n}\sum_{i=1}^{n}(\hat{r}_i - \overline{\theta})^2}$$

Intervalo de confianza del 95% ($Z = 1.96$)

$$Promedio\ Jackknife\ correlation(A,B)\ \pm\ 1.96 * SE$$

$$\overline{\theta} - 1.96\sqrt{\frac{n-1}{n}\sum_{i=1}^{n}(\hat{r}_i - \overline{\theta})^2}\ ,\ \ \overline{\theta} + 1.96\sqrt{\frac{n-1}{n}\sum_{i=1}^{n}(\hat{r}_i - \overline{\theta})^2}$$

P-value para la hipótesis nula de que $\overline{\theta} = 0$:

$$Z_{calculada} = \frac{\overline{\theta} - H_0}{SE} = \frac{\overline{\theta} - 0}{\sqrt{\frac{n-1}{n}\sum_{i=1}^{n}(\hat{r}_i - \overline{\theta})^2}}$$

$$p_{value} = P(Z > Z_{calculada})$$

Cuando se emplea este método es conveniente calcular la diferencia entre el valor de correlación obtenido por *Jackknife correlation* ($\overline{\theta}$) y el que se obtiene si se emplean todas las observaciones ($\overline{r}$). A esta diferencia se le conoce como *Bias*. Su magnitud es un indicativo de cuanto está influenciada la estimación de la correlación entre dos variables debido a un valor atípico u *outlier*.

$$Bias = (n-1) * (\overline{\theta} - \hat{r})$$

Si se calcula la diferencia entre cada correlación ($\hat{r}_i$) estimada en el proceso de *Jackknife* y el valor de correlación ($\hat{r}$) obtenido si se emplean todas las observaciones, se puede identificar que observaciones son más influyentes.

Cuando el estudio requiere minimizar al máximo la presencia de falsos positivos, a pesar de que se incremente la de falsos negativos, se puede seleccionar como valor de correlación el menor de entre todos los calculados en el proceso de *Jackknife*.

$$Correlacion = min\{\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_n\}$$

A pesar de que el método de *Jackknife* permite aumentar la robustez de la correlación de *Pearson*, si los *outliers* son muy extremos su influencia seguirá siendo notable.

Simple matching coefficient

Cuando las variables con las que se pretende determinar la similitud entre observaciones son de tipo binario, a pesar de que es posible codificarlas de forma numérica como 1 ó 0, no tiene sentido aplicar operaciones aritméticas sobre ellas (media, suma…). Por ejemplo, si la variable sexo se codifica como 1 para mujer y 0 para hombre, carece de significado decir que la media de la variable sexo en un determinado set de datos es 0.5. En situaciones como esta no se pueden emplear medidas de similitud basadas en distancia euclídea, manhattan, correlación…

Dado dos objetos A y B, cada uno con n atributos binarios, *simple matching coefficient (SMC)* define la similitud entre ellos como:

$$SMC = \frac{numero\ coincidencias}{numero\ total\ de\ atributos} = \frac{M_{00} + M11}{M_{00} + M_{01} + M_{10} + M_{11}}$$

donde M_{00} y M_{11} son el número de variables para las que ambas observaciones tienen el mismo valor (ambas 0 o ambas 1), y M_{01} y M_{10} el número de variables que no coinciden. El valor de distancia *simple matching distance (SMD)* se corresponde con $1 - SMC$.

Índice Jaccard

El índice *Jaccard* o coeficiente de correlación *Jaccard* es similar al *simple matching coefficient (SMC)*. La diferencia radica en que el *SMC* tiene el término M_{00} en el numerador y denominador, mientras que el índice de *Jaccard* no. Esto significa que *SMC* considera como coincidencias tanto si el atributo está presente en ambos sets como si el atributo no está en ninguno de los sets, mientras que *Jaccard* solo cuenta como coincidencias cuando el atributo está presente en ambos sets.

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

o en términos matemáticos de sets:

$$J(A, B) = \frac{A \cap B}{A \cup B}$$

La distancia de *Jaccard* $(1 - J)$ supera a la *simple matching distance* en aquellas situaciones en las que la coincidencia de ausencia no aporta información. Para ilustrar este hecho, supóngase que se quiere cuantificar la similitud entre dos clientes de un supermercado en base a los artículos comprados. Es de esperar que cada cliente solo adquiera unos pocos artículos de los muchos disponibles, por lo que el número de artículos no comprados por ninguno (M_{00}) será muy alto. Como la distancia de *Jaccard* ignora las coincidencias de tipo

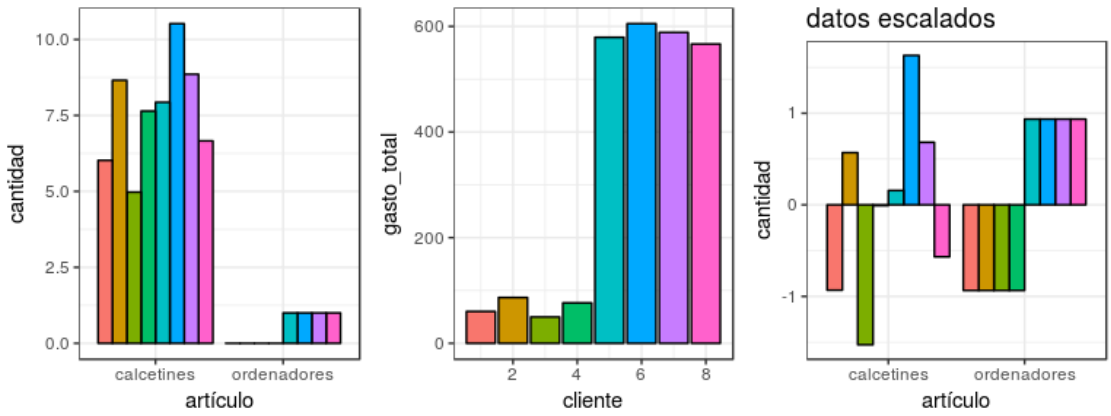
M_{00}, el grado de similitud dependerá únicamente de las coincidencias entre los artículos comprados.

Escala de las variables

Al igual que en otros métodos estadísticos (*PCA*, *ridge regression*, *lasso*...), la escala en la que se miden las variables y la magnitud de su varianza pueden afectar en gran medida a los resultados obtenidos por *clustering*. Si una variable tiene una escala mucho mayor que el resto, determinará en gran medida el valor de distancia/similitud obtenido al comparar las observaciones, dirigiendo así la agrupación final. Escalar y centrar las variables de forma que todas ellas tengan media 0 y desviación estándar 1 antes de calcular la matriz de distancias asegura que todas las variables tengan el mismo peso cuando se realice el *clustering*.

$$\frac{x_i - mean(x)}{sd(x)}$$

Para ilustrar este hecho, supóngase que una tienda online quiere clasificar a los compradores en función de los artículos que adquieren, por ejemplo, calcetines y ordenadores. La siguiente imagen muestra el número de artículos comprados por 8 clientes a lo largo de un año, junto con el gasto total de cada uno.



Si se intenta agrupar a los clientes por el número de artículos comprados, dado que los calcetines se compran con mucha más frecuencia que los ordenadores, van a tener más peso al crear los *clusters*. Por el contrario, si la agrupación se hace en base al gasto total de los clientes, como los ordenadores son mucho más caros, van a determinar en gran medida la clasificación. Escalando y centrando las variables se consigue igualar la influencia de calcetines y ordenadores.

Cabe destacar que, si se aplica la estandarización descrita, existe una relación entre la distancia euclídea y la correlación de Pearson que hace que los resultados obtenidos por *clustering* en ambos casos sean equivalentes.

$$d_{euc}(p, q\ estandarizados) = \sqrt{2n(1 - cor(p, q))}$$

Ejemplo

El set de datos `USArrests` *contiene información sobre el número de delitos (asaltos, asesinatos y secuestros) junto con el porcentaje de población urbana para cada uno de los 50 estados de USA. Empleando estas variables se pretende calcular una matriz de distancias que permita identificar los estados más similares*

Dos de las funciones en `R` que permiten calcular matrices de distancia empleando variables numéricas son `scale()` y `get_dist()`. Esta última incluye más tipos de distancias.

```
data(USArrests)

# Se escalan las variables
datos <- scale(USArrests)

# Distancia euclídea
mat_dist <- dist(x = datos, method = "euclidean")
round(as.matrix(mat_dist)[1:5, 1:5], 2)
```

```
##            Alabama Alaska Arizona Arkansas California
## Alabama       0.00   2.70    2.29     1.29       3.26
## Alaska        2.70   0.00    2.70     2.83       3.01
## Arizona       2.29   2.70    0.00     2.72       1.31
## Arkansas      1.29   2.83    2.72     0.00       3.76
## California    3.26   3.01    1.31     3.76       0.00
```

```
# Distancia basada en la correlación de pearson
library(factoextra)
mat_dist <- get_dist(x = datos, method = "pearson")
round(as.matrix(mat_dist)[1:5, 1:5], 2)
```

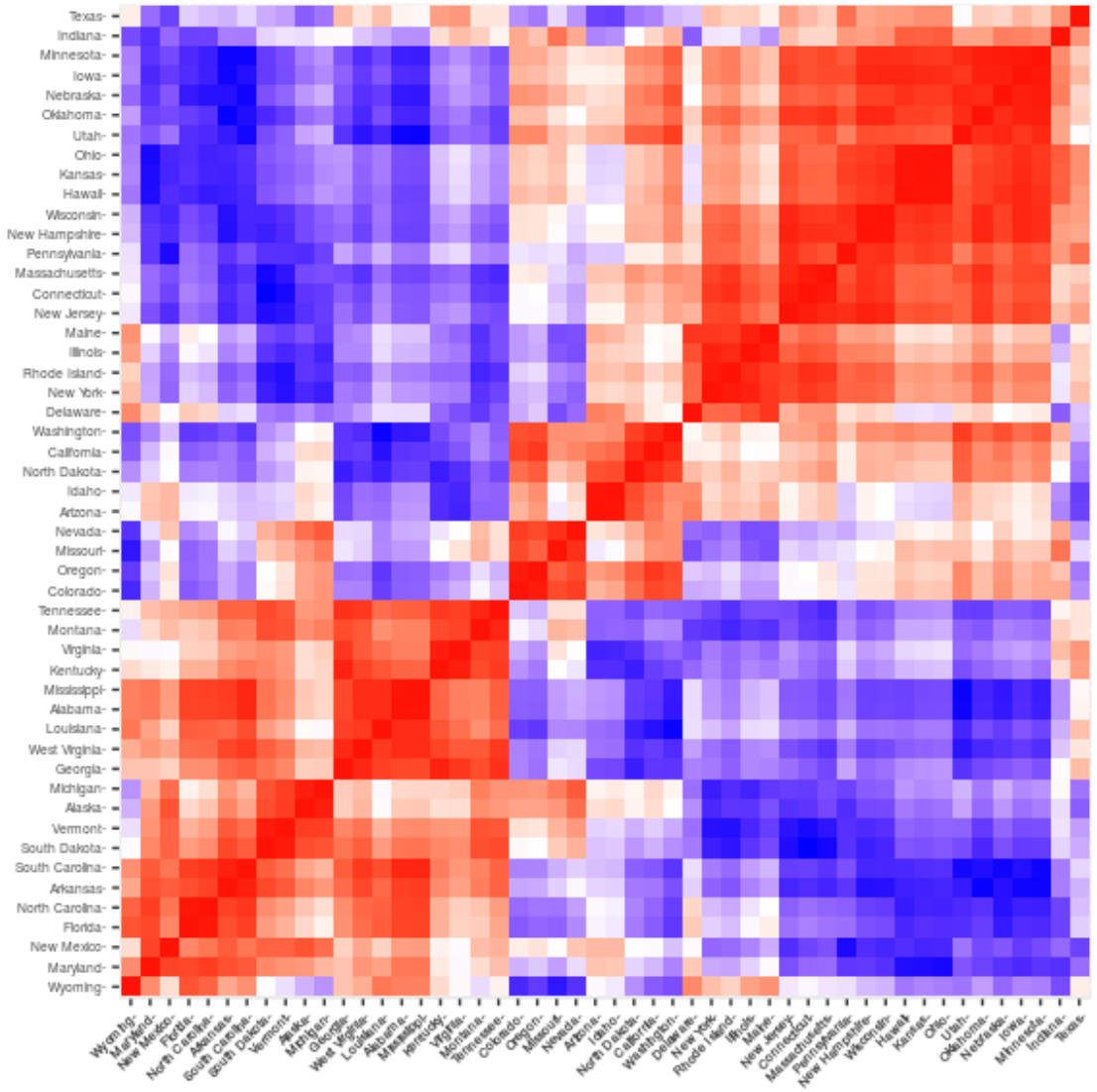
```
##            Alabama Alaska Arizona Arkansas California
## Alabama       0.00   0.71    1.45     0.09       1.87
## Alaska        0.71   0.00    0.83     0.37       0.81
## Arizona       1.45   0.83    0.00     1.18       0.29
## Arkansas      0.09   0.37    1.18     0.00       1.59
## California    1.87   0.81    0.29     1.59       0.00
```

```
# Esto es equivalente a 1 - correlación pearson
round(1 - cor(x = t(datos), method = "pearson"), 2)[1:5, 1:5]
```

```
##            Alabama Alaska Arizona Arkansas California
## Alabama       0.00   0.71    1.45     0.09       1.87
## Alaska        0.71   0.00    0.83     0.37       0.81
## Arizona       1.45   0.83    0.00     1.18       0.29
## Arkansas      0.09   0.37    1.18     0.00       1.59
## California    1.87   0.81    0.29     1.59       0.00
```

La función `fviz_dist` del paquete `factoextra` resulta muy útil para generar representaciones gráficas (*heatmap*) de matrices de distancia.

```
fviz_dist(dist.obj = mat_dist, lab_size = 5) +
  theme(legend.position = "none")
```



K-means clustering

Idea intuitiva

El método *K-means clustering* (MacQueen, 1967) agrupa las observaciones en *K clusters* distintos, donde el número *K* lo determina el analista. *K-means clustering* encuentra los *K* mejores *clusters*, entendiendo como *mejor cluster* aquel cuya varianza interna (*intra-cluster variation*) sea lo más pequeña posible. Se trata por lo tanto de un problema de optimización, en el que se reparten las observaciones en *k clusters* de forma que la suma de las varianzas internas de todos ellos sea lo menor posible. Para poder solucionar este problema es necesario definir un modo de cuantificar la varianza interna.

Considérense C_1,..., C_K como los sets formados por los índices de las observaciones de cada uno de los *clusters*. Por ejemplo, el set C_1 contiene los índices de las observaciones agrupadas en el *cluster* 1. La nomenclatura empleada para indicar que la observación i pertenece al *cluster* k es: $i \in C_k$. Todos los sets satisfacen dos propiedades:

- $C_1 \cup C_2 \cup \ldots \cup C_K = \{1, \ldots, n\}$. Significa que toda observación pertenece al menos a uno de los *K clusters*.
- $C_k \cap C_{k'} = \emptyset$ para todo $k \neq k'$. Implica que los *clusters* no solapan, ninguna observación pertenece a más de un *cluster* a la vez.

Dos de las medidas más comúnmente empleadas definen la varianza interna de un *cluster* ($W(C_k)$) como:

- La suma de las distancias euclídeas al cuadrado entre cada observación (x_i) y el centroide (μ) de su *cluster*. Esto equivale a la suma de cuadrados internos del *cluster*.

$$W(C_k) = \sum_{x_i, \in C_k} (x_i - \mu_k)^2$$

- La suma de las distancias euclídeas al cuadrado entre todos los pares de observaciones que forman el *cluster*, dividida entre el número de observaciones del *cluster*.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

Minimizar la suma total de varianza interna $\sum_{k=1}^{k} W(C_k)$ de forma exacta es un proceso muy complejo debido a la inmensa cantidad de formas en las que n observaciones se pueden

dividir en *k* grupos. Sin embargo, es posible obtener una solución que, aun no siendo la mejor de entre todas las posibles, es muy buena (óptimo local). El algoritmo empleado para ello es:

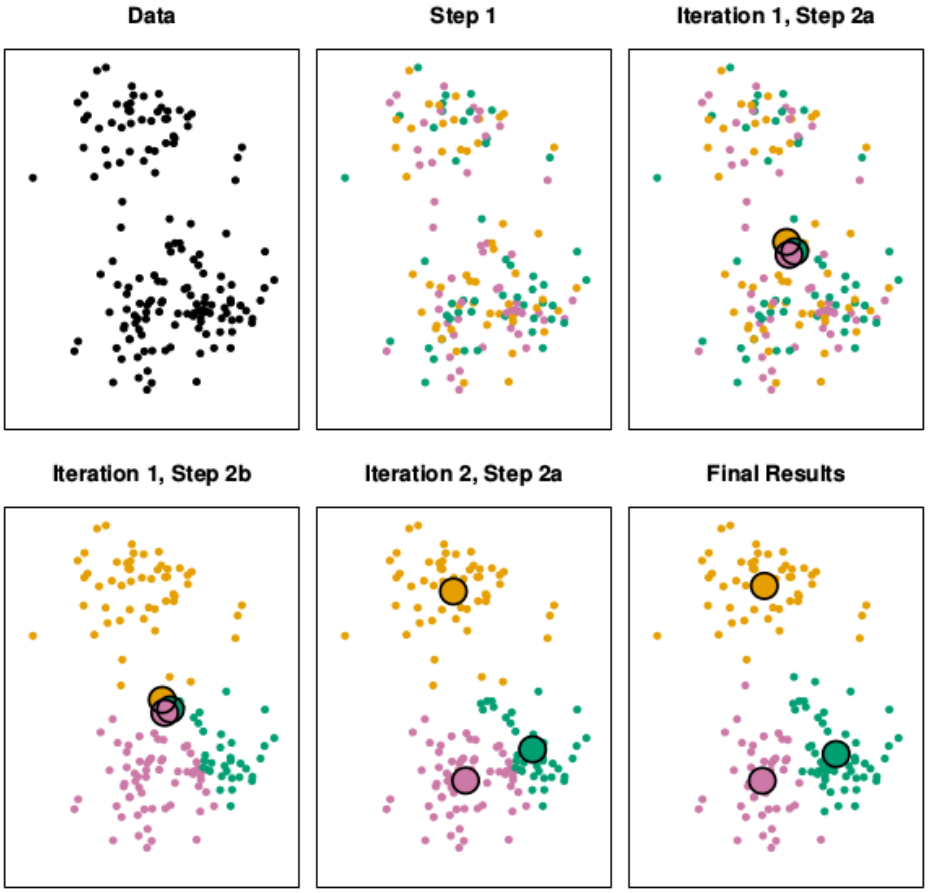
1. Asignar aleatoriamente un número entre 1 y *K* a cada observación. Esto sirve como asignación inicial aleatoria de las observaciones a los *clusters*.

2. Iterar los siguientes pasos hasta que la asignación de las observaciones a los *clusters* no cambie o se alcance un número máximo de iteraciones establecido por el usuario.

    a. Para cada uno de los *clusters* calcular su centroide. Entendiendo por centroide la posición definida por la media de cada una de las dimensiones (variables) de las observaciones que forman el *cluster*. Aunque no es siempre equivalente, puede entenderse como el centro de gravedad.

    b. Asignar cada observación al *cluster* cuyo centroide está más próximo.

Este algoritmo garantiza que en cada paso se reduzca la intra-varianza total de los *clusters* hasta alcanzar el *óptimo local*. La siguiente imagen muestra cómo van cambiando las asignaciones de las observaciones a medida que se ejecuta cada paso del algoritmo.



Otra forma de implementar el algoritmo de *K-means clustering* es la siguiente:

1. Especificar el número *K* de *clusters* que se quieren crear.
2. Seleccionar de forma aleatoria *k* observaciones del set de datos como centroides iniciales.
3. Asignar cada una de las observaciones al centroide más cercano.
4. Para cada uno de los *K clusters* recalcular su centroide.
5. Repetir los pasos 3 y 4 hasta que las asignaciones no cambien o se alcance el número máximo de iteraciones establecido.

Debido a que el algoritmo de *K-means* no evalúa todas las posibles distribuciones de las observaciones sino solo parte de ellas, los resultados obtenidos dependen de la asignación aleatoria inicial (paso 1). Por esta razón es importante ejecutar el algoritmo varias veces (20-50), cada una con una asignación aleatoria inicial distinta, y seleccionar aquella que haya conseguido un menor valor de varianza total.

Ventajas y desventajas

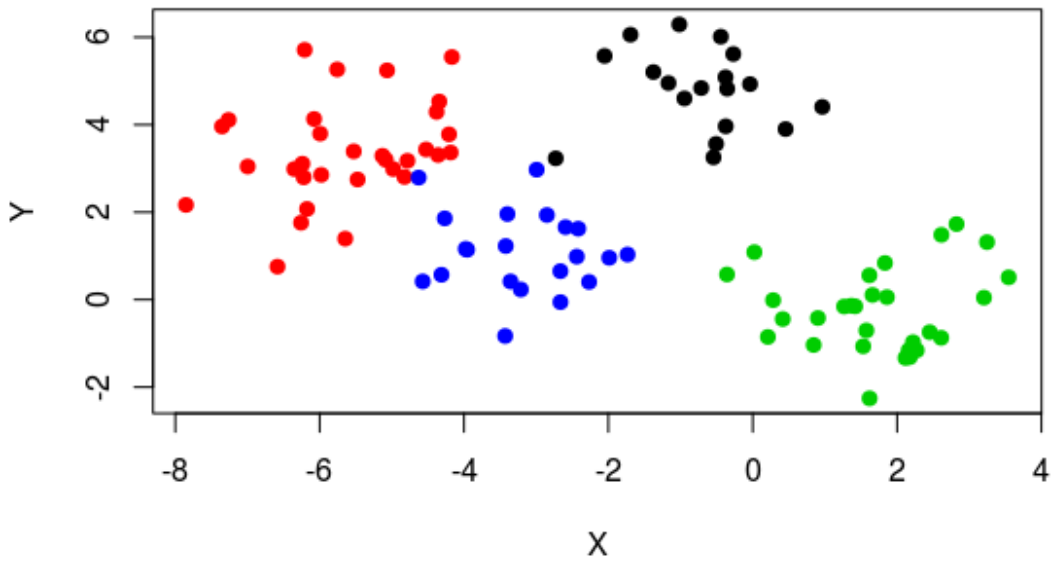
K-means es uno de los métodos de *clustering* más utilizados. Destaca por la sencillez y velocidad de su algoritmo, sin embargo, presenta una serie de limitaciones que se deben tener en cuenta.

- Requiere que se indique de antemano el número de *clusters* que se van a crear. Esto puede ser complicado si no se dispone de información adicional sobre los datos con los que se trabaja. Una posible solución es aplicar el algoritmo para un rango de valores *k* y evaluar con cual se consiguen mejores resultados, por ejemplo, menor suma total de varianza interna.
- Las agrupaciones resultantes pueden variar dependiendo de la asignación aleatoria inicial de los centroides. Para minimizar este problema se recomienda repetir el proceso de *clustering* entre 20 - 50 veces y seleccionar como resultado definitivo el que tenga menor suma total de varianza interna. Aun así, no se garantiza que para un mismo set de datos los resultados sean exactamente iguales.
- Presenta problemas de robustez frente a *outliers*. La única solución es excluirlos o recurrir a otros métodos de *clustering* más robustos como *K-medoids (PAM)*.

Ejemplo

Los siguientes datos simulados contienen observaciones que pertenecen a cuatro grupos distintos. Se pretende aplicar K-means-clustering con el fin de identificarlos.

```
set.seed(101)
# Se simulan datos aleatorios con dos dimensiones
datos <- matrix(rnorm(n = 100*2), nrow = 100, ncol = 2,
                dimnames = list(NULL,c("x", "y")))

# Se determina la media que va a tener cada grupo en cada una de las dos
# dimensiones. En total 2*4 medias. Este valor se va a utilizar para
# separar cada grupo de los demás.

media_grupos <- matrix(rnorm(n = 8, mean = 0, sd = 4), nrow = 4, ncol = 2,
                       dimnames = list(NULL, c("media_x", "media_y")))
media_grupos <- cbind(grupo = 1:4, media_grupos)

# Se genera un vector que asigne aleatoriamente cada observación a uno de
# los 4 grupos

grupo <- sample(x = 1:4, size = 100, replace = TRUE)
datos <- cbind(datos, grupo)

# Se incrementa el valor de cada observación con la media correspondiente al
# grupo asignado.

datos <- merge(datos, media_grupos, by = "grupo")
datos[,"x"] <- datos[, "x"] + datos[, "media_x"]
datos[,"y"] <- datos[, "y"] + datos[, "media_y"]

plot(x = datos[,"x"], y = datos[,"y"], col = datos[,"grupo"], pch = 19,
     xlab = "X", ylab = "Y")
```



La función `kmeans()` del paquete `stats` realiza *K-mean-clustering*. Entre sus argumentos destacan: `centers`, que determina el número *K* de *clusters* que se van a generar y `nstart`, que determina el número de veces que se va a repetir el proceso, cada vez con una asignación aleatoria inicial distinta. Es recomendable que este último valor sea alto, entre 20-50, para no obtener resultados malos debido a una iniciación poco afortunada del proceso.

Como los datos se han simulado considerando que todas las dimensiones tienen aproximadamente la misma magnitud, no es necesario escalarlos ni centrarlos.

```
set.seed(101)
km_clusters <- kmeans(x = datos[, c("x", "y")], centers = 4, nstart = 50)
km_clusters
```

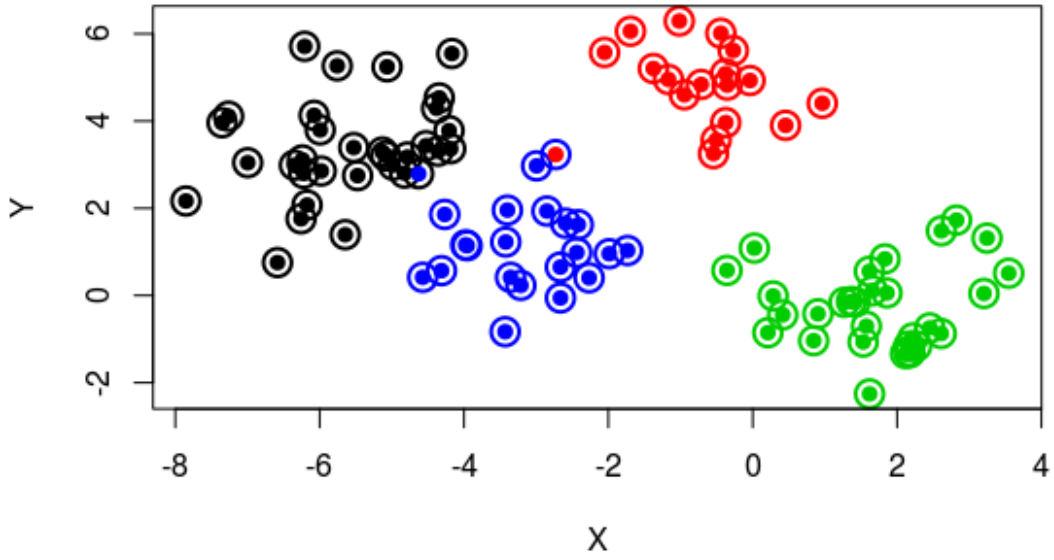
```
## K-means clustering with 4 clusters of sizes 32, 17, 30, 21
##
## Cluster means:
##            x          y
## 1 -5.5818142  3.3684991
## 2 -0.6148368  4.8861032
## 3  1.7226318 -0.2584919
## 4 -3.1068542  1.1213302
##
## Clustering vector:
##   [1] 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 1 4 4 4 4 4 4 4 4
##
```

```
## Within cluster sum of squares by cluster:
## [1] 71.98228 21.04952 54.48008 30.82790
##  (between_SS / total_SS =  87.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

El objeto devuelto por la función `kmeans()` contiene entre otros datos: la media de cada una de las variables para cada *cluster*, un vector indicando a que *cluster* se ha asignado cada observación, la suma de cuadrados interna de cada *cluster* y el ratio de la suma de cuadrados entre *clusters* y la suma de cuadrados totales. Este último término es equivalente al R^2 de los modelos de regresión, indica el porcentaje de varianza explicada por el modelo respecto al total de varianza observada. Puede utilizarse para evaluar el *clustering* obtenido, pero, al igual que ocurre con R^2 al incrementar el número de predictores, el ratio *between_SS / total_SS* aumenta con el número de *clusters* creados. Esto último se debe de tener en cuenta para evitar problemas de *overfitting*.

Al tratarse de una simulación, se conoce el número real de grupos (4) y a cuál de ellos pertenece cada observación. Esto no sucede en la mayoría de casos prácticos, pero es útil para evaluar cómo de bueno es el método de *K-means-clustering* clasificando las observaciones.

```
# Se representan circunferencias con las asignaciones hechas por K-means-clustering
datos <- cbind(cluster = km_clusters$cluster, datos)
plot(x = datos[,"x"], y = datos[,"y"], col = km_clusters$cluster, pch = 1,
     cex = 2, lwd = 2, xlab = "X", ylab = "Y")

# Se rellenan las circunferencias con puntos del color real del grupo al
# que pertenecen las observaciones. Es necesario hacer coincidir los
# colores con el mismo orden que el devuelto por la función kmeans() ya
# que el clustering no asigna variable respuesta, solo agrupa las
# observaciones.

points(x = datos[,"x"], y = datos[,"y"],
       col = c(2, 1, 3, 4)[datos[, "grupo"]], pch = 19)
```



El gráfico muestra que solo dos observaciones se han agrupado incorrectamente. Este tipo de visualización es muy útil e informativa, sin embargo, solo es posible cuando se trabaja con dos dimensiones. Si los datos contienen más de dos variables (dimensiones), una posible solución es utilizar las dos primeras componentes principales obtenidas en un *PCA* previo.

El número de aciertos y errores puede representarse también en modo de matriz de confusión. A la hora de interpretarlas es importante recordar que el *clustering* asigna las observaciones a *clusters* cuyo identificador no tiene que por qué coincidir con la nomenclatura empleada para los grupos reales. Por ejemplo, el grupo 2 puede haberse identificado como *cluster 1* (tal como ocurre en este ejemplo). Así pues, por cada fila de la matriz cabe esperar un valor alto (coincidencias) para una de las posiciones y valores bajos en las otras (errores de clasificación).

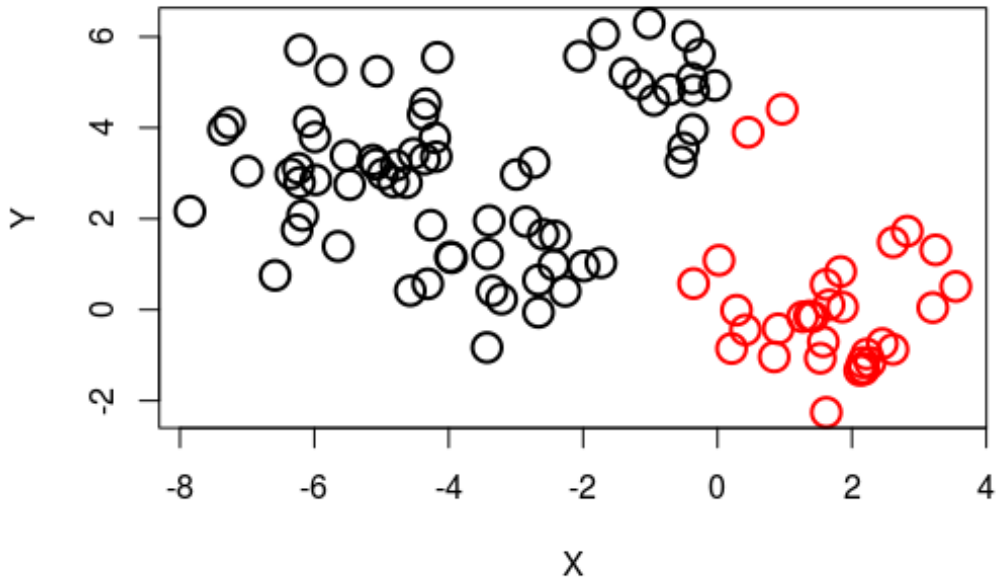
```
table(km_clusters$cluster, datos[, "grupo"],
      dnn = list("cluster", "grupo real"))
```

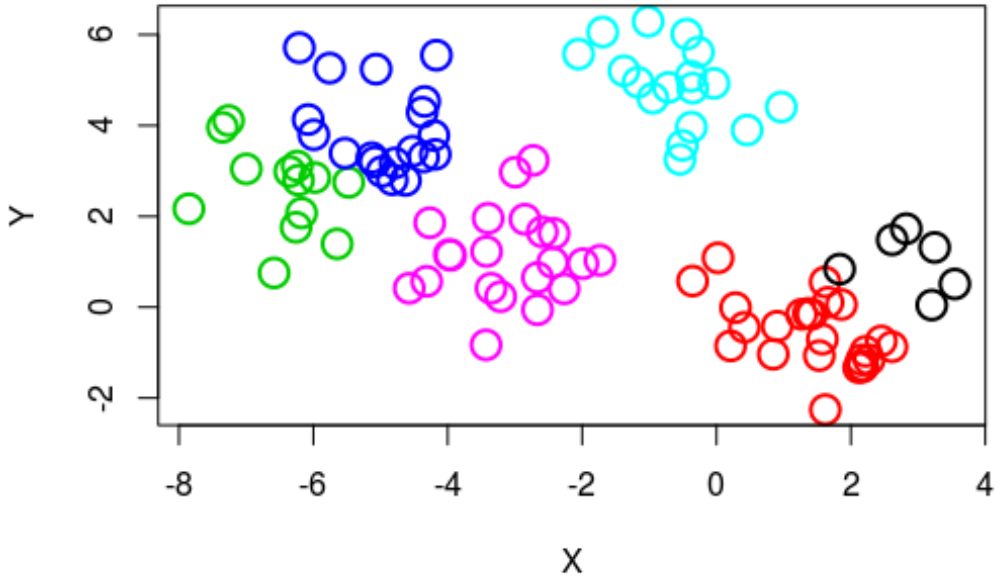
```
##         grupo real
## cluster  1  2  3  4
##       1  0 31  0  1
##       2 17  0  0  0
##       3  0  0 30  0
##       4  1  0  0 20
```

En este análisis, solo 2 de las 100 observaciones se han agrupado erróneamente. De nuevo repetir que, en la realidad, no se suelen conocer los verdaderos grupos en los que se dividen las observaciones, de lo contrario no se necesitaría aplicar *clustering*.

Supóngase ahora que se trata de un caso real en el que se desconoce el número de grupos en los que se subdividen las observaciones. El investigador tendría que probar con diferentes valores de K y decidir cuál parece más razonable (en los siguientes apartados se describen métodos más sofisticados). A continuación, se muestran los resultados para $K = 2$ y $K = 6$.

```
par(mfrow = c(2,1))
# Resultados para K = 2
set.seed(101)
km_clusters_2 <- kmeans(x = datos[, c("x", "y")], centers = 2, nstart = 50)
datos <- cbind(cluster = km_clusters_2$cluster, datos)
plot(x = datos[,"x"], y = datos[,"y"], col = km_clusters_2$cluster,
     pch = 1, cex = 2, lwd = 2, xlab = "X", ylab = "Y")

# Resultados para K = 6
set.seed(101)
km_clusters_6 <- kmeans(x = datos[, c("x", "y")], centers = 6, nstart = 50)
datos <- cbind(cluster = km_clusters_6$cluster, datos)
plot(x = datos[,"x"], y = datos[,"y"], col = km_clusters_6$cluster,
     pch = 1, cex = 2, lwd = 2, xlab = "X", ylab = "Y")
```





Al observar los resultados obtenidos para $K = 2$, es intuitivo pensar que el grupo que se encuentra entorno a las coordenadas $x = -1, y = 6$ (mayoritariamente considerado como negro) debería ser un grupo separado. Para $K = 6$ no parece muy razonable la separación de los grupos rojo y negro. Este ejemplo muestra la principal limitación del método de *K-means*, el hecho de tener que escoger de antemano el número de *clusters* que se generan.

Ejemplo: paquete factoextra

El paquete `factoextra` creado por Alboukadel Kassambara contiene funciones que facilitan en gran medida la visualización y evaluación de los resultados de *clustering*.

El set de datos `USArrests` *contiene información sobre el número de delitos (asaltos, asesinatos y secuestros) junto con el porcentaje de población urbana para cada uno de los 50 estados de USA. Se pretende estudiar si existe una agrupación subyacente de los estados empleando K-means-clustering.*

Si se emplea *K-means-clustering* con distancia euclídea hay que asegurarse de que las variables empleadas son de tipo continuo, ya que trabaja con la media de cada una de ellas.

```
data("USArrests")
str(USArrests)
```

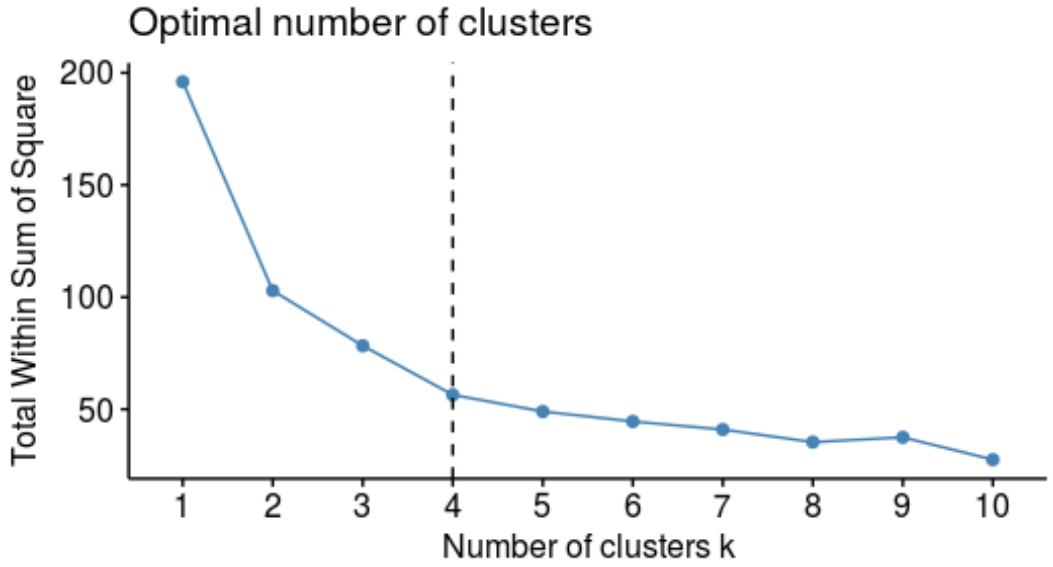
```
## 'data.frame':    50 obs. of  4 variables:
##  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
##  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
##  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
##  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

Como la magnitud de los valores difiere notablemente entre variables, se procede a escalarlas antes de aplicar el *clustering*.

```
datos <- scale(USArrests)
```

Una forma sencilla de estimar el número *K* óptimo de *clusters* cuando no se dispone de información adicional en la que basarse es aplicar el algoritmo para un rango de valores de *K*, identificando aquel a partir del cual la reducción en la suma total de varianza *intra-cluster* deja de ser sustancial (en los siguientes apartados se detallan otras opciones). La función `fviz_nbclust()` automatiza este proceso.

```
library(factoextra)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "wss",
             diss = dist(datos, method = "euclidean")) +
  geom_vline(xintercept = 4, linetype = 2)
```

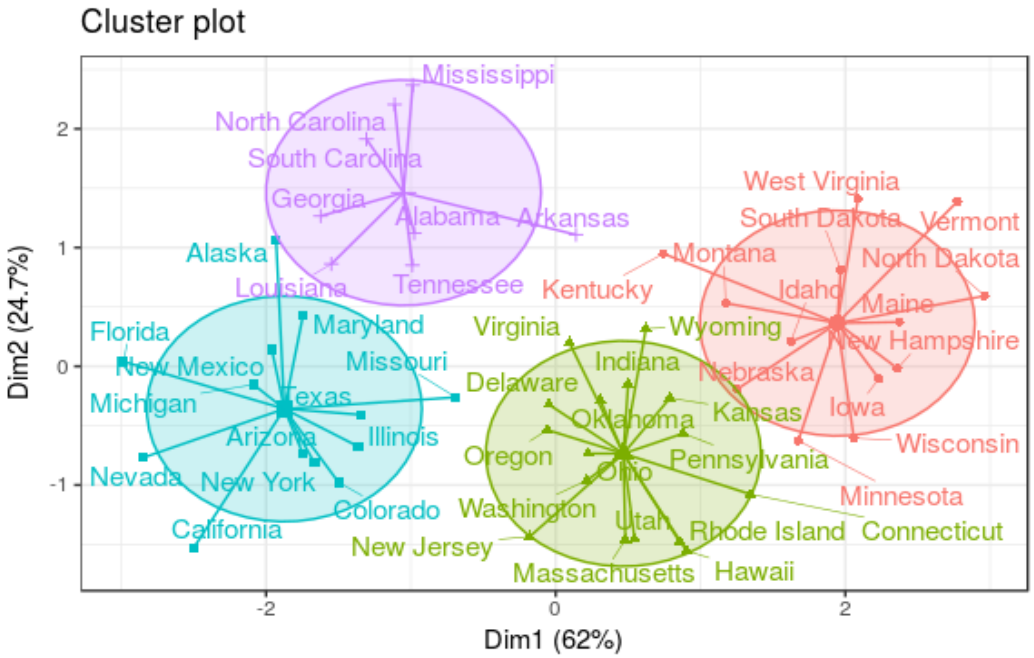


En este caso, a partir de 4 *clusters* la reducción en la suma total de cuadrados internos parece estabilizarse, indicando que $K = 4$ es una buena opción.

```
set.seed(123)
km_clusters <- kmeans(x = datos, centers = 4, nstart = 25)
```

El paquete `factoextra` también permite obtener visualizaciones de las agrupaciones resultantes. Si el número de variables (dimensionalidad) es mayor de 2, automáticamente realiza un *PCA* y representa las dos primeras componentes principales.

```
fviz_cluster(object = km_clusters, data = datos, show.clust.cent = TRUE,
             ellipse.type = "euclid", star.plot = TRUE, repel = TRUE) +
theme_bw() +
theme(legend.position = "none")
```



K-medoids clustering (PAM)

Idea intuitiva

K-medoids es un método de *clustering* muy similar a *K-means* en cuanto a que ambos agrupan las observaciones en *K clusters*, donde *K* es un valor preestablecido por el analista. La diferencia es que en *K-medoids* cada *cluster* está representado por una observación presente en el *cluster* (*medoid*), mientras que en *K-means* cada *cluster* está representado por su centroide, que se corresponde con el promedio de todas las observaciones del *cluster* pero con ninguna en particular.

Una definición más exacta del término *medoid* es: elemento dentro de un *cluster* cuya distancia (diferencia) promedio entre él y todos los demás elementos del mismo *cluster* es lo menor posible. Se corresponde con el elemento más central del *cluster* y por lo tanto puede considerarse como el más representativo. El hecho de utilizar *medoids* en lugar de centroides hace de *K-medoids* un método más robusto que *K-means*, viéndose menos afectado por *outliers* o ruido. A modo de idea intuitiva puede considerarse como la analogía entre media y mediana.

El algoritmo más empleado para aplicar *K-medoids* se conoce como *PAM* (*Partitioning Around Medoids*) y sigue los siguientes pasos:

1. Seleccionar *K* observaciones aleatorias como *medoids* iniciales. También es posible identificarlas de forma específica.
2. Calcular la matriz de distancia entre todas las observaciones si esta no se ha calculado anteriormente.
3. Asignar cada observación a su *medoid* más cercano.
4. Para cada uno de los *clusters* creados, comprobar si seleccionando otra observación como *medoid* se consigue reducir la distancia promedio del *cluster*, si esto ocurre seleccionar la observación que consigue una mayor reducción como nuevo *medoid*.
5. Si al menos un *medoid* ha cambiado en el paso 4, volver al paso 3, de lo contrario se termina el proceso.

Por lo general, el método de *K-medoids* se utiliza cuando se conoce o se sospecha de la presencia de *outliers*. Si esto ocurre, es recomendable utilizar como medida de similitud la distancia de Manhattan, ya que es menos sensible a *outliers* que la euclídea.

Ventajas y desventajas

- *K-medoids* es un método de *clustering* más robusto que *K-means*, por lo es más adecuado cuando el set de datos contiene *outliers* o ruido.
- Al igual que *K-means*, necesita que se especifique de antemano el número de *clusters* que se van a crear. Esto puede ser complicado de determinar si no se dispone de información adicional sobre los datos.
- Para sets de datos grandes necesita muchos recursos computacionales. En tal situación se recomienda aplicar el método *CLARA*.

Ejemplo

El set de datos `USArrests` *contiene información sobre el número de delitos (asaltos, asesinatos y secuestros) junto con el porcentaje de población urbana para cada uno de los 50 estados de USA. Se pretende estudiar si existe una agrupación subyacente de los estados mediante clustering. Dado que se sospecha de la presencia de outliers se recurre a K-medoids.*

El proceso a seguir en `R` para aplicar el método de *K-medoids* es igual al seguido en *K-means*, pero en este caso empleando la función `pam()` del paquete `cluster`.

```
data("USArrests")
str(USArrests)
```

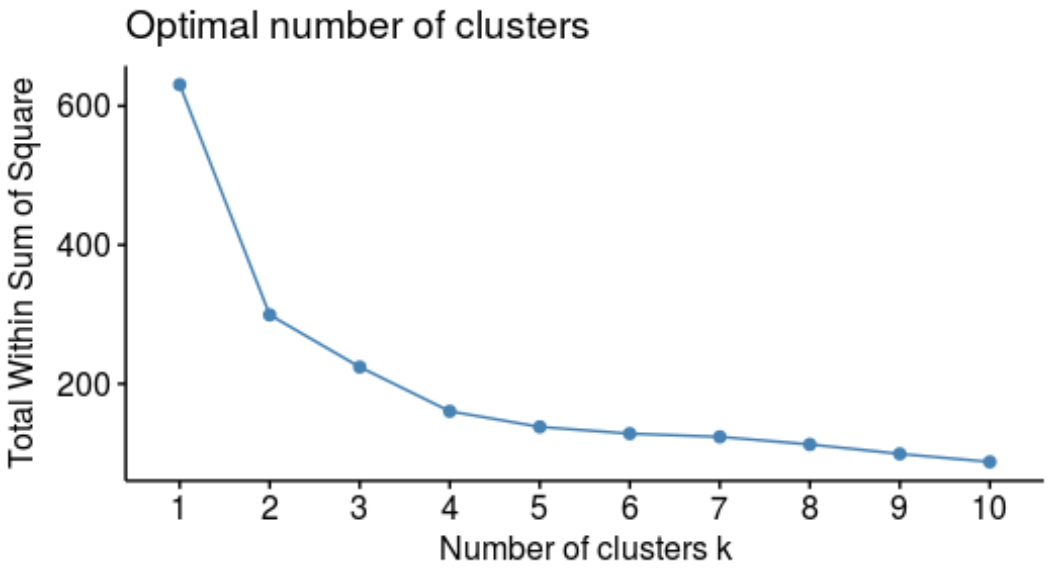
```
## 'data.frame':    50 obs. of  4 variables:
##  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
##  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
##  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
##  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

Como la magnitud de los valores difiere notablemente entre variables, se procede a escalarlas antes de aplicar el *clustering*.

```
datos <- scale(USArrests)
```

Una forma sencilla de estimar el número *K* óptimo de *clusters* cuando no se dispone de información adicional en la que basarse es aplicar el algoritmo para un rango de valores de *K*, identificando aquel a partir del cual la reducción en la suma total de varianza *intra-cluster* deja de ser sustancial (en las siguientes secciones se describen otras alternativas). La función `fviz_nbclust()` automatiza este proceso. En este caso, dado que se sospecha de la presencia de *outliers*, se emplea la distancia de Manhattan como medida de similitud.

```
library(cluster)
library(factoextra)
fviz_nbclust(x = datos, FUNcluster = pam, method = "wss",
             diss = dist(datos, method = "manhattan"))
```



A partir de 4 *clusters* la reducción en la suma total de cuadrados internos parece estabilizarse, indicando que $K = 4$ es una buena opción.

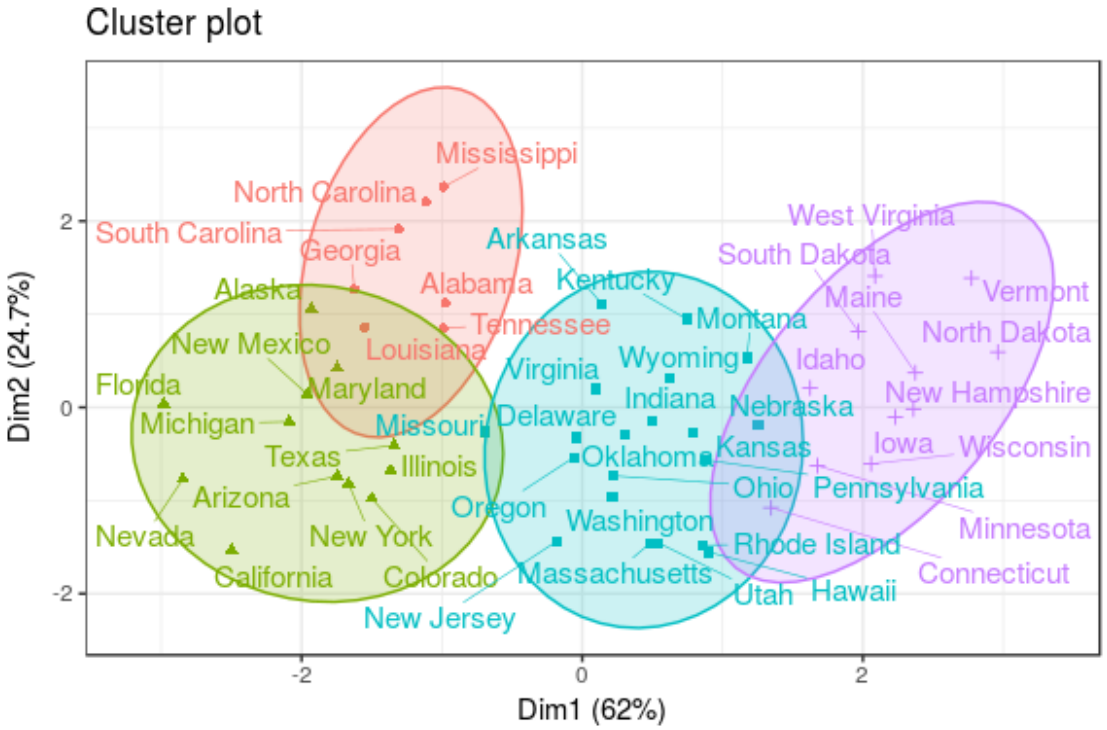
```
set.seed(123)
pam_clusters <- pam(x = datos, k = 4, metric = "manhattan")
pam_clusters
```

```
## Medoids:
##          ID     Murder    Assault   UrbanPop         Rape
## Alabama   1  1.2425641  0.7828393 -0.5209066 -0.003416473
## Michigan 22  0.9900104  1.0108275  0.5844655  1.480613993
## Oklahoma 36 -0.2727580 -0.2371077  0.1699510 -0.131534211
## Iowa     15 -1.2829727 -1.3770485 -0.5899924 -1.060387812
## Clustering vector:
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              2              2              3              2
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              4              3              2              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              3              4              2              3              4
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              3              3              1              4              2
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              3              2              4              1              3
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              3              3              2              4              3
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              2              2              1              4              3
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              3              3              3              3              1
```

```
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              4              1              2              3              4
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              3              3              4              4              3
## Objective function:
##    build     swap
## 1.730682 1.712075
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
##  [6] "clusinfo"   "silinfo"    "diss"       "call"       "data"
```

El objeto devuelto por `pam()` contiene entre otra información: las observaciones que finalmente se han seleccionado como *medoids* y el *cluster* al que se ha asignado cada observación.

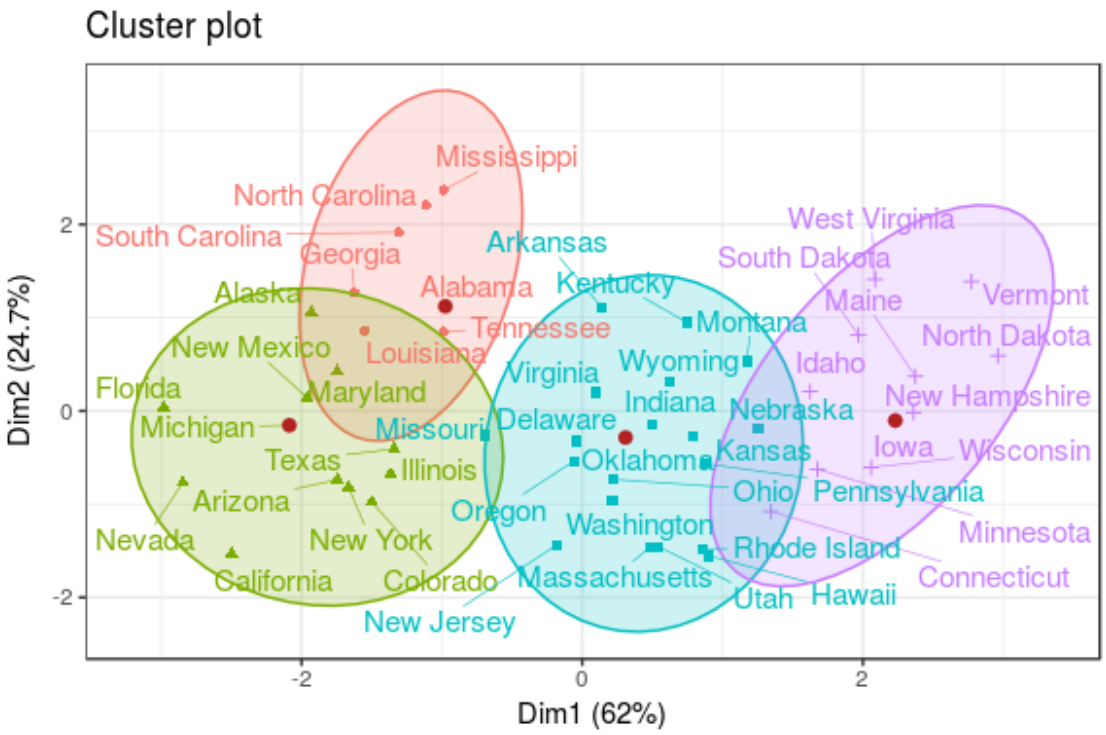
```
fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t",
             repel = TRUE) +
  theme_bw() +
  theme(legend.position = "none")
```




```
# Como en k-medoids no hay centroides, no se muestran en la representación ni
# tampoco las distancias desde este al resto de observaciones
```

La función `fviz_cluster()` no permite resaltar las observaciones que actúan como *medoids*, sin embargo, al tratarse de un objeto `ggplot2` es sencillo conseguirlo.

```
# Como hay más de 2 variables, se están representando las 2 primeras componentes
# de un PCA. Se tienen que calcular el PCA y extraer las proyecciones almacenadas
# en el elemento x
medoids <- prcomp(datos)$x
# Se seleccionan únicamente las proyecciones de las observaciones que son medoids
medoids <- medoids[rownames(pam_clusters$medoids), c("PC1", "PC2")]
medoids <- as.data.frame(medoids)
# Se emplean los mismos nombres que en el objeto ggplot
colnames(medoids) <- c("x", "y")

# Creación del gráfico
fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t",
             repel = TRUE) +
theme_bw() +
# Se resaltan las observaciones que actúan como medoids
geom_point(data = medoids, color = "firebrick", size = 2) +
theme(legend.position = "none")
```



CLARA

Idea intuitiva

Una de las limitaciones del método *K-medoids-clustering* es que su algoritmo requiere mucha memoria RAM, lo que impide que se pueda aplicar cuando el set de datos contiene varios miles de observaciones. *CLARA* (*Clustering Large Applications*) es un método que combina la idea de *K-medoids* con el *resampling* para que pueda aplicarse a grandes volúmenes de datos.

En lugar de intentar encontrar los *medoids* empleando todos los datos a la vez, *CLARA* selecciona una muestra aleatoria de un tamaño determinado y le aplica el algoritmo de *PAM* (*K-medoids*) para encontrar los *clusters* óptimos acorde a esa muestra. Utilizando esos *medoids* se agrupan las observaciones de todo el set de datos. La calidad de los *medoids* resultantes se cuantifica con la suma total de las distancias entre cada observación del set de datos y su correspondiente *medoid* (suma total de distancias *intra-clusters*). *CLARA* repite este proceso un número predeterminado de veces con el objetivo de reducir el *bias* de muestreo. Por último, se seleccionan como *clusters* finales los obtenidos con los *medoids* que han minimizado la suma total de distancias. A continuación, se describen los pasos del algoritmo *CLARA*.

1. Se divide aleatoriamente el set de datos en *n* partes de igual tamaño, donde *n* es un valor que determina el analista.
2. Para cada una de las *n* partes:
    a. Aplicar el algoritmo *PAM* e identificar cuáles son los *k medoids*.
    b. Utilizando los *medoids* del paso anterior agrupar todas las observaciones del set de datos.
    c. Calcular la suma total de las distancias entre cada observación del set de datos y su correspondiente *medoid* (suma total de distancias *intra-clusters*).
3. Seleccionar como *clustering* final aquel que ha conseguido menor suma total de distancias *intra-clusters* en el paso 3.

Ejemplo

Para ilustrar la aplicación del método CLARA se simula un set de datos bidimensional (dos variables) con 500 observaciones, de las cuales 200 pertenecen a un grupo y 300 a otro (número de grupos reales = 2).

```
set.seed(1234)
grupo_1 <- cbind(rnorm(n = 200, mean = 0, sd = 8),
                 rnorm(n = 200, mean = 0, sd = 8))
grupo_2 <- cbind(rnorm(n = 300, mean = 50, sd = 8),
                 rnorm(n = 300, mean = 50, sd = 8))
datos <- rbind(grupo_1, grupo_2)
colnames(datos) <- c("x", "y")
head(datos)
```

```
##               x        y
## [1,]  -9.656526 3.881815
## [2,]   2.219434 5.574150
## [3,]   8.675529 1.484111
## [4,] -18.765582 5.605868
## [5,]   3.432998 2.493448
## [6,]   4.048447 6.083699
```

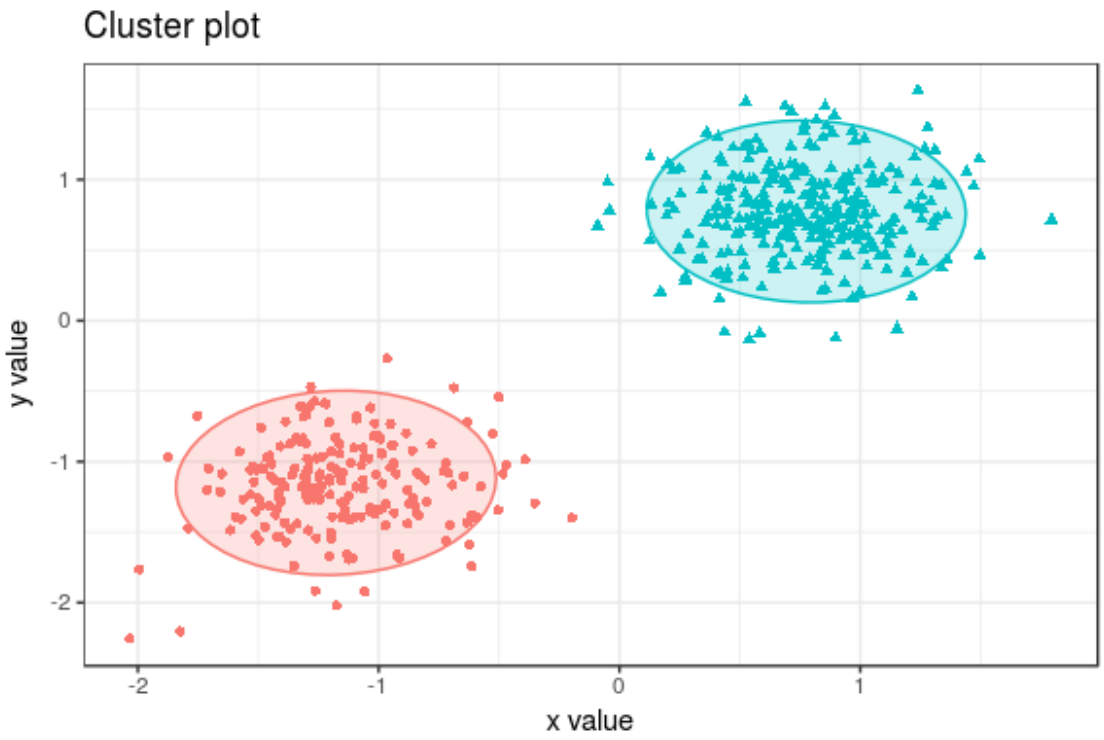
La función `clara()` del paquete `cluster` permite aplicar *CLARA*. Entre sus argumentos destaca: una matriz numérica `x` donde cada fila es una observación, el número de *clusters* `k`, la medida de distancia empleada (euclídea o manhattan) `metric`, si los datos se tienen que estandarizar `stand`, el número de partes `samples` en las que se divide el set de datos (recomendable 50) y el si se utiliza el algoritmo *PAM* `pamLike`.

```
library(cluster)
library(factoextra)
clara_clusters <- clara(x = datos, k = 2, metric = "manhattan", stand = TRUE,
                        samples = 50, pamLike = TRUE)
clara_clusters
```

```
## Call:     clara(x = datos, k = 2, metric = "manhattan", stand = TRUE, samples =
50,      pamLike = TRUE)
## Medoids:
##               x         y
## [1,] -0.2780831  1.269004
## [2,] 50.3298450 49.233511
## Objective function:   0.5266187
## Clustering vector:    int [1:500] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
## Cluster sizes:            200 300
```

```
## Best sample:
##  [1]  17  30  33  34  63  71  80  84  85 100 115 141 149 162 165 167 168
## [18] 171 188 214 223 249 259 261 291 302 318 320 333 386 391 402 412 417
## [35] 419 422 437 439 440 450 469 472 498 499
##
## Available components:
##  [1] "sample"     "medoids"    "i.med"      "clustering" "objective"
##  [6] "clusinfo"   "diss"       "call"       "silinfo"    "data"
```

```
fviz_cluster(object = clara_clusters, ellipse.type = "t", geom = "point") +
theme_bw() +
theme(legend.position = "none")
```



Hierarchical clustering

Idea intuitiva

Hierarchical clustering es una alternativa a los métodos de *partitioning clustering* que no requiere que se pre-especifique el número de *clusters*. Los métodos que engloba el *hierarchical clustering* se subdividen en dos tipos dependiendo de la estrategia seguida para crear los grupos:

- *Agglomerative clustering (bottom-up)*: el agrupamiento se inicia en la base del árbol, donde cada observación forma un *cluster* individual. Los *clusters* se van combinado a medida que la estructura crece hasta converger en una única "rama" central.
- *Divisive clustering (top-down)*: es la estrategia opuesta al *agglomerative clustering*, se inicia con todas las observaciones contenidas en un mismo *cluster* y se suceden divisiones hasta que cada observación forma un *cluster* individual.

En ambos casos los resultados pueden representarse de forma muy intuitiva en una estructura de árbol llamada dendrograma. En este capítulo se abarca únicamente métodos de *hierarchical clustering* de tipo *agglomerative*.

Algoritmo

La estructura resultante de un *agglomerative hierarchical clustering* se obtiene mediante un algoritmo sencillo.

1. El proceso se inicia considerando cada una de las observaciones como un *cluster* individual, formando así la base del dendrograma.
2. Se inicia un proceso iterativo hasta que todas las observaciones pertenecen a un único *cluster*:
    a. Se calcula la distancia entre cada posible par de los *n clusters*. El investigador debe determinar el tipo de medida emplea para cuantificar la similitud entre observaciones o grupos (distancia y *linkage*).
    b. Los dos *clusters* más similares se fusionan, de forma que quedan *n*-1 *clusters*.
3. Determinar dónde cortar la estructura de árbol generada (dendrograma).

La siguiente imagen muestra cómo se van sucediendo las agrupaciones a medida que avanzan las primeras iteraciones del algoritmo.

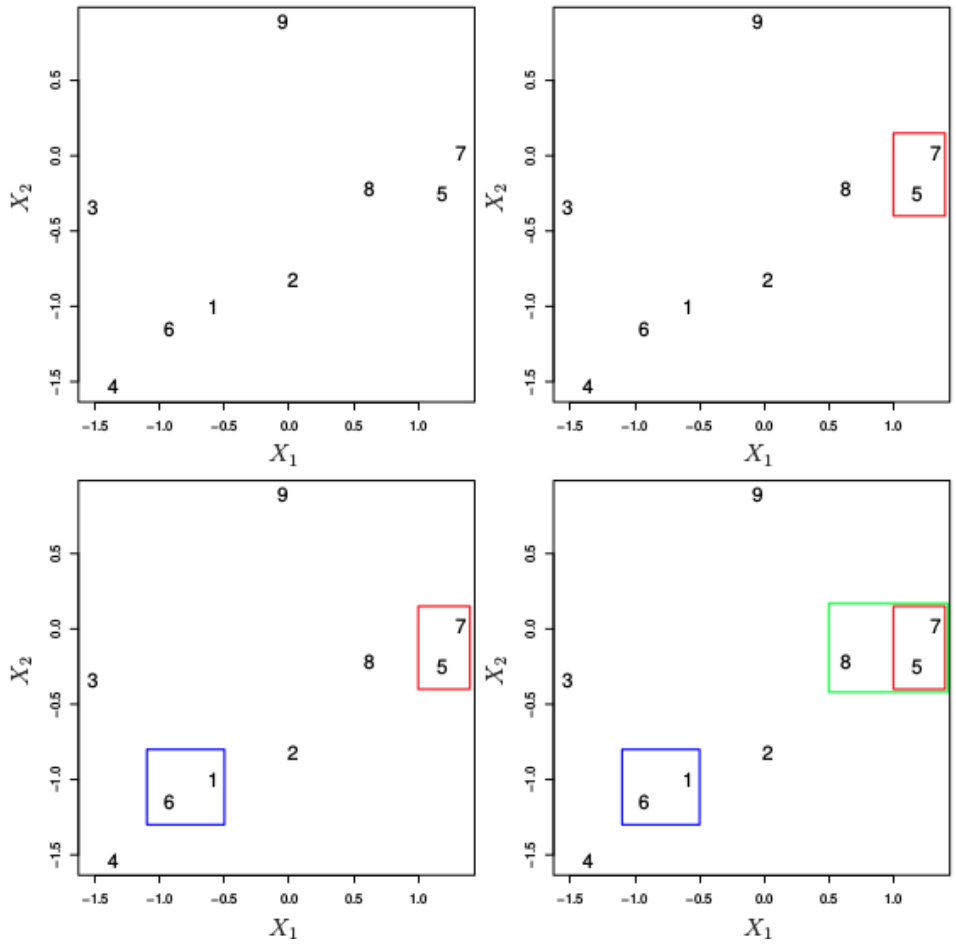


Imagen obtenida del libro ISLR

Para que el proceso de agrupamiento pueda llevarse a cabo tal como indica el algoritmo anterior, es necesario definir cómo se cuantifica la similitud entre dos *clusters*. Es decir, se tiene que extender el concepto de distancia entre pares de observaciones para que sea aplicable a pares de grupos, cada uno formado por varias observaciones. A este proceso se le conoce como *linkage*. A continuación, se describen los 5 tipos de *linkage* más empleados y sus definiciones.

- **Complete or Maximum**: Se calcula la distancia entre todos los posibles pares formados por una observación del *cluster A* y una del *cluster B*. La mayor de todas ellas se selecciona como la distancia entre los dos *clusters*. Se trata de la medida más conservadora (*maximal intercluster dissimilarity*).

- **Single or Minimum**: Se calcula la distancia entre todos los posibles pares formados por una observación del *cluster A* y una del *cluster B*. La menor de todas ellas se

selecciona como la distancia entre los dos *clusters*. Se trata de la medida menos conservadora (*minimal intercluster dissimilarity*).

- **Average**: Se calcula la distancia entre todos los posibles pares formados por una observación del *cluster A* y una del *cluster B*. El valor promedio de todas ellas se selecciona como la distancia entre los dos *clusters* (*mean intercluster dissimilarity*).

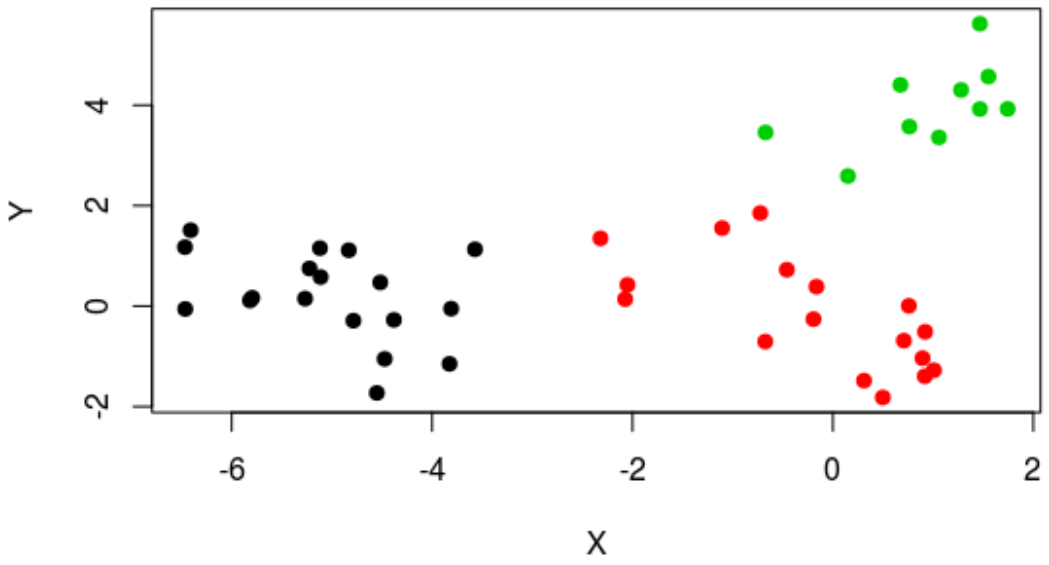
- **Centroid**: Se calcula el centroide de cada uno de los *clusters* y se selecciona la distancia entre ellos como la distancia entre los dos *clusters*.

- **Ward**: Se trata de un método general. La selección del par de *clusters* que se combinan en cada paso del *agglomerative hierarchical clustering* se basa en el valor óptimo de una función objetivo, pudiendo ser esta última cualquier función definida por el analista. El conocido método *Ward's minimum variance* es un caso particular en el que el objetivo es minimizar la suma total de varianza *intra-cluster*. En cada paso se identifican aquellos 2 *clusters* cuya fusión conlleva menor incremento de la varianza total *intra-cluster*. La implementación en R de este método ha sido causa de confusiones (*Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? by Fionn Murtagh y Pierre Legendre*). Si se emplea la función `hclust()` se tiene que especificar `method = "ward.D2"`, mientras que en la función `agnes()` es `method = ward`".

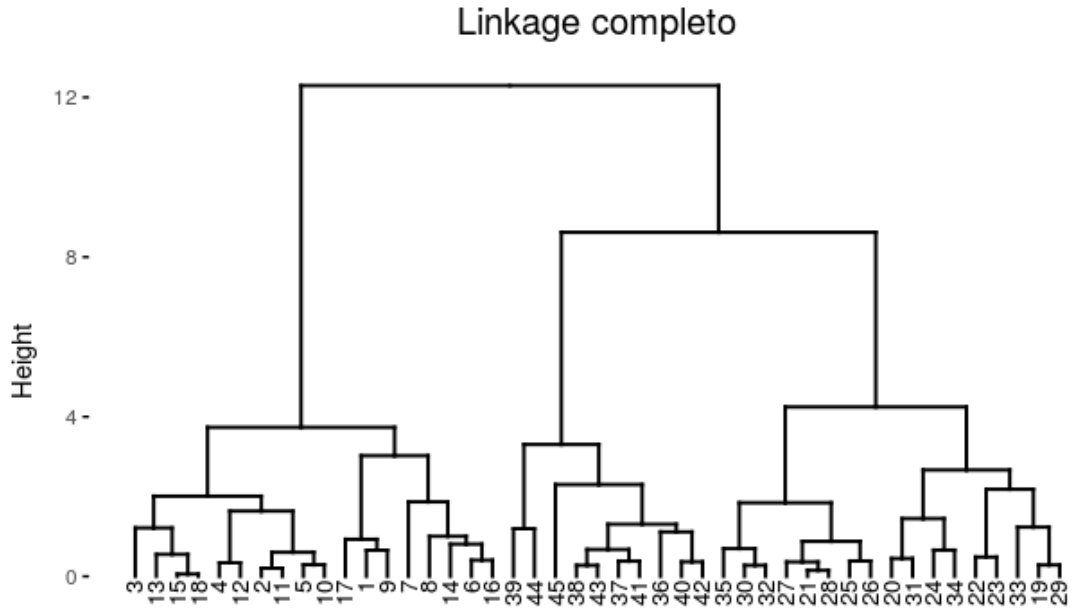
Los métodos de *linkage complete*, *average* y *Ward's minimum variance* suelen ser los preferidos por los analistas debido a que generan dendrogramas más compensados. Sin embargo, no se puede determinar que uno sea mejor que otro, ya que depende del caso de estudio en cuestión. Por ejemplo, en genómica se emplea con frecuencia el método de *centroides*. Junto con los resultados de un proceso de *hierarchical clustering* siempre hay que indicar qué distancia se ha empleado, así como el tipo de *linkage*, ya que dependiendo de estos los resultados pueden variar en gran medida.

Dendrograma

Supóngase que se dispone de 45 observaciones en un espacio de dos dimensiones que pertenecen a 3 grupos. Aunque se ha coloreado de forma distinta cada uno de los grupos para facilitar la comprensión de la idea, se va a suponer que se desconoce esta información y que se desea aplicar el método de *hierarchical clustering* para intentar reconocer los grupos.



Al aplicar *hierarchical clustering* empleando como medida de similitud la distancia euclídea y *linkage complete*, se obtiene el siguiente dendrograma.



En la base del dendrograma, cada observación forma una terminación individual conocida como hoja o *leaf* del árbol. A medida que se asciende por la estructura, pares de hojas se fusionan formando las primeras ramas. Estas uniones (nodos) se corresponden con los pares de observaciones más similares. También ocurre que ramas se fusionan con otras ramas o con hojas. Cuanto más temprana (más próxima a la base del dendrograma) ocurre una fusión,

mayor es la similitud. Esto significa que, para cualquier par de observaciones, se puede identificar el punto del árbol en el que las ramas que contienen dichas observaciones se fusionan. La altura a la que esto ocurre (eje vertical) indica cómo de similares/diferentes son las dos observaciones. Los dendrogramas, por lo tanto, se deben interpretar únicamente en base al eje vertical y no por las posiciones que ocupan las observaciones en el eje horizontal, esto último es simplemente por estética y puede variar de un programa a otro. Por ejemplo, la observación 11 es la más similar a la 2 ya que es la primera fusión que recibe la observación 2 (y viceversa). Podría resultar tentador decir que la observación 12, situada inmediatamente a la derecha de la 2, es la siguiente más similar, sin embargo, las observaciones 5 y 10 son más similares a la 2 a pesar de que se encuentran más alejadas en el eje horizontal. Del mismo modo, no es correcto decir que la observación 12 es más similar a la observación 2 de lo que lo es la 4 por el hecho de que está más próxima en el eje horizontal. Prestando atención a la altura en que las respectivas ramas se unen, la única conclusión válida es que la similitud entre los pares 4-2 y 12-2 es la misma.

Verificar el árbol resultante

Una vez creado el dendrograma, hay que evaluar hasta qué punto su estructura refleja las distancias originales entre observaciones. Una forma de hacerlo es empleando el coeficiente de correlación entre las distancias *cophenetic* del dendrograma (altura de los nodos) y la matriz de distancias original. Cuanto más cercano es el valor a 1, mejor refleja el dendrograma la verdadera similitud entre las observaciones. Valores superiores a 0.75 suelen considerarse como buenos. Esta medida puede emplearse como criterio de ayuda para escoger entre los distintos métodos de *linkage*. En `R`, la función `cophenetic()` calcula las distancias *cophenetic* de un *hierarchical clustering*.

```
data(USArrests)
datos <- scale(USArrests)

# Matriz de distancias euclídeas
mat_dist <- dist(x = datos, method = "euclidean")
# Dendrogramas con linkage complete y average
hc_complete <- hclust(d = mat_dist, method = "complete")
hc_average <- hclust(d = mat_dist, method = "average")
cor(x = mat_dist, cophenetic(hc_complete))
```

```
## [1] 0.6979437
```

```
cor(x = mat_dist, cophenetic(hc_average))
```

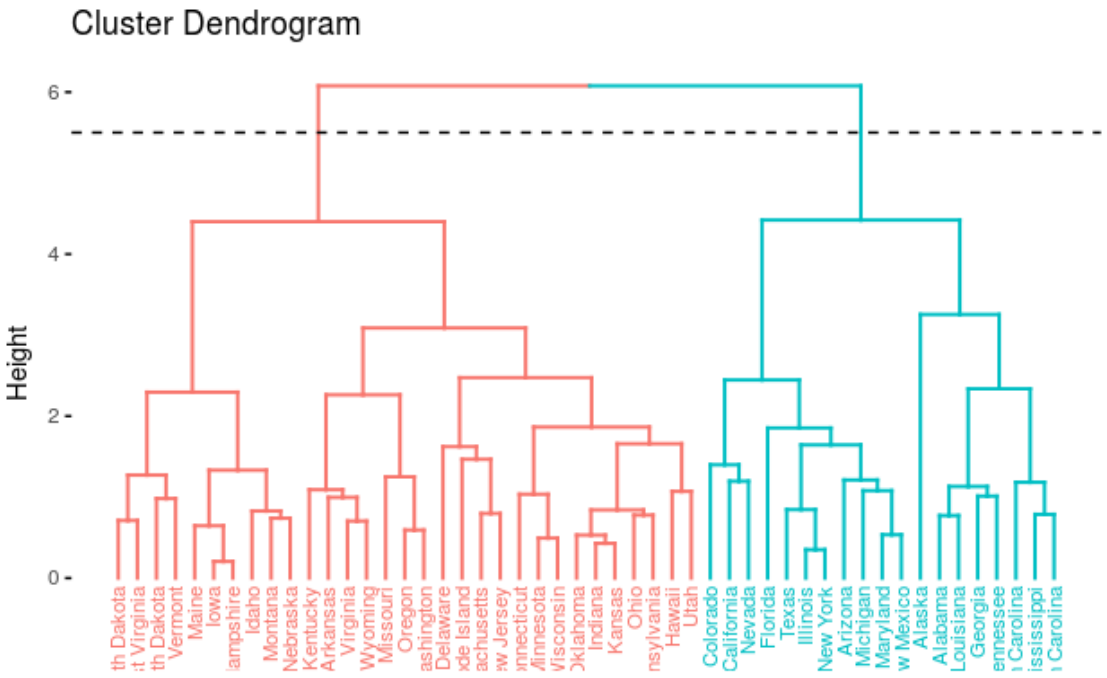
```
## [1] 0.7180382
```

En este caso, el método de *linkage average* consigue representar ligeramente mejor la similitud entre observaciones.

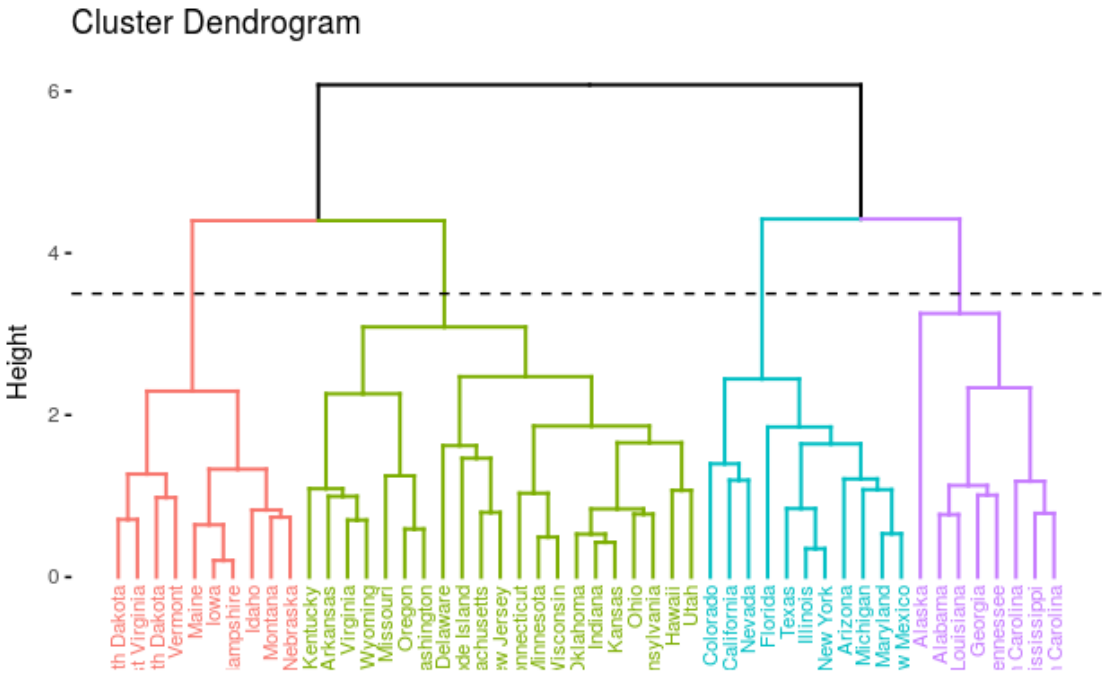
Cortar el árbol para generar los clusters

Además de representar en un dendrograma la similitud entre observaciones, se tiene que poder identificar el número de *clusters* creados y qué observaciones forman parte de cada uno. Si se realiza un corte horizontal a una determinada altura del dendrograma, el número de ramas que sobrepasan (en sentido ascendente) dicho corte se corresponde con el número de *clusters*. La siguiente imagen muestra dos veces el mismo dendrograma. Si se realiza el corte a la altura de 5, se obtienen dos *clusters*, mientras que si se hace a la de 3.5 se obtienen 4. La altura de corte tiene por lo tanto la misma función que el valor *K* en *K-means-clustering*: controla el número de *clusters* obtenidos.

```
library(factoextra)
datos <- USArrests
set.seed(101)

hc_completo <- datos %>% scale() %>% dist(method = "euclidean") %>%
               hclust(method = "complete")
fviz_dend(x = hc_completo, k = 2, cex = 0.6) +
  geom_hline(yintercept = 5.5, linetype = "dashed")
```



```
fviz_dend(x = hc_completo, k = 4, cex = 0.6) +
  geom_hline(yintercept = 3.5, linetype = "dashed")
```



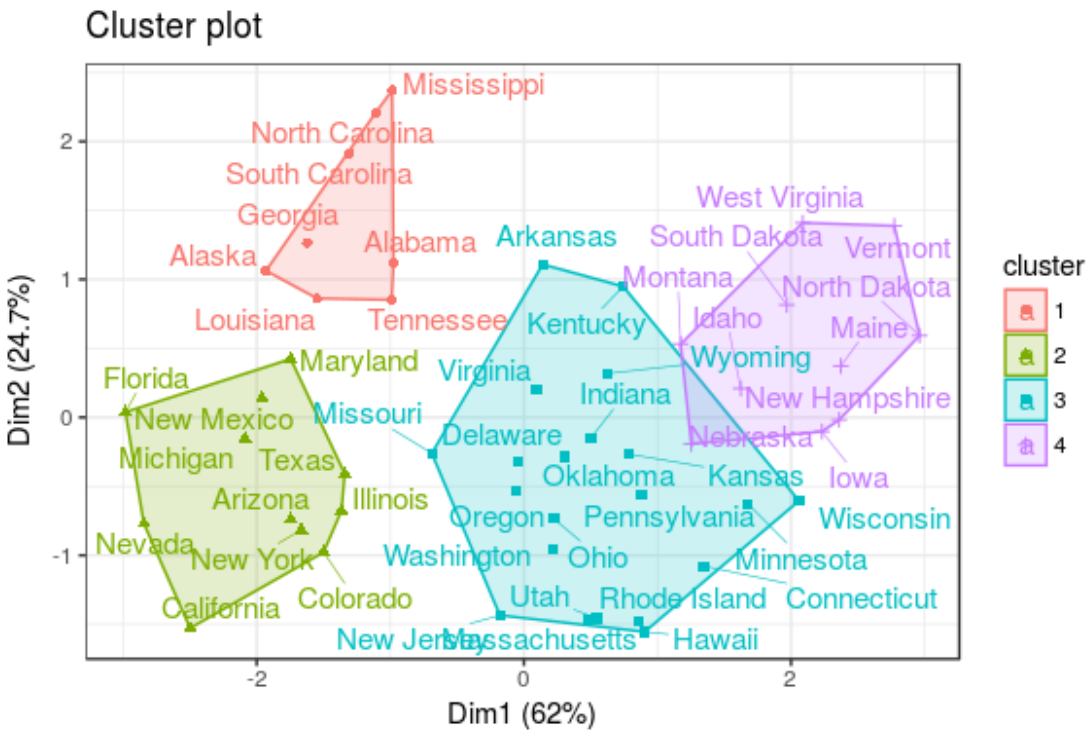
Dos propiedades adicionales se derivan de la forma en que se generan los *clusters* en el método de *hierarchical clustering*:

- Dada la longitud variable de las ramas, siempre existe un intervalo de altura para el que cualquier corte da lugar al mismo número de *clusters*. En el ejemplo anterior, todos los cortes entre las alturas 5 y 6 tienen como resultado los mismos 2 *clusters*.

- Con un solo dendrograma se dispone de la flexibilidad para generar cualquier número de *clusters* desde 1 a *n*. La selección del número óptimo suele hacerse de forma visual, tratando de identificar las ramas principales en base a la altura a la que ocurren las uniones. En el ejemplo expuesto es razonable elegir entre 2 o 4 *clusters*.

Una forma menos frecuente de representar los resultados de un *hierarchical clustering* es combinándolos con una reducción de dimensionalidad por *PCA*. Primero se calculan las componentes principales, se representan las observaciones en un *scatterplot* empleando las dos primeras componentes y finalmente se colorean los *clusters* mediante elipses.

```
fviz_cluster(object = list(data = datos, cluster = cutree(hc_completo, k = 4)),
             ellipse.type = "convex",
             repel = TRUE,
             show.clust.cent = FALSE) +
theme_bw()
```

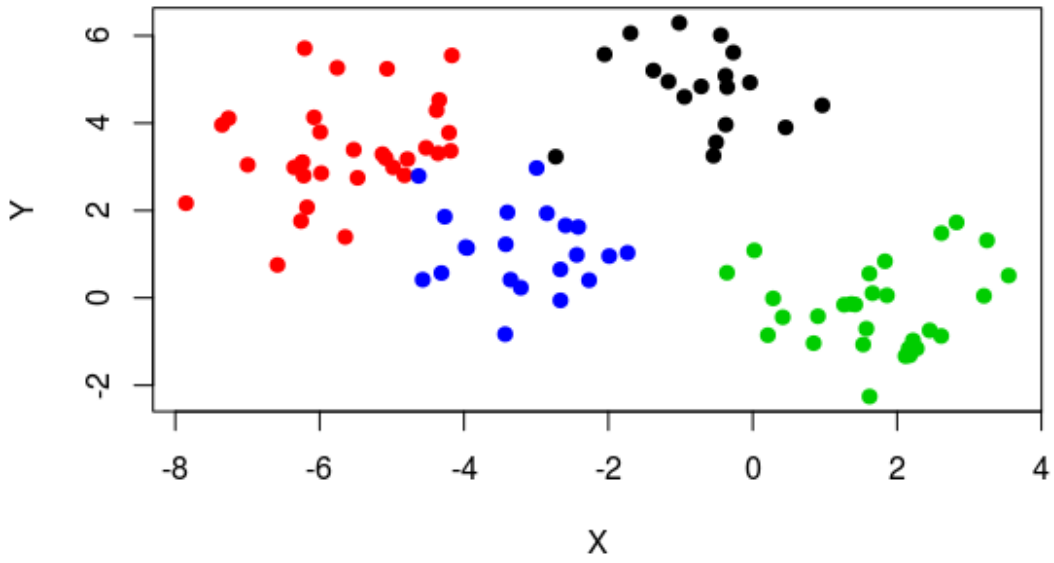


Ejemplo

Los siguientes datos simulados contienen observaciones que pertenecen a cuatro grupos distintos. Se pretende aplicar hierarchical clustering aglomerativo con el fin de identificarlos.

```
set.seed(101)
# Se simulan datos aleatorios con dos dimensiones
datos <- matrix(rnorm(n = 100*2), nrow = 100, ncol = 2,
                dimnames = list(NULL,c("x", "y")))

# Se determina la media que va a tener cada grupo en cada una de las dos
# dimensiones. En total 2*4 medias. Este valor se va a utilizar para
# separar cada grupo de los demás.
media_grupos <- matrix(rnorm(n = 8, mean = 0, sd = 4), nrow = 4, ncol = 2,
                       dimnames = list(NULL, c("media_x", "media_y")))
media_grupos <- cbind(grupo = 1:4, media_grupos)

# Se genera un vector que asigne aleatoriamente cada observación a uno de
# los 4 grupos
grupo <- sample(x = 1:4, size = 100, replace = TRUE)
datos <- cbind(datos, grupo)
```

```
# Se incrementa el valor de cada observación con la media correspondiente al
# grupo asignado.
datos <- merge(datos, media_grupos, by = "grupo")
datos[,"x"] <- datos[, "x"] + datos[, "media_x"]
datos[,"y"] <- datos[, "y"] + datos[, "media_y"]

plot(x = datos[,"x"], y = datos[,"y"], col = datos[,"grupo"], pch = 19,
     xlab = "X", ylab = "Y")
```

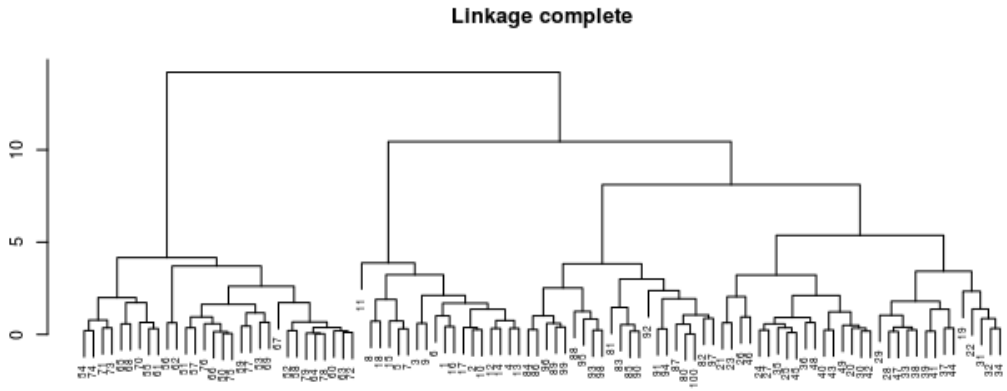


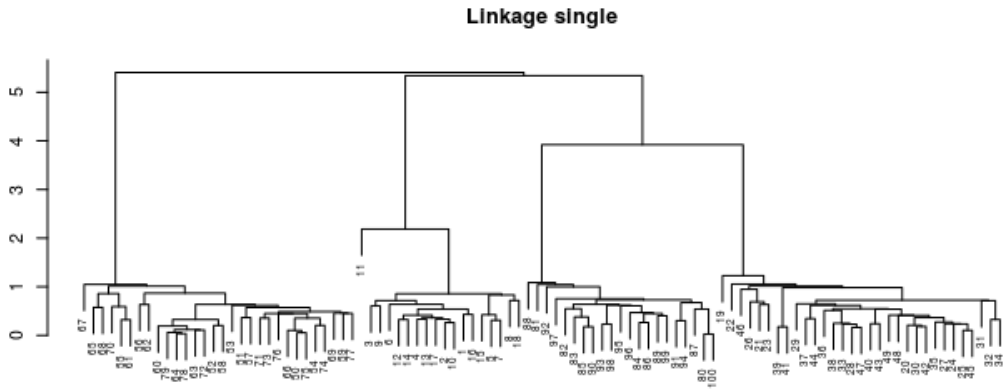
Al aplicar un *hierarchical clustering* se tiene que escoger una medida de distancia (1-similitud) y un tipo de *linkage*. En este caso, se emplea la función `hclust()` indicando la distancia euclídea como medida de similitud y se comparan los *linkages complete*, *single* y *average*. Dado que los datos se han simulado considerando que las dos dimensiones tienen aproximadamente la misma magnitud, no es necesario escalarlos ni centrarlos.

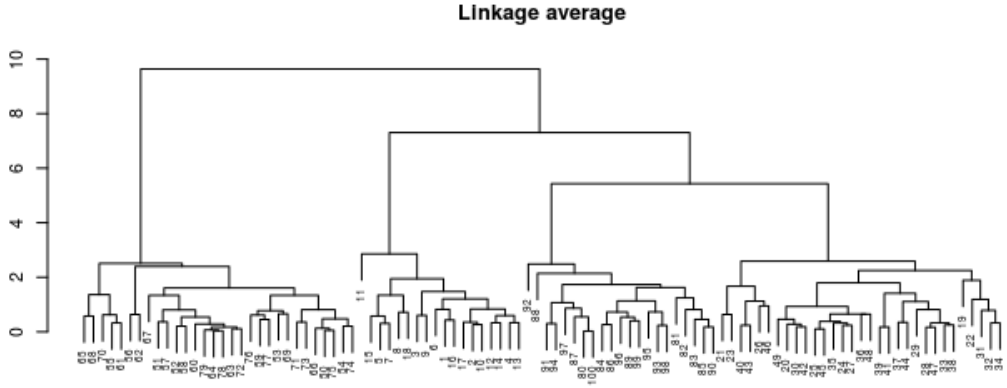
```
matriz_distancias <- dist(x = datos, method = "euclidean")
set.seed(567)
h_cluster_completo <- hclust(d = matriz_distancias, method = "complete")
h_cluster_single <- hclust(d = matriz_distancias, method = "single")
h_cluster_average <- hclust(d = matriz_distancias, method = "average")
```

Los objetos devueltos por `hclust()` pueden representarse en forma de dendrograma con la función `plot()` o con la función `fviz_dend()` del paquete `factoextra`.

```
par(mfrow = c(3,1))
plot(x = h_cluster_completo, cex = 0.6, xlab = "", ylab = "", sub = "",
     main = "Linkage complete")
plot(x = h_cluster_single, cex = 0.6, xlab = "", ylab = "", sub = "",
     main = "Linkage single")
plot(x = h_cluster_average, cex = 0.6, xlab = "", ylab = "", sub = "",
     main = "Linkage average")
```







El conocer que existen 4 grupos en la población permite evaluar qué *linkage* consigue los mejores resultados. En este caso, los tres tipos identifican claramente 4 *clusters*, si bien esto no significa que en los 3 dendrogramas los *clusters* estén formados por exactamente las mismas observaciones.

Una vez creado el dendrograma, se tiene que decidir a qué altura se corta para generar los *clusters*. La función `cutree()` recibe como *input* un dendrograma y devuelve el *cluster* al que se ha asignado cada observación dependiendo del número de *clusters* especificado (argumento `k`) o la altura de corte indicada (argumento `h`).

```
cutree(h_cluster_completo, k = 4)
```

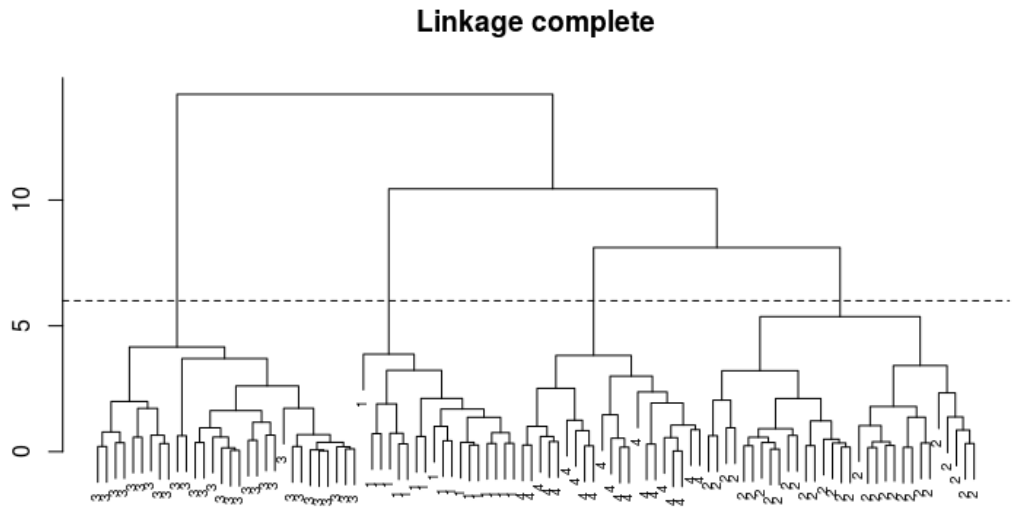
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

```
cutree(h_cluster_completo, h = 6)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

Una forma visual de comprobar los errores en las asignaciones es indicando en el argumento `labels` el grupo real al que pertenece cada observación. Si la agrupación resultante coincide con los grupos reales, entonces, dentro de cada *clusters* las *labels* serán las mismas.

```
plot(x = h_cluster_completo, cex = 0.6, main = "Linkage complete", sub = "",
     xlab = "", ylab = "", labels = datos[, "grupo"])
abline(h = 6, lty = 2)
```



```
table(cutree(h_cluster_completo, h = 6), datos[, "grupo"])
```

```
##
##      1  2  3  4
##   1 18  0  0  0
##   2  0 31  0  0
##   3  0  0 30  0
##   4  0  0  0 21
```

El método de *hierarchical clustering aglomerativo* con *linkage* completo ha sido capaz de agrupar correctamente todas las observaciones.

Ejemplo: Clasificar tumores por su perfil genético

El set de datos `NCI60` *contiene información genética de 64 líneas celulares cancerígenas. Para cada una de ellas se ha cuantificado la expresión de 6830 genes mediante tecnología microarray. Los investigadores conocen el tipo de cáncer (histopatología) al que pertenece cada línea celular y quieren utilizar esta información para evaluar si el método de clustering (hierarchical clustering) es capaz de agrupar correctamente las líneas empleando los niveles de expresión génica.*

Los métodos de *clustering* son *unsupervised*, lo que significa que al aplicarlos no se hace uso de la variable respuesta, en este caso el tipo de cáncer. Una vez obtenidos los resultados se añade esta información para determinar si es posible agrupar a las líneas celulares empleando su perfil de expresión.

```
library(ISLR)
data(NCI60)
str(NCI60)
```

```
## List of 2
##  $ data: num [1:64, 1:6830] 0.3 0.68 0.94 0.28 0.485 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:64] "V1" "V2" "V3" "V4" ...
##   .. ..$ : chr [1:6830] "1" "2" "3" "4" ...
##  $ labs: chr [1:64] "CNS" "CNS" "CNS" "RENAL" ...
```

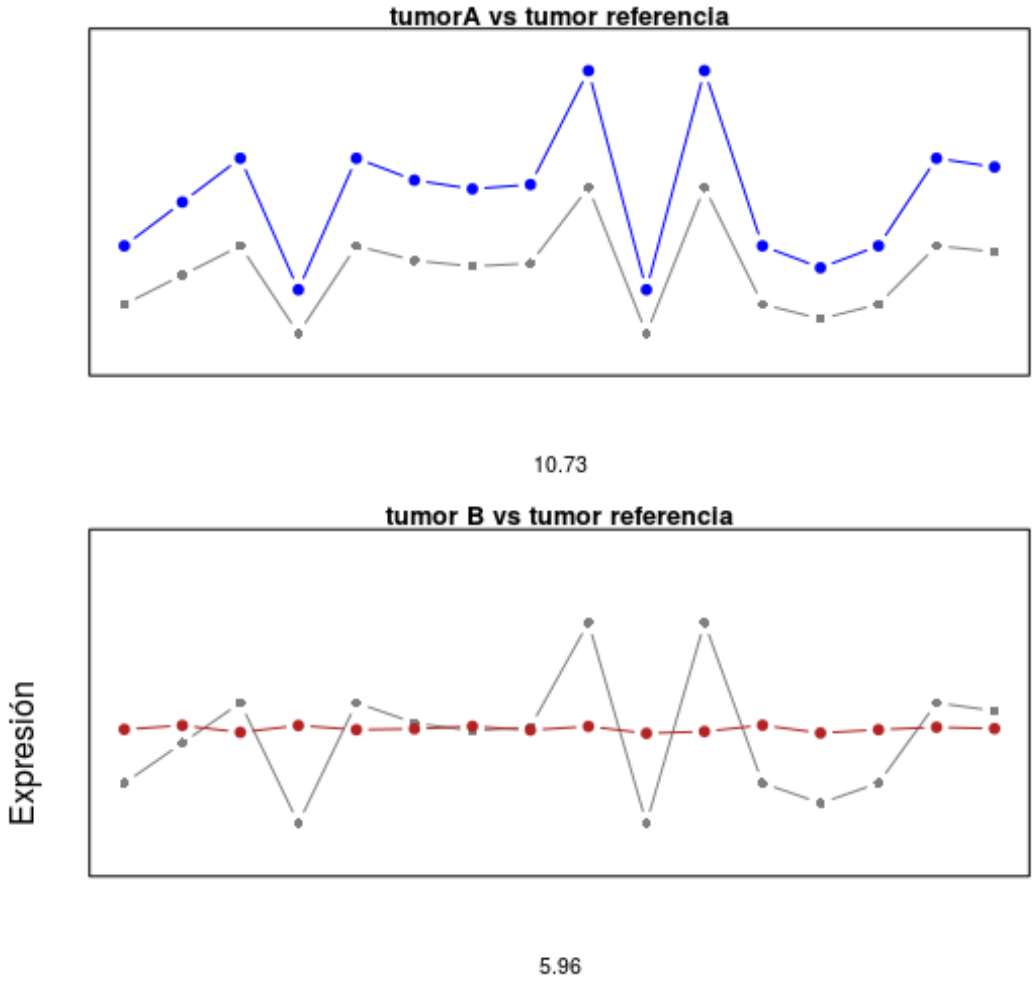
```
# Los datos están almacenados en forma de lista, un elemento contiene los niveles
# de expresión y otro el tipo de cáncer
expresion <- NCI60$data
tipo_cancer <- NCI60$labs
```

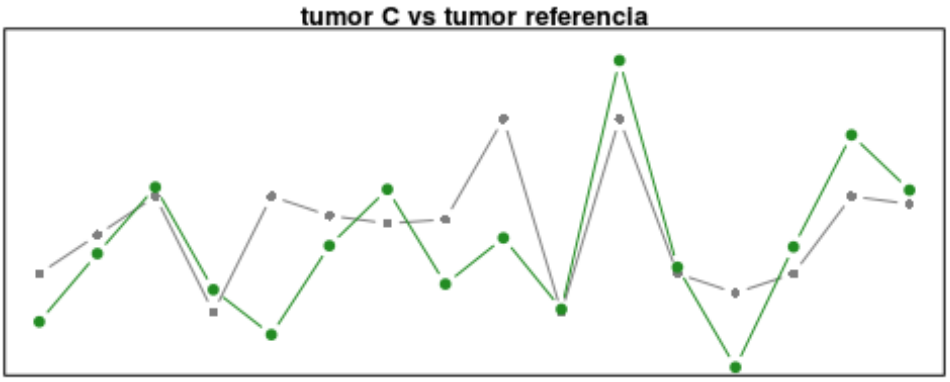
Una exploración inicial de los datos permite conocer el número de líneas celulares que hay de cada tipo de cáncer.

```
table(tipo_cancer)
```

```
## tipo_cancer
##      BREAST         CNS       COLON K562A-repro K562B-repro    LEUKEMIA 
##           7           5           7           1           1           6 
## MCF7A-repro MCF7D-repro    MELANOMA       NSCLC     OVARIAN    PROSTATE 
##           1           1           8           9           6           2 
##       RENAL     UNKNOWN 
##           9           1
```

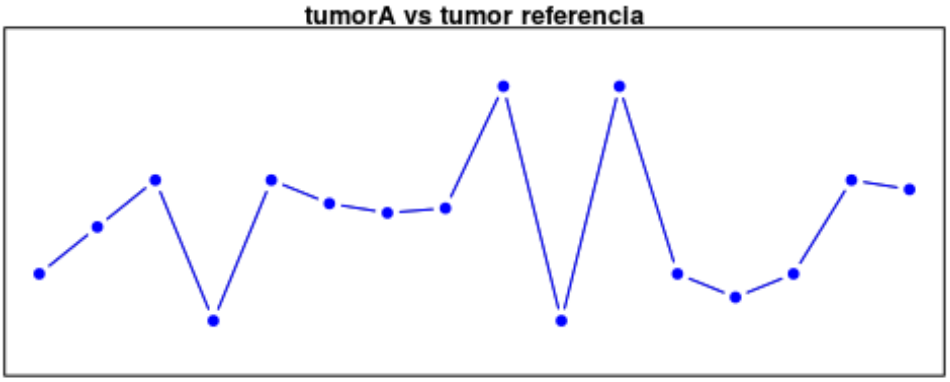
El siguiente paso antes de aplicar el método de *clustering* es decidir cómo se va a cuantificar la similitud. Los perfiles de expresión génica son un claro ejemplo de que es necesario comprender el problema en cuestión para hacer la elección adecuada. Para ilustrarlo se simula el perfil de 16 genes en 4 tumores (1 de referencia contra el que se comparan los otros 3).



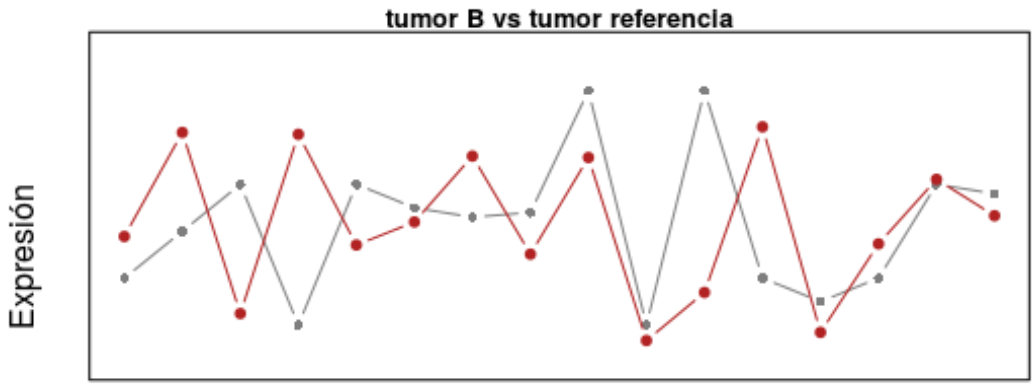


6.15

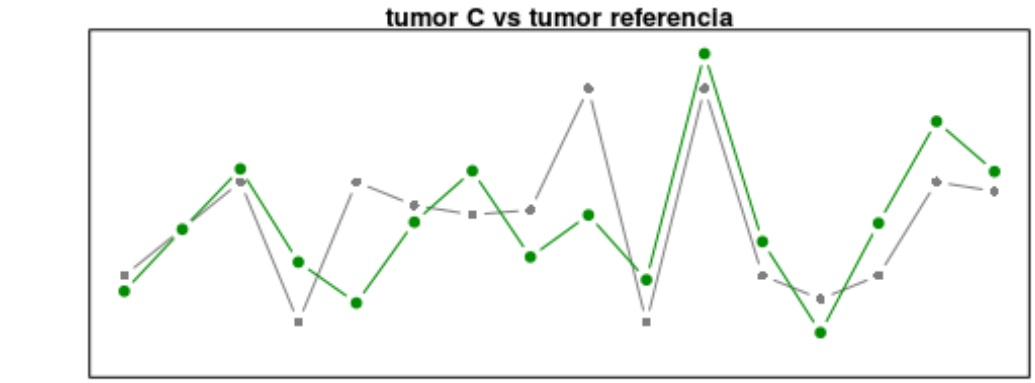
Debajo de cada gráfico se indica la distancia euclídea entre cada par de tumores. Acorde a esta medida, el tumor menos parecido al de referencia es el *A* y el más parecido el *B*. Sin embargo, analizando los perfiles con detenimiento puede observarse que el tumor *A* tiene exactamente el mismo perfil que el tumor de referencia, pero desplazado unas unidades; mientras que el tumor *B* tiene un perfil totalmente distinto. Para evitar que las diferencias en magnitud determinen la similitud, se convierten los valores de expresión en *Z-factors* de forma que tengan media 0 y desviación estándar 1.



0



5.32



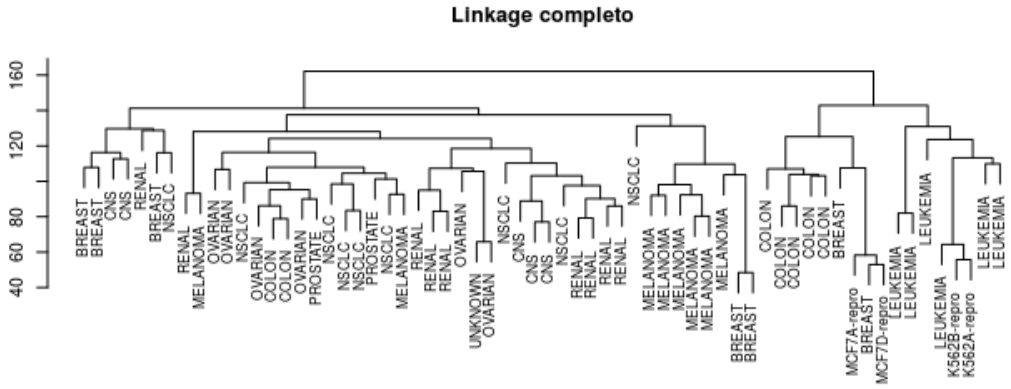
3.17

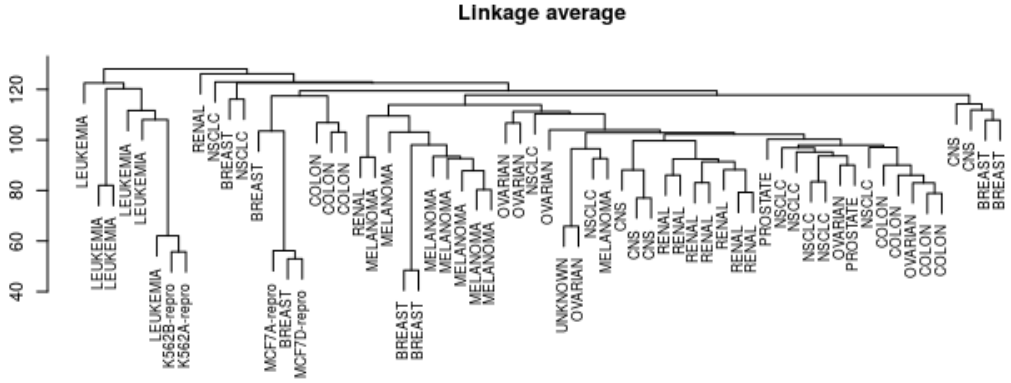
Una vez aplicada la estandarización, las distancias obtenidas en las comparaciones tienen más sentido dentro del contexto de los perfiles de expresión génica. La similitud entre el tumor *A* y el de referencia es total (distancia 0), y el tumor *B* pasa a ser el más distinto. El mismo resultado se hubiese obtenido empleando como medida de similitud la correlación lineal de Pearson.

Se procede por lo tanto a escalar la matriz de expresión `NCI60` y se aplica *hierarchical clustering* con *linkage completo*, *average* y *single*.

```
expresion <- scale(expresion, center = TRUE, scale = TRUE)
matriz_distancias <- dist(x = expresion, method = "euclidean")
hc_completo <- hclust(d = matriz_distancias, method = "complete")
hc_average  <- hclust(d = matriz_distancias, method = "average")
hc_single   <- hclust(d = matriz_distancias, method = "single")
par(mfrow = c(3, 1))
```

```
plot(hc_completo, labels = tipo_cancer, ylab = "", xlab = "", sub = "",
     main = "Linkage completo", cex = 0.8)
plot(hc_average, labels = tipo_cancer, ylab = "", xlab = "", sub = "",
     main = "Linkage average", cex = 0.8)
plot(hc_single, labels = tipo_cancer, ylab = "", xlab = "", sub = "",
     main = "Linkage single", cex = 0.8)
```





La elección del tipo de *linkage* influye de forma notable en los dendrogramas obtenidos. Por lo general, *single linkage* tiende a formar *clusters* muy grandes en los que las observaciones individuales se unen una a una. El completo y *average* suele generar dendrogramas más compensados con *clusters* más definidos, tal como ocurre en este ejemplo.

A pesar de que la agrupación no es perfecta, los *clusters* tienden a segregar bastante bien las líneas celulares procedentes de *leukemia*, *melanoma* y *renal*. Véase los aciertos cuando el dendrograma se corta a una altura tal que genera 4 *clusters*.

```
clusters <- cutree(tree = hc_completo, k = 4)
table(clusters, tipo_cancer, dnn = list("clusters", "tipo de cáncer"))
```

```
##          tipo de cáncer
## clusters BREAST CNS COLON K562A-repro K562B-repro LEUKEMIA MCF7A-repro
##        1      2   3     2           0           0        0           0
##        2      3   2     0           0           0        0           0
##        3      0   0     0           1           1        6           0
##        4      2   0     5           0           0        0           1
##          tipo de cáncer
## clusters MCF7D-repro MELANOMA NSCLC OVARIAN PROSTATE RENAL UNKNOWN
##        1           0        8     8       6        2     8       1
##        2           0        0     1       0        0     1       0
##        3           0        0     0       0        0     0       0
##        4           1        0     0       0        0     0       0
```

Todas las líneas celulares de *leukemia* caen en el *cluster 3*, todas las de *melanoma* y *ovarian* en el 1. Las líneas celulares de *breast* son las más distribuidas (heterogéneas en su perfil genético) ya que están presentes en los *clusters* 1, 2 y 4.

Nota: de los 6830 genes medidos, muchos pueden estar aportando información redundante o puede que no varíen lo suficiente como para contribuir al modelo. Con el fin de eliminar todo ese ruido y mejorar los resultados del *clustering*, es aconsejable filtrar aquellos genes cuya expresión no supere un mínimo de varianza. Del mismo modo, puede ser útil evaluar si aplicando un *Principal Component Analysis* se consigue capturar la mayor parte de la información en unas pocas componentes y utilizarlas como variables de *clustering*. Este es un problema muy importante en la aplicación del *clustering* a perfiles genéticos. En la siguiente sección se describe con más detalle cómo solucionarlo.

Hierarchical K-means clustering

Idea intuitiva

K-means es uno de los métodos de *clustering* más utilizados y cuyos resultados son satisfactorios en muchos escenarios, sin embargo, como se ha explicado en apartados anteriores, sufre las limitaciones de necesitar que se especifique el número de *clusters* de antemano y de que sus resultados puedan variar en función de la iniciación aleatoria. Una forma de contrarrestar estos dos problemas es combinando el *K-means* con el *hierarchical clustering*. Los pasos a seguir son los siguientes:

1. Aplicar *hierarchical clustering* a los datos y cortar el árbol en *k clusters*. El número óptimo puede elegirse de forma visual o con cualquiera de los métodos explicados en la sección *Número óptimo de clusters*.

2. Calcular el centro (por ejemplo, la media) de cada *cluster*.

3. Aplicar *k-means clustering* empleando como centroides iniciales los centros calculados en el paso 2.

El algoritmo de *K-means* tratará de mejorar la agrupación hecha por el *hierarchical clustering* en el paso 1, de ahí que las agrupaciones finales puedan variar respecto a las iniciales.

Ejemplo

El set de datos `USArrests` *contiene información sobre el número de delitos (asaltos, asesinatos y secuestros) junto con el porcentaje de población urbana para cada uno de los 50 estados de USA. Se pretende estudiar si existe una agrupación subyacente de los estados empleando Hierarchical K-means clustering.*

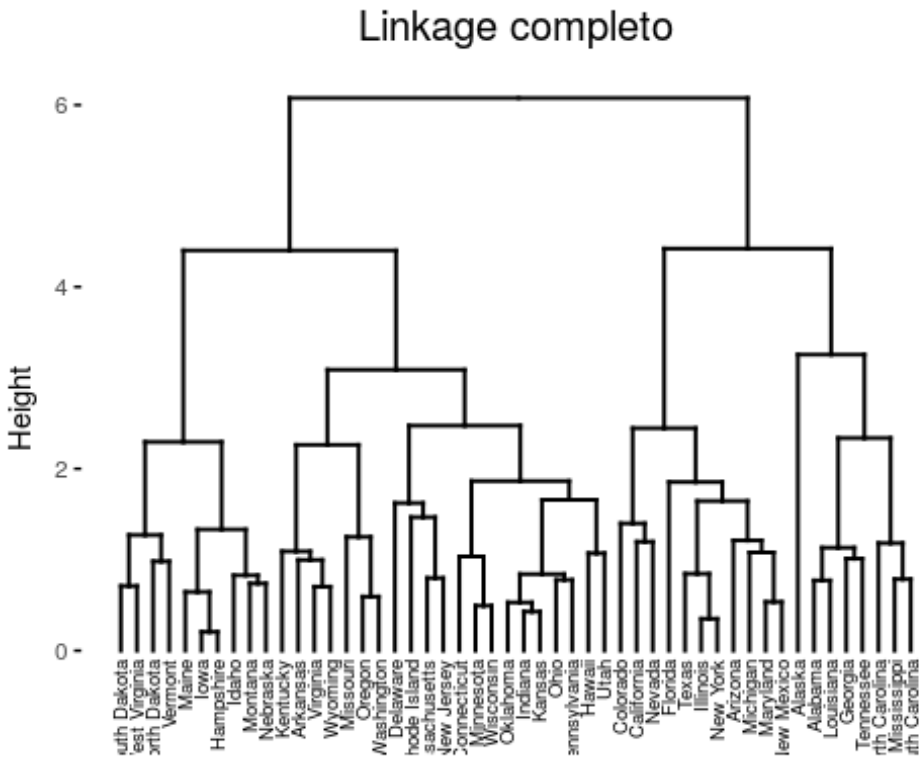
```
data("USArrests")
```

Como la magnitud de los valores difiere notablemente entre variables, se procede a escalarlas antes de aplicar el *clustering*.

```
datos <- scale(USArrests)
```

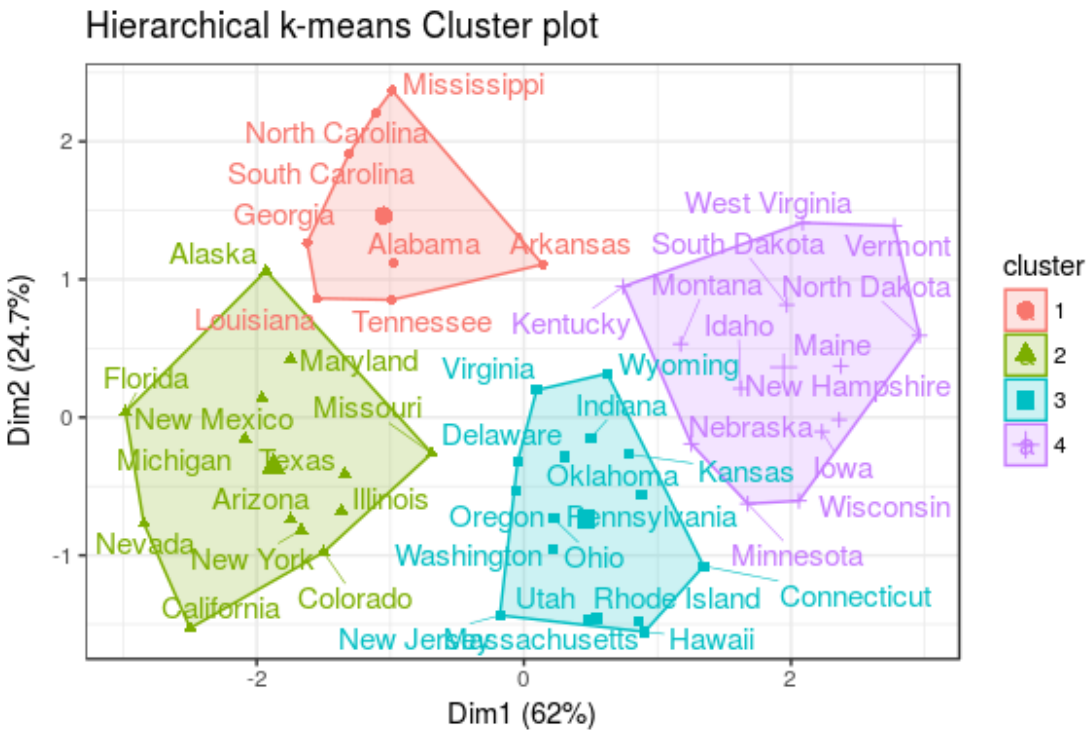
La función `hkmeans()` del paquete `factoextra` permite aplicar el método *hierarchical K-means clustering* de forma muy similar a la función estándar `kmeans()`.

```
library(factoextra)
# Se obtiene el dendrograma de hierarchical clustering para elegir el número de
# clusters.
set.seed(101)
hc_completo <- hclust(d = dist(x = datos, method = "euclidean"),
                      method = "complet")
fviz_dend(x = hc_completo, cex = 0.5, main = "Linkage completo") +
  theme(plot.title =  element_text(hjust = 0.5, size = 15))
```



Empleando la representación del dendrograma se considera que existen 4 grupos.

```
hkmeans_cluster <- hkmeans(x = datos, hc.metric = "euclidean",
                           hc.method = "complete", k = 4)
fviz_cluster(object = hkmeans_cluster, pallete = "jco", repel = TRUE) +
  theme_bw() + labs(title = "Hierarchical k-means Cluster plot")
```



Fuzzy clustering

Idea intuitiva

Los métodos de *clustering* descritos hasta ahora (*K-means, hierarchical, K-medoids, CLARA*...) asignan cada observación únicamente a un *cluster*, de ahí que también se conozcan como *hard clustering*. Los métodos de *fuzzy clustering o soft clustering* se caracterizan porque cada observación puede pertenecer potencialmente a varios *clusters*, en concreto, cada observación tiene asignado un grado de pertenencia a cada uno de los *cluster*.

Fuzzy c-means (FCM) es uno de los algoritmos más empleado para generar *fuzzy clustering*. Se asemeja en gran medida al algoritmo de *k-means* pero con dos diferencias: el cálculo de los centroides de los *clusters* y que devuelve para cada observación la probabilidad de pertenecer a cada *cluster*. La definición de centroide empleada por *c-means* es: la media de todas las observaciones del set de datos ponderada por la probabilidad de pertenecer a al *cluster*.

Ejemplo

El set de datos `USArrests` *contiene información sobre el número de delitos (asaltos, asesinatos y secuestros) junto con el porcentaje de población urbana para cada uno de los 50 estados de USA. Se pretende estudiar si existe una agrupación subyacente de los estados empleando fuzzy clustering.*

```
data("USArrests")
```

Como la magnitud de los valores difiere notablemente entre variables, se procede a escalarlas antes de aplicar el *clustering*.

```
datos <- scale(USArrests)
```

La función `fanny()` (*fuzzy analysis*) del paquete cluster permite aplicar el algoritmo *c-means clustering*.

```
library(cluster)
fuzzy_cluster <- fanny(x = datos, diss = FALSE, k = 3, metric = "euclidean",
                       stand = FALSE)
```

El objeto devuelto `fanny()` incluye entre sus elementos: una matriz con el grado de pertenencia de cada observación a cada *cluster* (las columnas son los *clusters* y las filas las observaciones).

```
head(fuzzy_cluster$membership)
```

```
##                [,1]      [,2]      [,3]
## Alabama    0.4676004 0.3144516 0.2179480
## Alaska     0.4278809 0.3178707 0.2542484
## Arizona    0.5092197 0.2945668 0.1962135
## Arkansas   0.2934077 0.3787718 0.3278205
## California 0.4668527 0.3084149 0.2247324
## Colorado   0.4542018 0.3236683 0.2221299
```

El coeficiente de partición *Dunn* normalizado y sin normalizar. Valores normalizados próximos a 0 indican que la estructura tiene un alto nivel *fuzzy* y valores próximos a 1 lo contrario.

```
fuzzy_cluster$coeff
```

```
## dunn_coeff normalized
## 0.37371071 0.06056606
```

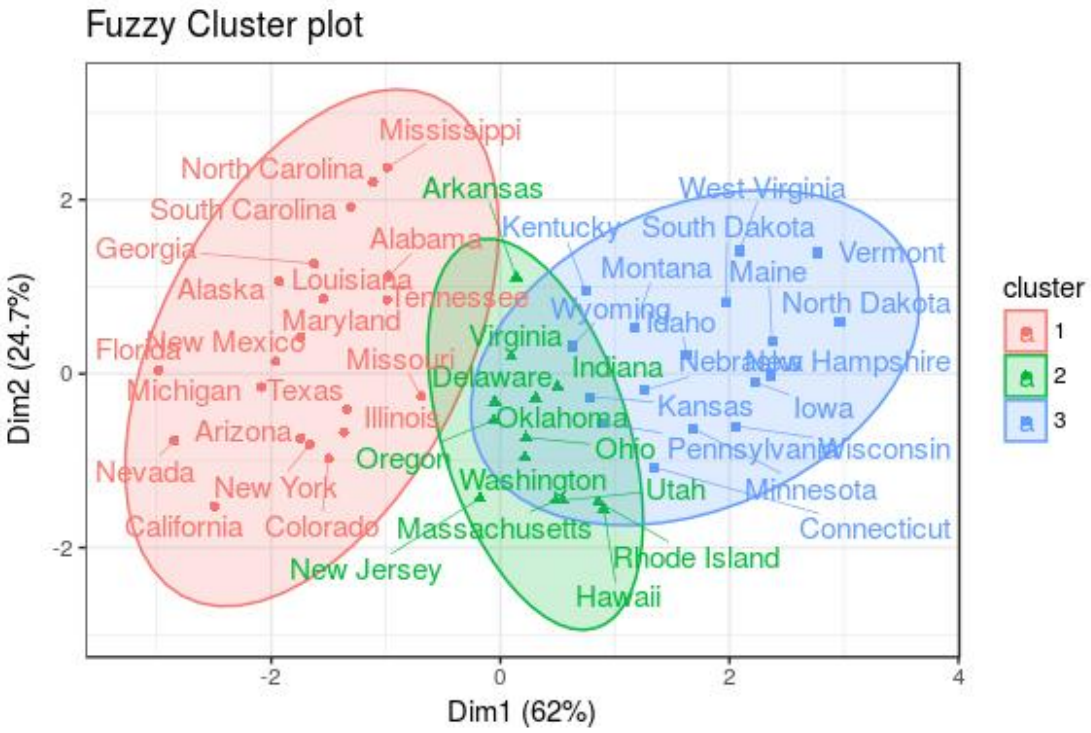
El *cluster* al que se ha asignado mayoritariamente cada observación.

```
head(fuzzy_cluster$clustering)
```

```
##    Alabama     Alaska    Arizona   Arkansas California   Colorado
##          1          1          1          2          1          1
```

Para obtener una representación gráfica del *clustering* se puede emplear la función `fviz_cluster()`.

```
library(factoextra)
fviz_cluster(object = fuzzy_cluster, repel = TRUE, ellipse.type = "norm",
             pallete = "jco") + theme_bw() + labs(title = "Fuzzy Cluster plot")
```



Model based clustering

Idea intuitiva

El *clustering* basado en modelos considera que las observaciones proceden de una distribución que es a su vez una combinación de dos o más componentes (*clusters*), cada uno con una distribución propia. En principio, cada *cluster* puede estar descrito por cualquier función de densidad, pero normalmente se asume que siguen una distribución multivariante normal. Para estimar los parámetros que definen la función de distribución de cada *cluster* (media y matriz de covarianza si se asume que son de tipo normal) se recurre al algoritmo de *Expectation-Maximization (EM)*. Este resuelve distintos modelos en los que el volumen, forma y orientación de las distribuciones pueden considerarse iguales para todos los *clusters* o distintas para cada uno. Por ejemplo, un posible modelo es: volumen constante, forma variable, orientación variable.

El paquete `mclust` emplea *maximum likelihood* para ajustar todos estos modelos con distinto número *k* de *clusters* y selecciona el mejor en base al *Bayesian Information Criterion (BIC)*.

Ejemplo

El set de datos `diabetes` *del paquete* `mclust` *contiene 3 parámetros sanguíneos medidos en 145 pacientes con 3 tipos distintos de diabetes. Se pretende emplear model-based-clustering para encontrar las agrupaciones.*

```
library(mclust)
data("diabetes")
head(diabetes)
```

```
##    class glucose insulin sspg
## 1 Normal      80     356  124
## 2 Normal      97     289  117
## 3 Normal     105     319  143
## 4 Normal      90     356  199
## 5 Normal      90     323  240
## 6 Normal      86     381  157
```

```
# Estandarización de variables
datos <- scale(diabetes[, -1])
# Model-based-clustering
model_clustering <- Mclust(data = datos, G = 1:10)
summary(model_clustering)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 3
components:
##
##  log.likelihood   n df       BIC       ICL
##       -169.0918 145 29 -482.5089 -501.4368
##
## Clustering table:
##  1  2  3
## 81 36 28
```

El algoritmo de ajuste selecciona como mejor modelo el formado por 3 *clusters*, cada uno con forma elipsoidal y con *volume*, *shape* y *orientation* propias.

El *clustering* basado en modelos es de tipo *fuzzy*, es decir, para cada observación se calcula un grado de pertenencia a cada *cluster* y se asigna finalmente al que mayor valor tiene.

```
# Grado de asignación a cada cluster
head(model_clustering$z)
```

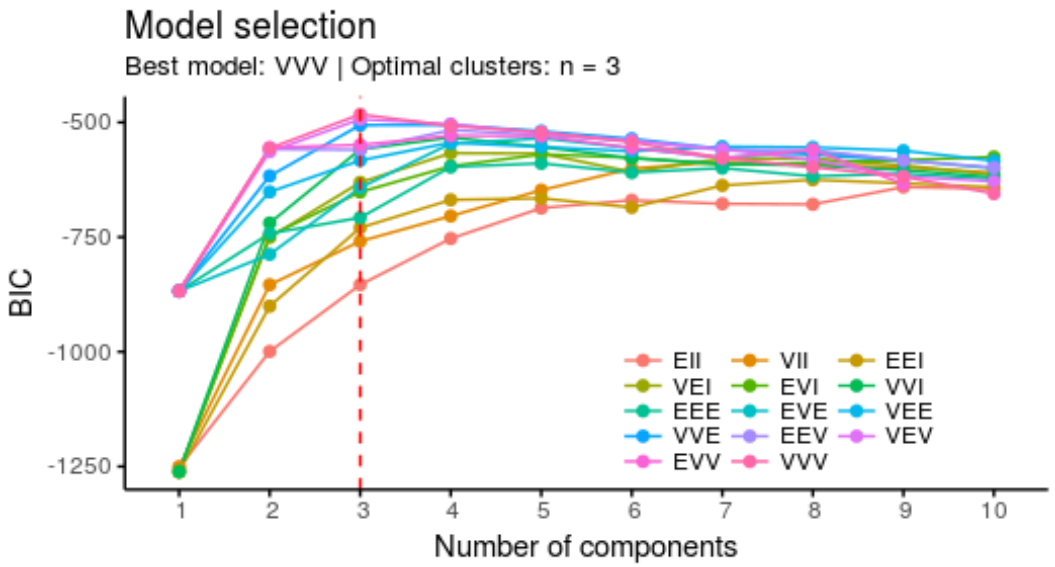
```
##        [,1]        [,2]         [,3]
## 1 0.9907874 0.008870671 3.419770e-04
## 2 0.9824097 0.017586177 4.107625e-06
## 3 0.9779945 0.021948580 5.690665e-05
## 4 0.9777076 0.022077703 2.147003e-04
## 5 0.9217797 0.078151587 6.868319e-05
## 6 0.9864768 0.012835588 6.876134e-04
```

```
# Clasificación final
head(model_clustering$classification)
```

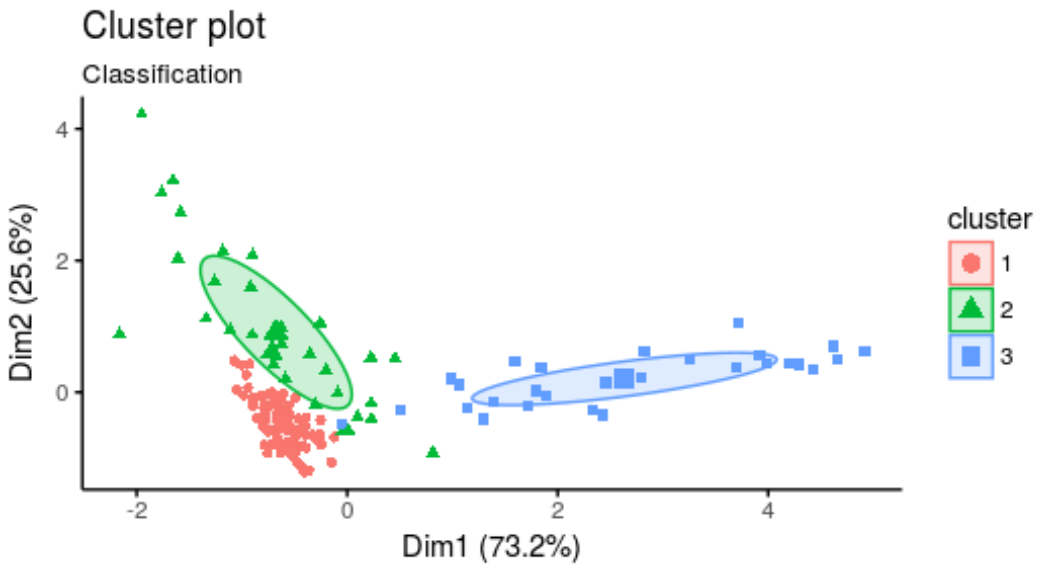
```
## 1 2 3 4 5 6
## 1 1 1 1 1 1
```

La visualización del *clustering* puede hacerse mediante la función `plot.Mclust()` o mediante `fviz_mclust()`.

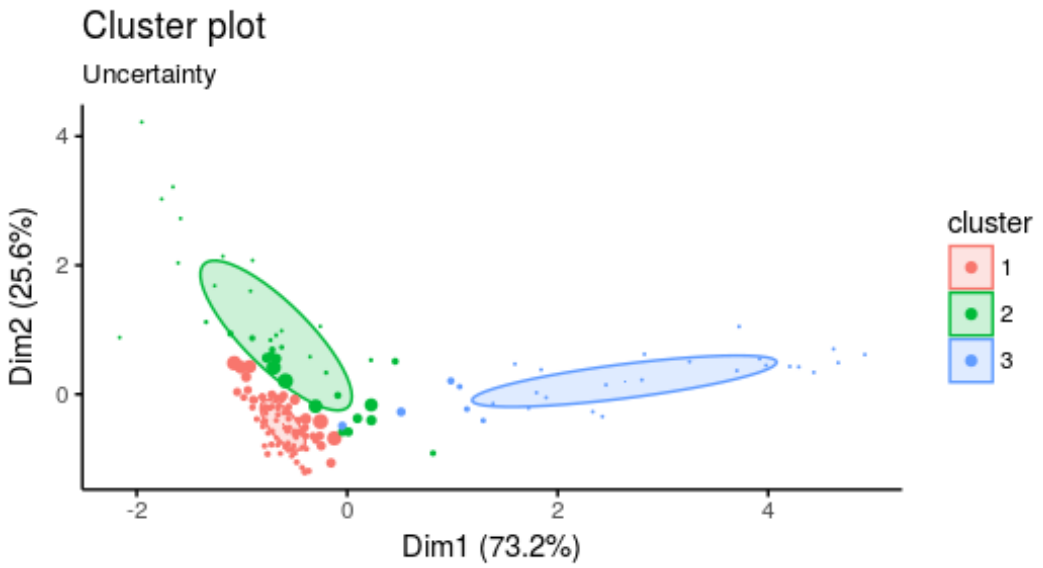
```
library(factoextra)
# Curvas del valor BIC en función del número de clusters para cada modelo.
# Atención al orden en el que se muestra la variable horizontal, por defecto es
# alfabético.
fviz_mclust(object = model_clustering, what = "BIC", pallete = "jco") +
  scale_x_discrete(limits = c(1:10))
```




```
# Clusters
fviz_mclust(model_clustering, what = "classification", geom = "point",
            pallete = "jco")
```



```
# Certeza de las clasificaciones. Cuanto mayor el tamaño del punto menor la
# seguridad de la asignación
fviz_mclust(model_clustering, what = "uncertainty", pallete = "jco")
```

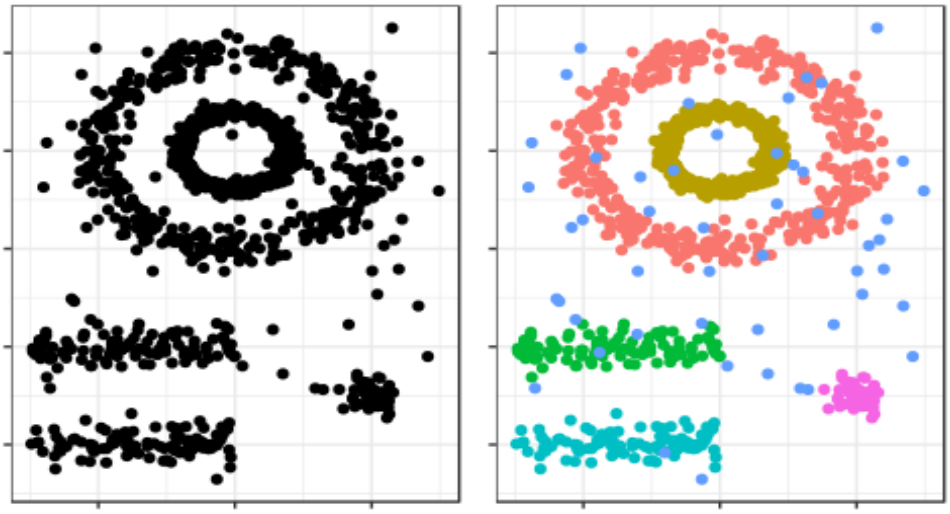


Density based clustering (DBSCAN)

Idea intuitiva

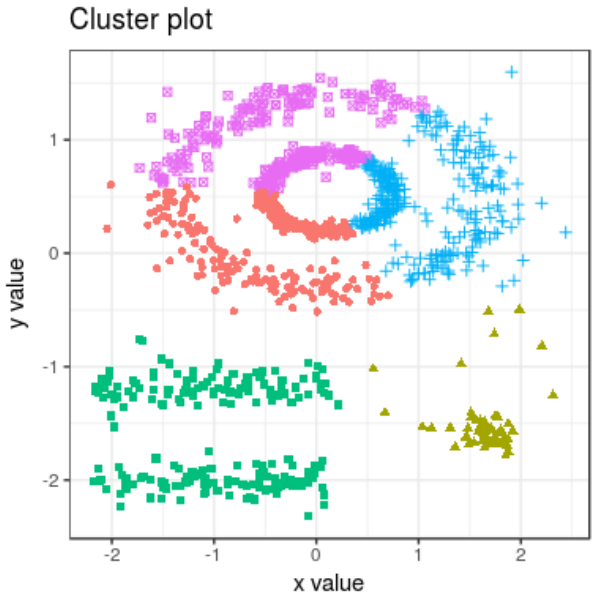
Density-based spatial clustering of applications with noise (DBSCAN) fue presentado en 1996 por Ester et al. como una forma de identificar *clusters* siguiendo el modo intuitivo en el que lo hace el cerebro humano, identificando regiones con alta densidad de observaciones separadas por regiones de baja densidad.

Véase la siguiente representación bidimensional de los datos `multishape` del paquete `factoextra`.



El cerebro humano identifica fácilmente 5 agrupaciones y algunas observaciones aisladas (ruido). Véanse ahora los *clusters* que se obtienen si se aplica, por ejemplo, *K-means clustering*.

```
library(factoextra)
data("multishapes")
datos <- multishapes[, 1:2]
set.seed(321)
km_clusters <- kmeans(x = datos, centers = 5, nstart = 50)
fviz_cluster(object = km_clusters, data = datos, geom = "point", ellipse = FALSE,
             show.clust.cent = FALSE, pallete = "jco") +
  theme_bw() +
  theme(legend.position = "none")
```



Los *clusters* generados distan mucho de representar las verdaderas agrupaciones. Esto es así porque los métodos de *partitioining clustering* como *k-means*, *hierarchical*, *k-medoids*, *c-means*... son buenos encontrando agrupaciones con forma esférica o convexa que no contengan un exceso de *outliers* o ruido, pero fallan al tratar de identificar formas arbitrarias. De ahí que el único *cluster* que se corresponde con un grupo real sea el amarillo.

DBSCAN evita este problema siguiendo la idea de que, para que una observación forme parte de un *cluster*, tiene que haber un mínimo de observaciones vecinas dentro de un radio de proximidad y de que los *clusters* están separados por regiones vacías o con pocas observaciones.

El algoritmo *DBSCAN* necesita dos parámetros:

- *Epsilon* (ϵ): radio que define la región vecina a una observación, también llamada ϵ-*neighborhood*.
- *Minimum points* (*minPts*): número mínimo de observaciones dentro de la región *epsilon*.

Empleando estos dos parámetros, cada observación del set de datos se puede clasificar en una de las siguientes tres categorías:

- *Core point*: observación que tiene en su ϵ-*neighborhood* un número de observaciones vecinas igual o mayor a *minPts*.
- *Border point*: observación no satisface el mínimo de observaciones vecinas para ser *core point* pero que pertenece al ϵ-*neighborhood* de otra observación que sí es *core point*.
- *Noise* u *outlier*: observación que no es *core point* ni *border point*.

Por último, empleando las tres categorías anteriores se pueden definir tres niveles de conectividad entre observaciones:

- Directamente alcanzable (*direct density reachable*): una observación A es directamente alcanzable desde otra observación B si A forma parte del ϵ-*neighborhood* de B y B es un *core point*. Por definición, las observaciones solo pueden ser directamente alcanzables desde un *core point*.
- Alcanzable (*density reachable*): una observación A es alcanzable desde otra observación B si existe una secuencia de *core points* que van desde B a A.
- Densamente conectadas (*density conected*): dos observaciones A y B están densamente conectadas si existe una observación *core point* C tal que A y B son alcanzables desde C.

La siguiente imagen muestra las conexiones existentes entre un conjunto de observaciones si se emplea $minPts = 4$. La observación A y el resto de observaciones marcadas en rojo son *core points*, ya que todas ellas contienen al menos 4 observaciones vecinas (incluyéndose a ellas mismas) en su ϵ-*neighborhood*. Como todas son alcanzables entre ellas, forman un

cluster. Las observaciones B y C no son *core points* pero son alcanzables desde A a través de otros *core points*, por lo tanto, pertenecen al mismo *cluster* que A. La observación N no es ni un *core point* ni es directamente alcanzable, por lo que se considera como ruido.

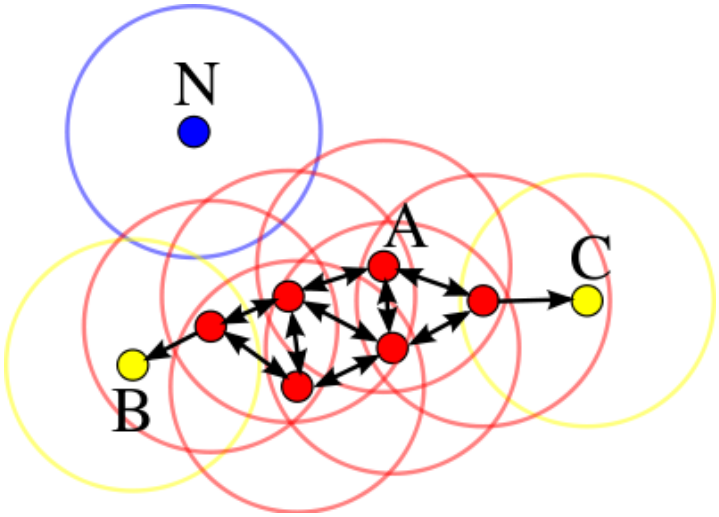


Imagen obtenida de Wikipedia

Algoritmo

1. Para cada observación x_i calcular la distancia entre ella y el resto de observaciones. Si en su ϵ-*neighborhood* hay un número de observaciones $\geq minPts$ marcar la observación como *core point*, de lo contrario marcarla como *visitada*.

2. Para cada observación x_i marcada como *core point*, si todavía no ha sido asignada a ningún *cluster*, crear uno nuevo y asignarla a él. Encontrar recursivamente todas las observaciones densamente conectadas a ella y asignarlas al mismo *cluster*.

3. Iterar el mismo proceso para todas las observaciones que no hayan sido visitadas.

4. Aquellas observaciones que tras haber sido visitadas no pertenecen a ningún *cluster* se marcan como *outliers*.

Como resultado, todo *cluster* cumple dos propiedades: todos los puntos que forman parte de un mismo *cluster* están densamente conectados entre ellos y, si una observación A es densamente alcanzable desde cualquier otra observación de un *cluster*, entonces A también pertenece al *cluster*.

Selección de parámetros

Como ocurre en muchas otras técnicas estadísticas, en *DBSCAN* no existe una forma única y exacta de encontrar el valor adecuado de *epsilon* (ϵ) y $minPts$. A modo orientativo se pueden seguir las siguientes premisas:

- $minPts$: cuanto mayor sea el tamaño del set de datos, mayor debe ser el valor mínimo de observaciones vecinas. En el libro *Practical Guide to Cluster Analysis in R* recomiendan no bajar nunca de 3. Si los datos contienen niveles altos de ruido, aumentar $minPts$ favorecerá la creación de *clusters* significativos menos influenciados por *outliers*.

- *epsilon*: una buena forma de escoger el valor de ϵ es estudiar las distancias promedio entre las $k = minPts$ observaciones más próximas. Al representar estas distancias en función de ϵ, el punto de inflexión de la curva suele ser un valor óptimo. Si el valor de ϵ escogido es muy pequeño, una proporción alta de las observaciones no se asignarán a ningún *cluster*, por el contrario, si el valor es demasiado grande, la mayoría de observaciones se agruparán en un único *cluster*.

Ventajas de DBSCAN

- No requiere que el usuario especifique el número de *clusters*.
- Es independiente de la forma que tengan los *clusters*, no tienen por qué ser circulares.
- Puede identificar *outliers*, por lo que los *clusters* generados no se influenciados por ellos.

Desventajas de DBSCAN

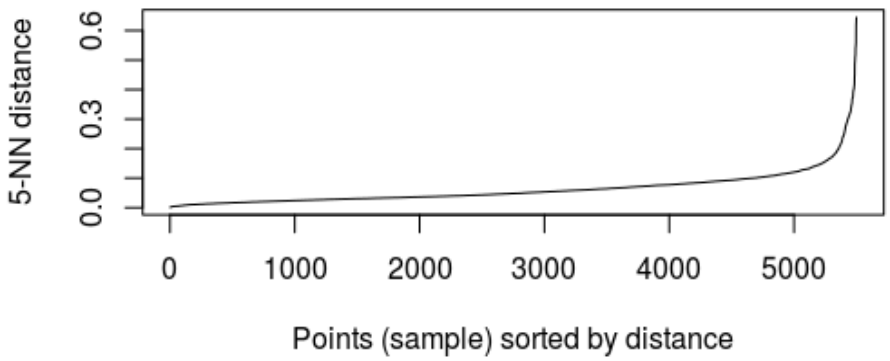
- No es un método totalmente determinístico: los *border points* que son alcanzables desde más de un *cluster* pueden asignarse a uno u otro dependiendo del orden en el que se procesen los datos.
- No genera buenos resultados cuando la densidad de los grupos es muy distinta, ya que no es posible encontrar los parámetros ϵ y *minPts* que sirvan para todos a la vez.

Ejemplo

El set de datos `multishape` *del paquete* `factoextra` *contiene observaciones que pertenecen a 5 grupos distintos junto con cierto ruido (outliers). Como se espera que la distribución espacial de los grupos no sea esférica, se aplica el método de clustering DBSCAN.*

En `R` existen dos paquetes con funciones que permiten aplicar el algoritmo *DBSCAN*: `fpc` y `dbscan`. El segundo contiene una modificación del algoritmo original que lo hace más rápido. La función `kNNdistplot` del paquete `dbscan` calcula y representa las *k*-distancias para ayudar a identificar el valor óptimo de *epsilon*.

```
library(fpc)
library(dbscan)
library(factoextra)
data("multishapes")
datos <- multishapes[, 1:2]

# Selección del valor óptimo de epsilon. Como valor de minPts se emplea 5.
dbscan::kNNdistplot(datos, k = 5)
```

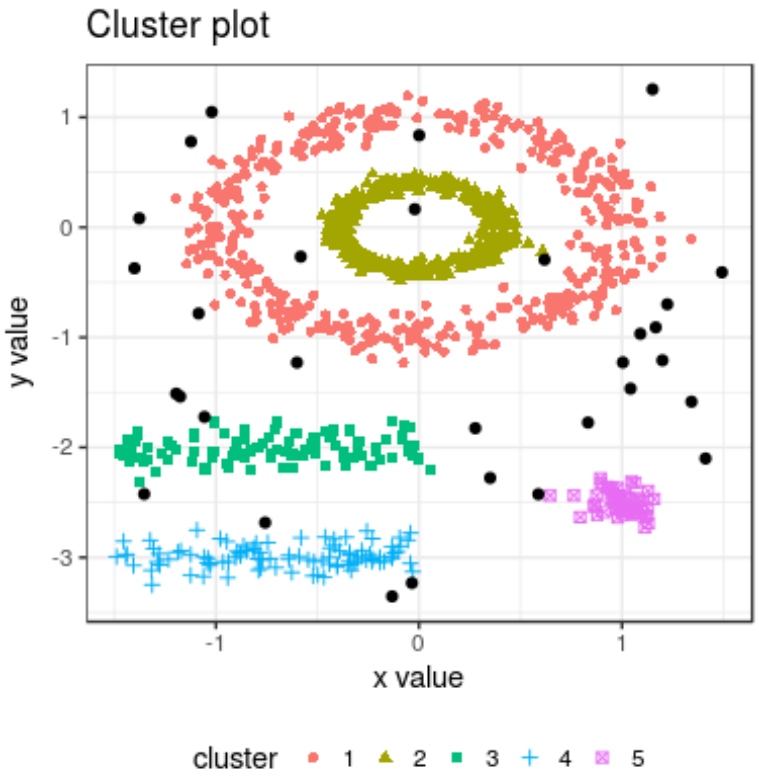


La curva tiene el punto de inflexión en torno a 0.15, por lo que se escoge este valor como *epsilon* para *DBSCAN*.

```
set.seed(321)
# DBSCAN con epsilon = 0.15 y minPts = 5
dbscan_cluster <- fpc::dbscan(data = datos, eps = 0.15, MinPts = 5)

# Resultados de la asignación
head(dbscan_cluster$cluster)
```

```
## [1] 1 1 1 1 1 1
```

```
# Visualización de los clusters
fviz_cluster(object = dbscan_cluster, data = datos, stand = FALSE,
             geom = "point", ellipse = FALSE, show.clust.cent = FALSE,
             pallete = "jco") +
  theme_bw() +
  theme(legend.position = "bottom")
```



Clustering de perfiles genéticos

Uno de los estudios más frecuentes en el ámbito de los perfiles de expresión genética es lo que se conoce como *sample based clustering*, que consiste en agrupar muestras (tumores, pacientes...) en base a los niveles de expresión de múltiples genes. Aunque los métodos de *clustering* convencionales como *K-means* o *hierarchical clustering* pueden aplicarse directamente empleando todos los genes disponibles (varios miles normalmente), se ha demostrado que, por lo general, solo un pequeño subgrupo de estos está realmente relacionado con la diferenciación de los grupos. A estos genes se les conoce como genes informativos. El resto de genes son irrelevantes en cuanto a la clasificación de interés y por lo tanto solo aportan ruido, perjudicando en gran medida la calidad de los resultados de *clustering*.

Se han desarrollado métodos específicos con la finalidad de identificar los genes informativos y reducir así la dimensionalidad genética. Estos métodos se agrupan en dos categorías: *clustering based on supervised informative gene selection* y *unsupervised clustering and informative gene selection*.

Clustering based on supervised informative gene selection

Esta aproximación combina la naturaleza *unsupervided* del *clustering* con métodos *supervised* de clasificación. Asumiendo que se conoce el verdadero grupo al que pertenece cada muestra o al menos parte de ellas, por ejemplo, pacientes sanos vs enfermos, se identifican aquellos genes cuya expresión está más diferenciada entre ambos grupos. Una vez identificados, se emplean únicamente estos genes informativos para realizar *clustering* sobre las muestras ya conocidas u otras nuevas. Se utiliza con mucha frecuencia en biología.

Unsupervised clustering and informative gene selection

Esta aproximación puede seguir dos estrategias distintas. Una en la que el proceso de selección de genes informativos y el *clustering* se llevan a cabo como procesos independientes (*unsupervised gene selection*) y otra en la que ambos procesos se combinan de forma simultánea e iterativa (*interrelated clustering*).

Unsupervised gene selection: En primer lugar, se reduce la dimensionalidad de los datos y a continuación se aplican los algoritmos de *clustering*. Las dos formas más empleadas para conseguir la reducción en el número de variables (genes) son:

- El uso de *Principal Component Analysis*, tratando de capturar la mayoría de la información en unas pocas componentes.
- La selección de aquellos genes que muestran mayor varianza en la matriz de expresión.

Si bien esta estrategia es sencilla y a menudo útil, requiere asumir que los genes informativos para la agrupación que se quiere llevar a cabo tienen mayor varianza que los genes irrelevantes, cosa que no siempre tiene porque ser cierta.

Interrelated clustering: Si se analiza con detenimiento el problema de la selección de genes informativos y el *clustering* de muestras, se llega a la conclusión de que ambos procesos están estrechamente relacionados. Por un lado, una vez que los genes informativos han sido identificados, es relativamente sencillo aplicar con éxito los algoritmos de *clustering* convencionales. Por otro lado, una vez que las muestras han sido correctamente agrupadas, es sencillo identificar los genes informativos empleando test estadísticos tales como el *t-test* o *anova*. Estos dos hechos se pueden combinar de modo iterativo de forma que uno retroalimente al otro. El proceso empieza generando agrupaciones iniciales por *clustering* empleando todos los genes disponibles. Si bien esta partición no es exactamente la real, permite identificar potenciales genes informativos (genes cuya expresión es diferente entre los grupos generados). En la siguiente iteración se repite el *clustering* pero empleando únicamente

los genes identificados como informativos en el paso anterior. Con cada repetición del proceso, las particiones se irán aproximando cada vez más a la estructura real de las muestras y la selección final de genes convergerá en aquellos que más influyen en la agrupación resultante.

Comparación de dendrogramas

Dados los muchos parámetros que se tienen que determinar a lo largo del proceso de *hierarchical clustering*, es frecuente que el analista genere varios dendrogramas para compararlos y escoger finalmente uno de ellos. Existen varias formas de estudiar las diferencias entre dendrogramas, dos de las más utilizadas son: la comparación visual y el cálculo de correlación entre dendrogramas. Ambos métodos pueden aplicarse con las funciones `tanglegram()` y `cor.dendlist()` del paquete dendextend.

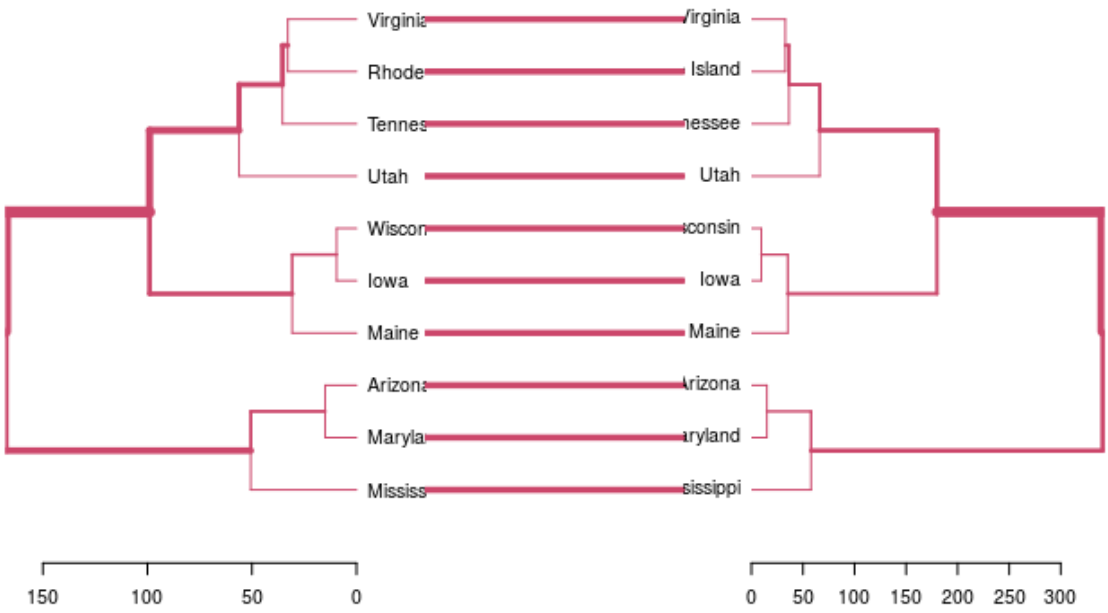
Se emplea el set de datos `USArrests` *con el objetivo de generar dendrogramas que agrupen los diferentes estados por su similitud en el porcentaje de asesinatos, asaltos, secuestros y proporción de población rural. Se comparan los resultados obtenidos empleando linkage average y ward.D2.*

```
# Para facilitar la interpretación se simplifican los dendrogramas empleando
# únicamente 10 estados
library(dendextend)
set.seed(123)
datos <- USArrests[sample(1:50, 10), ]

# Cálculo matriz de distancias
mat_dist <- dist(x = datos, method = "euclidean")

# Cálculo de hierarchical clustering
hc_average <- hclust(d = mat_dist, method = "average")
hc_ward    <- hclust(d = mat_dist, method = "ward.D2")

# Las funciones del paquete dendextend trabajan con objetos de tipo dendrograma,
# para obtenerlos se emplea la función as.dendogram()
dend_1 <- as.dendrogram(hc_average)
dend_2 <- as.dendrogram(hc_ward)
```

Comparación visual

La función `tanglegram()` representa dos dendrogramas a la vez, enfrentados uno al otro, y conecta las hojas terminales con líneas. Los nodos que aparecen solo en uno de los dendrogramas, es decir, que están formados por una combinación de observaciones que no se da en el otro, aparecen destacados con líneas discontinuas.

```
tanglegram(dend1 = dend_1, dend2 = dend_2, highlight_distinct_edges = TRUE,
           common_subtrees_color_branches = TRUE)
```



Comparación por correlación

Con la función `cor.dendlist()` se puede calcular la matriz de correlación entre dendrogramas basada en las distancias de *Cophenetic* o *Baker*.

```
# Se almacenan los dendrogramas a comparar en una lista
list_dendrogramas <- dendlist(dend_1, dend_2)
cor.dendlist(dend = list_dendrogramas, method = "cophenetic")
```

```
##           [,1]      [,2]
## [1,] 1.0000000 0.9977162
## [2,] 0.9977162 1.0000000
```

```
# Se pueden obtener comparaciones múltiples incluyendo más de dos dendrogramas en
# la lista pasada como argumento
dend_1 <- datos %>% dist(method = "euclidean") %>% hclust(method = "average") %>%
          as.dendrogram()
dend_2 <- datos %>% dist(method = "euclidean") %>% hclust(method = "ward.D2") %>%
          as.dendrogram()
dend_3 <- datos %>% dist(method = "euclidean") %>% hclust(method = "single") %>%
          as.dendrogram()
dend_4 <- datos %>% dist(method = "euclidean") %>% hclust(method = "complete") %>%
          as.dendrogram()
list_dendrogramas <- dendlist("average" = dend_1, "ward.D2" = dend_2,
                              "single" = dend_3, "complete" = dend_4)
cor.dendlist(dend = list_dendrogramas, method = "cophenetic") %>% round(digits= 3)
```

```
##          average ward.D2 single complete
## average    1.000   0.998  0.977    0.959
## ward.D2    0.998   1.000  0.964    0.952
## single     0.977   0.964  1.000    0.943
## complete   0.959   0.952  0.943    1.000
```

```
# Si solo se comparan dos dendrogramas se puede emplear la función cor_cophenetic
cor_cophenetic(dend1 = dend_1, dend2 = dend_2)
```

```
## [1] 0.9977162
```

Validación del clustering

Los métodos de *clustering* tienen la propiedad de encontrar agrupaciones en cualquier set de datos, independientemente de que realmente existan o no dichos grupos en la población de la que proceden las observaciones. Por ejemplo, si se aplica el mismo método a una segunda muestra de la misma población ¿Se obtendrían los mismos grupos? Además, cada uno de los métodos de *clustering* da lugar a resultados distintos. La validación de *clusters* es el proceso por el cual se evalúa la veracidad de los grupos obtenidos. A modo general, este proceso consta de tres partes: estudio de la tendencia de *clustering*, elección del número óptimo de *clusters* y estudio de la calidad/significancia de los *clusters* generados.

Estudio de la tendencia de clustering

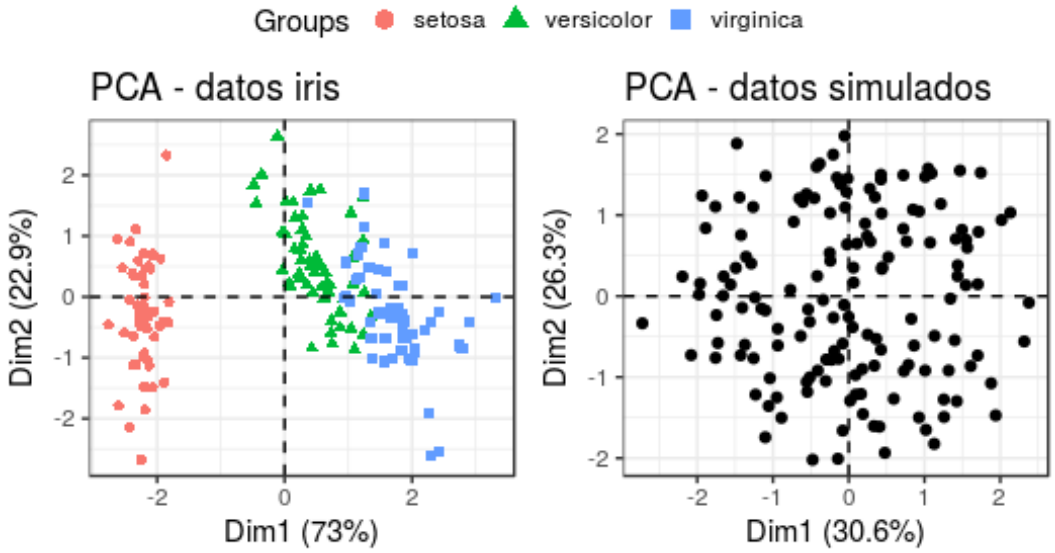
Antes de aplicar un método de *clustering* a los datos es conveniente evaluar si hay indicios de que realmente existe algún tipo de agrupación en ellos. A este proceso se le conoce como *assessing cluster tendecy* y puede llevarse a cabo mediante test estadísticos (*Hopkins statistic*) o de forma visual (*Visual Assessment of cluster Tendency*).

Para ilustrar la importancia de este pre-análisis inicial, se aplica clustering a dos sets de datos, uno que sí contiene grupos reales (`iris`) y otro aleatoriamente simulado que no.

```
library(purrr)

# Se elimina la columna que contiene la especie de planta
datos_iris <- iris[, -5]

# Se generan valores aleatorios dentro del rango de cada variable. Se utiliza la
# función map del paquete purrr.
datos_simulados <- map_df(datos_iris,
                          .f = function(x){runif(n = length(x),
                                                 min = min(x),
                                                 max = max(x))
                                           }
                          )

# Estandarización de los datos
datos_iris      <- scale(datos_iris)
datos_simulados <- scale(datos_simulados)
```

Una representación gráfica permite comprobar que el set de datos `iris` sí contiene grupos reales, mientras que los datos simulados no. Al haber más de dos variables es necesario reducir la dimensionalidad mediante un *Principal Component Analysis*.

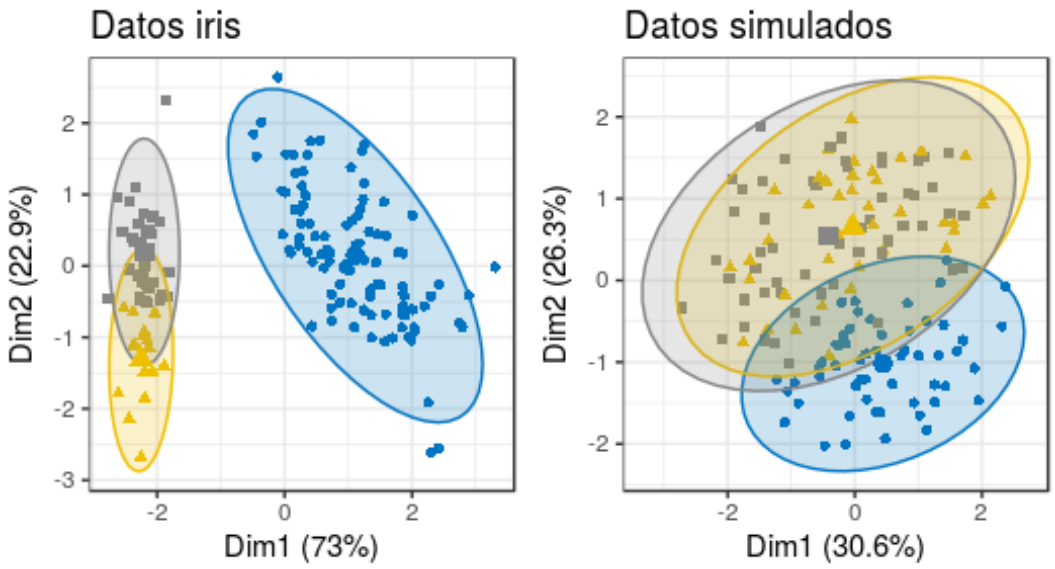
```
library(factoextra)
library(ggpubr)
pca_datos_iris      <- prcomp(datos_iris)
pca_datos_simulados <- prcomp(datos_simulados)
p1 <- fviz_pca_ind(X = pca_datos_iris, habillage = iris$Species,
                   geom = "point", title = "PCA - datos iris",
                   pallete = "jco") +
      theme_bw() + theme(legend.position = "bottom")
p2 <- fviz_pca_ind(X = pca_datos_simulados, geom = "point",
                   title = "PCA - datos simulados", pallete = "jco") +
      theme_bw() + theme(legend.position = "bottom")

ggarrange(p1, p2, common.legend = TRUE)
```



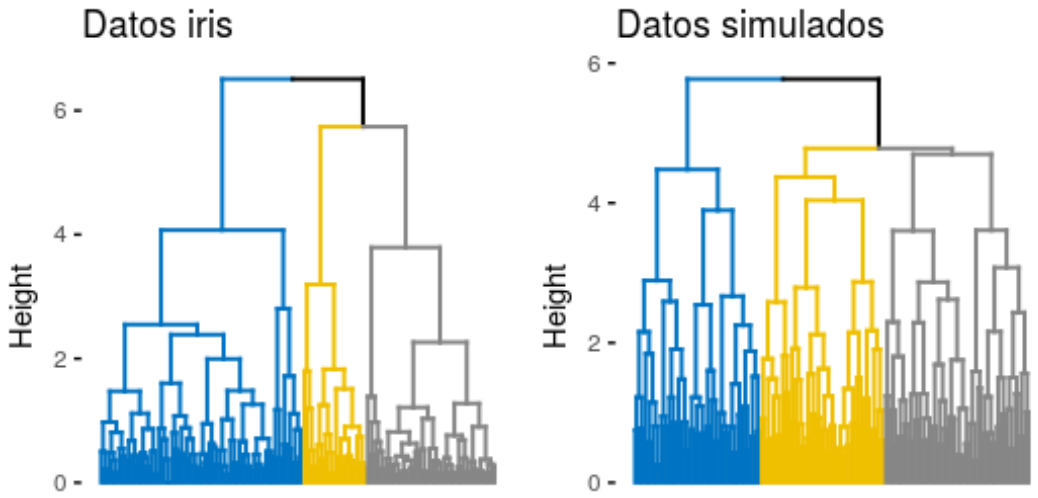
Véase que ocurre cuando se aplican métodos de *clustering* a estos dos sets de datos.

```
# K-means clustering
km_datos_iris <- kmeans(x = datos_iris, centers = 3)
p1 <- fviz_cluster(object = km_datos_iris, data = datos_iris,
                   ellipse.type = "norm", geom = "point", main = "Datos iris",
                   stand = FALSE, palette = "jco") +
      theme_bw() + theme(legend.position = "none")
km_datos_simulados <- kmeans(x = datos_simulados, centers = 3)
p2 <- fviz_cluster(object = km_datos_simulados, data = datos_simulados,
                   ellipse.type = "norm", geom = "point",
                   main = "Datos simulados", stand = FALSE, palette = "jco") +
      theme_bw() + theme(legend.position = "none")
```

```
# Hierarchical clustering
p3 <- fviz_dend(x = hclust(dist(datos_iris)), k = 3, k_colors = "jco",
                show_labels = FALSE, main = "Datos iris")
p4 <- fviz_dend(x = hclust(dist(datos_simulados)), k = 3, k_colors = "jco",
                show_labels = FALSE, main = "Datos simulados")

ggarrange(p1, p2)
```




```
ggarrange(p3, p4)
```



Ambos métodos de *clustering* crean agrupaciones en el set de datos simulado. De no analizarse con detenimiento, podrían darse por válidos estos grupos aun cuando realmente no existen. A continuación, se muestran dos métodos que ayudan a identificar casos como este y prevenir la utilización de *clustering* en escenarios en los que no tiene sentido hacerlo.

Hopkins statistics

El estadístico *Hopkins* permite evaluar la tendencia de *clustering* de un conjunto de datos mediante el cálculo de la probabilidad de que dichos datos procedan de una distribución uniforme, es decir, estudia la distribución espacial aleatoria de las observaciones. La forma de calcular este estadístico es la siguiente:

- Extraer una muestra uniforme de n observaciones (p_1,..., p_n) del set de datos estudiado.
- Para cada observación p_i seleccionada, encontrar la observación vecina más cercana p_j y calcular la distancia entre ambas, $x_i = dist(p_i, p_j)$.
- Simular un conjunto de datos de tamaño n (q_1,..., q_n) extraídos de una distribución uniforme con la misma variación que los datos originales.
- Para cada observación simulada q_i, encontrar la observación vecina más cercana q_j y calcular la distancia entre ambas, $y_i = dist(q_i, q_j)$.
- Calcular el estadístico *Hopkins (H)* como la media de las distancias de vecinos más cercanos en el set de datos simulados, dividida por la suma de las medias de las distancias vecinas más cercanas del set de datos original y el simulado.

$$H = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i}$$

Valores de H en torno a 0.5 indican que $\sum_{i=1}^{n} x_i$ y $\sum_{i=1}^{n} y_i$ son muy cercanos el uno al otro, es decir, que los datos estudiados se distribuyen uniformemente y que por lo tanto no tiene sentido aplicar *clustering*. Cuanto más se aproxime a 0 el estadístico H, más evidencias se tienen a favor de que existen agrupaciones en los datos y de que, si se aplica *clustering* correctamente, los grupos resultantes serán reales. La función `hopkins()` del paquete `clustertend` permite calcular el estadístico *Hopkins*.

```
library(clustertend)
set.seed(321)

# Estadístico H para el set de datos iris
hopkins(data = datos_iris, n = nrow(datos_iris) - 1)
```

```
## $H
## [1] 0.1842089
```

```
# Estadístico H para el set de datos simulado
hopkins(data = datos_simulados, n = nrow(datos_simulados) - 1)
```

```
## $H
## [1] 0.5153943
```

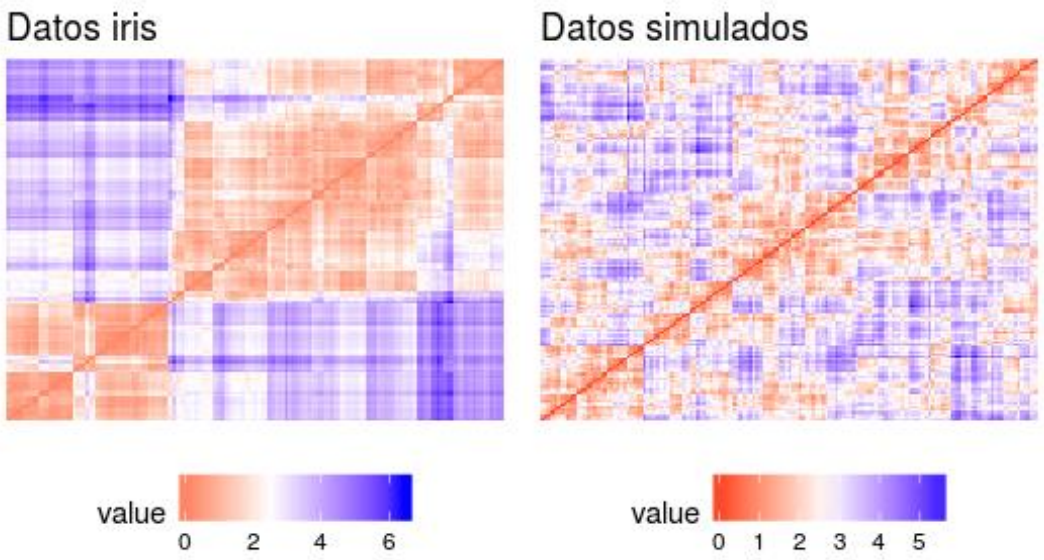
Los resultados muestran evidencias de que las observaciones del set de datos iris no siguen una distribución espacial uniforme, su estructura contiene algún tipo de agrupación. Por contra, el valor del estadístico *H* obtenido para el set de datos simulado es muy próximo a *0.5*, lo que indica que los datos están uniformemente distribuidos y desaconseja la utilización de métodos de *clustering*.

Visual Assessment of cluster Tendency (VAT)

VAT es método que permite evaluar visualmente si los datos muestran indicios de algún tipo de agrupación. La idea es sencilla:

- Se calcula una matriz de distancias euclídeas entre todos los pares de observaciones.
- Se reordena la matriz de distancias de forma que las observaciones similares están situadas cerca unas de otras (*ordered dissimilarity matrix*).
- Se representa gráficamente la matriz de distancias ordenada, empleando un gradiente de color para el valor de las distancias. Si existen agrupaciones subyacentes en los datos se forma un patrón de bloques cuadrados.

```
library(factoextra)
library(ggpubr)
dist_datos_iris      <- dist(datos_iris, method = "euclidean")
dist_datos_simulados <- dist(datos_simulados, method = "euclidean")

p1 <- fviz_dist(dist.obj = dist_datos_iris, show_labels = FALSE) +
      labs(title = "Datos iris") + theme(legend.position = "bottom")
p2 <- fviz_dist(dist.obj = dist_datos_simulados, show_labels = FALSE) +
      labs(title = "Datos simulados") + theme(legend.position = "bottom")

ggarrange(p1, p2)
```



El método *VAT* confirma que en el set de datos iris sí hay una estructura de grupos, mientras que, en los datos simulados, no.

Número óptimo de clusters

Determinar el número óptimo de *clusters* es uno de los pasos más complicados a la hora de aplicar métodos de *clustering*, sobre todo cuando se trata de *partitioning clustering*, donde el número se tiene que especificar antes de poder ver los resultados.

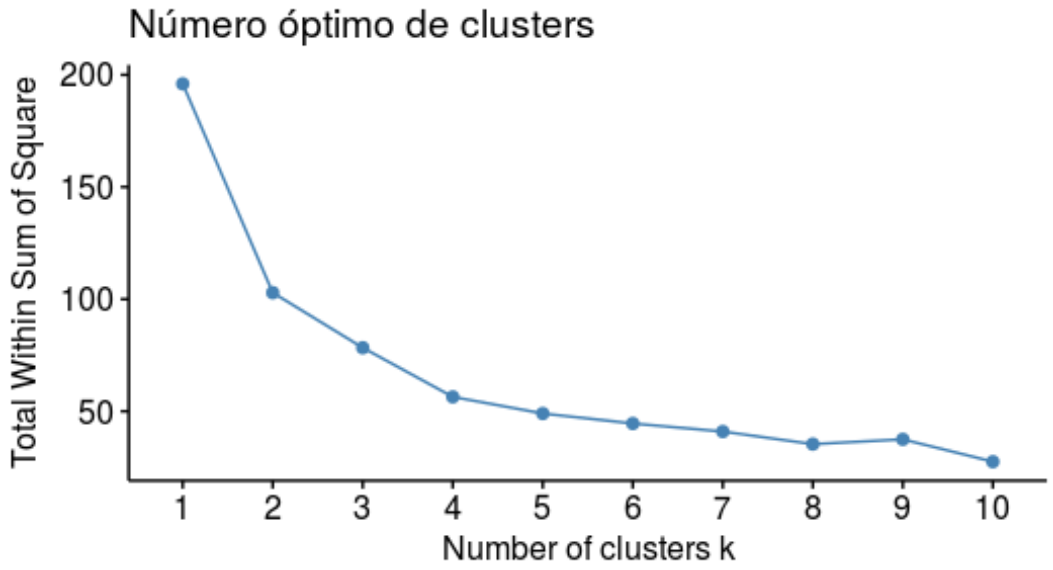
No existe una forma única de averiguar el número adecuado de *clusters*. Es un proceso bastante subjetivo que depende en gran medida del tipo de *clustering* empleado y de si se dispone de información previa sobre los datos con los que se está trabajando, por ejemplo, estudios anteriores pueden sugerir o acotar las posibilidades. A pesar de ello, se han desarrollado varias estrategias que ayudan en el proceso.

Elbow method

El método *Elbow* sigue una estrategia comúnmente empleada para encontrar el valor óptimo de un parámetro. La idea general es probar un rango de valores del parámetro en cuestión, representar gráficamente los resultados obtenidos con cada uno e identificar aquel punto de la curva a partir del cual la mejora deja de ser sustancial (principio de verosimilitud). En los casos de *partitioning clustering*, como por ejemplo *K-means*, las observaciones se agrupan de una forma tal que se minimiza la varianza total *intra-cluster*. El método *Elbow*

calcula la varianza total *intra-cluster* en función del número de *clusters* y escoge como óptimo aquel valor a partir del cual añadir más *clusters* apenas consigue mejoría. La función `fviz_nbclsut()` del paquete `factoextra` automatiza todo el proceso, empleando como medida de varianza *intra-cluster* la suma de residuos cuadrados internos (*wss*).

```
library(factoextra)
datos <- scale(USArrests)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "wss") +
  labs(title = "Número óptimo de clusters")
```



La curva indica que a partir de 4 *clusters* la mejora es mínima.

Average silhouette method

El método de *average silhouette* es muy similar al de *Elbow*, con la diferencia de que, en lugar minimizar el *total inter-cluster sum of squares (wss)*, se maximiza la media de los *silhouette coeficient* (s_i). Este coeficiente cuantifica cómo de buena es la asignación que se ha hecho de una observación comparando su similitud con el resto de observaciones de su *cluster* frente a las de los otros *clusters*. Su valor puede estar entre -1 y 1, siendo valores altos un indicativo de que la observación se ha asignado al *cluster* correcto.

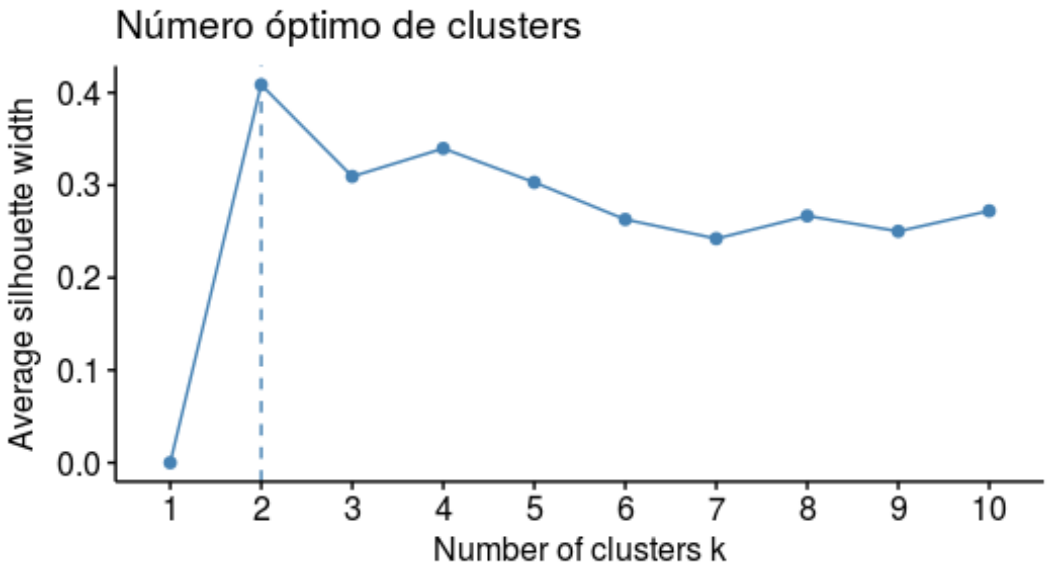
Para cada observación *i*, el *silhouette coeficient* (s_i) se obtiene del siguiente modo:

- Calcular el promedio de las distancias (llámese a_i) entre la observación *i* y el resto de observaciones que pertenecen al mismo *cluster*. Cuanto menor sea a_i mejor ha sido la asignación de *i* a su *cluster*.

- Calcular la distancia promedio entre la observación i y el resto de *clusters*. Entendiendo por distancia promedio entre i y un determinado *cluster* C como la media de las distancias entre i y las observaciones del *cluster* C.
- Identificar como b_i a la menor de las distancias promedio entre i y el resto de *clusters*, es decir, la distancia al *cluster* más próximo (*neighbouring cluster*).
- Calcular el valor de *silhouette* como:

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)}$$

```
library(factoextra)
datos <- scale(USArrests)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "silhouette") +
  labs(title = "Número óptimo de clusters")
```



El método de *average silhouette* considera como número óptimo de *clusters* aquel que maximiza la media del *silhouette coeficient* de todas las observaciones, en este caso 2.

Gap statistic method

El estadístico *gap* fue publicado por *R.Tibshirani, G.Walther y T. Hastie*, autores también del magnífico libro *Introduction to Statistical Learning*. Este estadístico compara, para diferentes valores de *k*, la varianza total *intra-cluster* observada frente al valor esperado acorde a una distribución uniforme de referencia. La estimación del número óptimo de *clusters* es el valor *k* con el que se consigue maximizar el estadístico *gap*, es decir, encuentra el valor de

k con el que se consigue una estructura de *clusters* lo más alejada posible de una distribución uniforme aleatoria. Este método puede aplicarse a cualquier tipo de *clustering*.

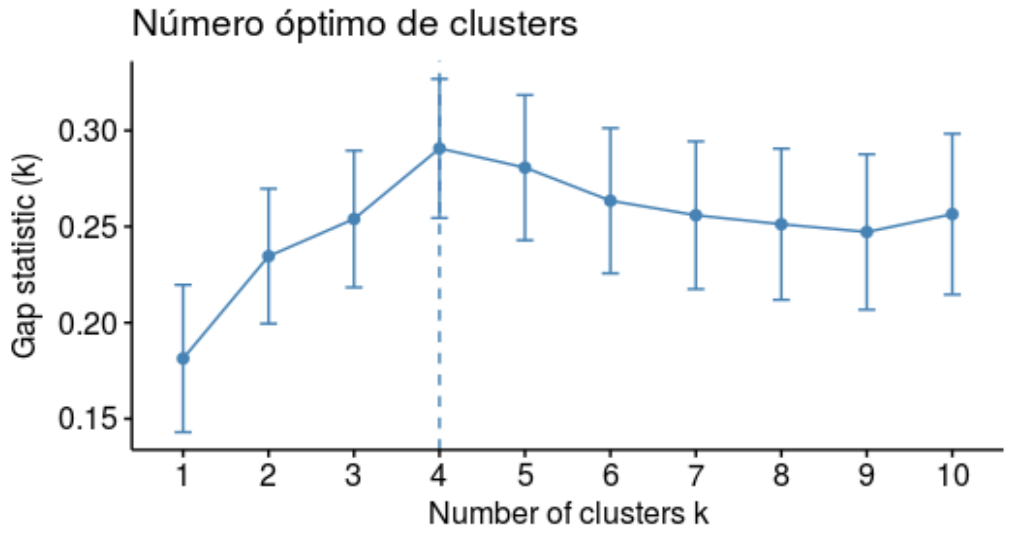
El algoritmo del *gap statistic method* es el siguiente:

- Hacer *clustering* de los datos para un rango de valores de k ($k = 1,..., K = n$) y calcular para cada uno el valor de la varianza total *intra-cluster* (W_k).
- Simular B sets de datos de referencia, todos ellos con una distribución aleatoria uniforme. Aplicar *clustering* a cada uno de los sets con el mismo rango de valores k empleado en los datos originales, calculando en cada caso la varianza total *intra-cluster* (W_{kb}). Se recomienda emplear valores de $B = 500$.
- Calcular el estadístico *gap* para cada valor de k como la desviación de la varianza observada W_k respecto del valor esperado acorde a la distribución de referencia (W_{kb}). Calcular también su desviación estándar.

$$gap(k) = \frac{1}{B}\sum_{b=1}^{B} log(W_{kb}) - log(W_k)$$

- Identificar el número de *clusters* óptimo como el menor de los valores k para el que el estadístico *gap* se aleja menos de una desviación estándar del valor *gap* del siguiente k: $gap(k) \geq gap(k+1) - s_{k+1}$.

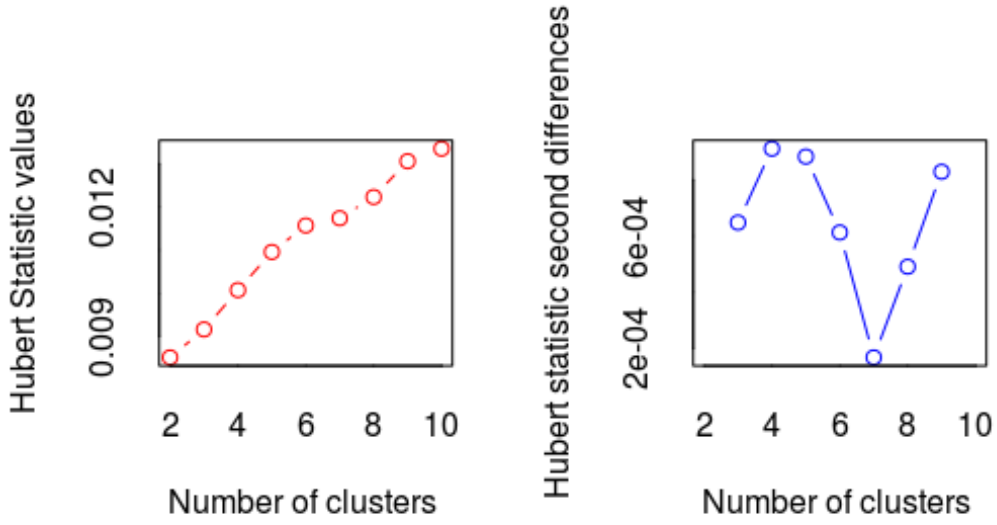
```
library(factoextra)
datos <- scale(USArrests)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "gap_stat", nboot = 500,
             verbose = FALSE, nstart = 25) +
  labs(title = "Número óptimo de clusters")
```



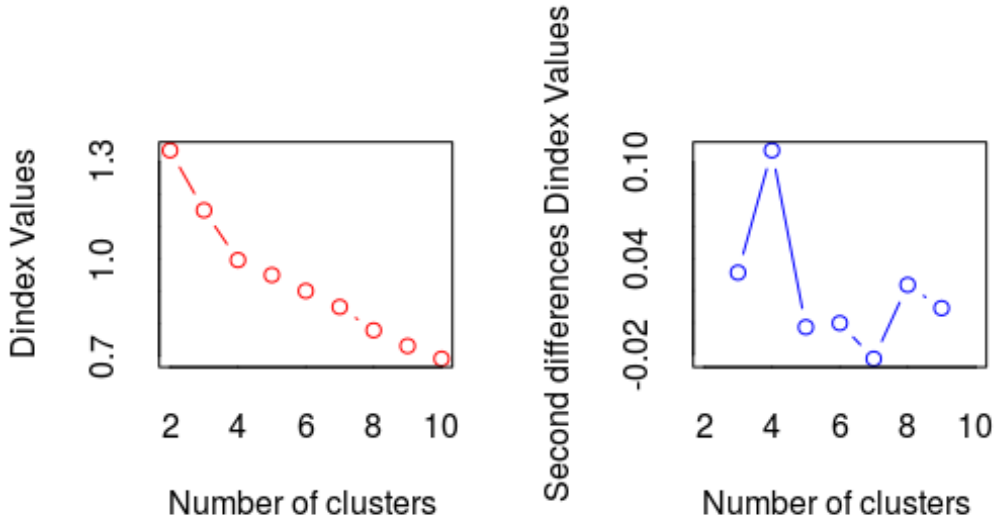
Los métodos *Elbow*, *Silhouette* y *gap* no tienen por qué coincidir exactamente en su estimación del número óptimo de *clusters*, pero tienden a acotar el rango de posibles valores. Por esta razón es recomendable calcular los tres y en función de los resultados decidir.

Además de estos tres métodos, existen en la bibliografía muchos otros desarrollados también para identificar el número óptimo de *clusters*. La función `NbClust()` del paquete `NbClust` incorpora 30 índices distintos, dando la posibilidad de calcularlos todos en un único paso. Esto último es muy útil, ya que permite identificar el valor en el que coinciden más índices, aportando seguridad de que se está haciendo una buena elección.

```
library(factoextra)
library(NbClust)

datos <- scale(USArrests)
numero_clusters <- NbClust(data = datos, distance = "euclidean", min.nc = 2,
                           max.nc = 10, method = "kmeans", index = "alllong")
```




```
## The Hubert index is a graphical method of determining the number of clusters.
## In the plot of Hubert index, we seek a significant knee that corresponds to a
## significant increase of the value of the measure i.e the significant peak in
## Hubert index second differences plot.
```

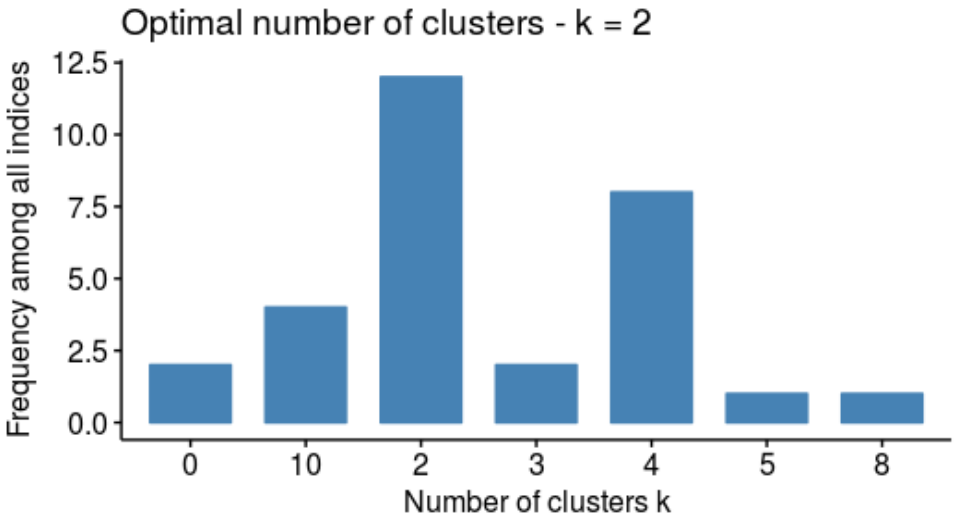



```
## The D index is a graphical method of determining the number of clusters.
## In the plot of D index, we seek a significant knee (the significant peak in
## Dindex second differences plot) that corresponds to a significant increase of
## the value of the measure.
##
## *******************************************************************
## * Among all indices:
## * 12 proposed 2 as the best number of clusters
## * 2 proposed 3 as the best number of clusters
## * 8 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 4 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

```
fviz_nbclust(numero_clusters)
```

```
## Among all indices:
## ==================
## * 2 proposed  0 as the best number of clusters
## * 12 proposed  2 as the best number of clusters
## * 2 proposed  3 as the best number of clusters
## * 8 proposed  4 as the best number of clusters
## * 1 proposed  5 as the best number of clusters
## * 1 proposed  8 as the best number of clusters
## * 4 proposed  10 as the best number of clusters
##
```

```
## Conclusion
## ========================
## * According to the majority rule, the best number of clusters is  2 .
```



Calidad de los clusters

Una vez seleccionado el número adecuado de *clusters* y aplicado el algoritmo de *clustering* pertinente se tiene que evaluar la calidad de los de los mismos, de lo contrario, podrían derivarse conclusiones de agrupación que no se corresponden con la realidad. Pueden diferenciarse tres tipos de estadísticos empleados con este fin:

- Validación interna de los *clusters*: Emplean únicamente información interna del proceso de *clustering* para evaluar la bondad de las agrupaciones generadas. Se trata de un proceso totalmente *unsupervised* ya que no se incluye ningún tipo de información que no estuviese ya incluida en el *clustering*.

- Validación externa de los *clusters* (*ground truth*): Combinan los resultados del *clustering* (*unsupervised*) con información externa (*supervised*), como puede ser un set de validación en el que se conoce el verdadero grupo al que pertenece cada observación. Permiten evaluar hasta qué punto el *clustering* es capaz de agrupar correctamente las observaciones. Se emplea principalmente para seleccionar el algoritmo de *clustering* más adecuado, aunque su uso está limitado a escenarios en los que se dispone de un set de datos de validación.

- Significancia de los *clusters*: Calculan la probabilidad (*p-value*) de que los *clusters* generados se deban únicamente al azar.

Validación interna de los clusters: estabilidad, silhouette y Dunn

La idea principal detrás del *clustering* es agrupar las observaciones de forma que sean similares a aquellas que están dentro de un mismo *cluster* y distintas a las de otros *clusters*, es decir, que la homogeneidad (también llamada *compactness* o *cohesion*) se lo mayor posible a la vez que lo es la separación entre *clusters*. Cuantificar estas dos características es una forma de evaluar cómo de bueno es el resultado obtenido.

Definiciones de homogeneidad de *cluster*:

- Promedio de la distancia entre todos los pares de observaciones (O) que forman el *cluster* (C):

$$Homogeneidad\ (C) = \frac{\sum_{O_i, O_j \in C, O_i \neq O_j} d\ istancia(O_i, O_j)}{||C|| * (||C|| - 1)}$$

- Promedio de la distancia entre las observaciones que forman el *cluster* y su centroide ($\overline{O}$):

$$Homogeneidad\ (C) = \frac{\sum_{O_i \in C,} d\ istancia(O_i, \overline{O})}{||C||}$$

La distancia entre *clusters* puede calcularse como 1-similitud. De forma análoga a la similitud *intra-clusters*, la similitud *inter-clusters* se puede definir como:

- Promedio de la distancia entre todos los pares formados por una observación del *cluster* C_1 y otra del *cluster* C_2:

$$Similitud\ (C_1, C_2) = \frac{\sum_{O_i \in C_1, O_j \in C_2} d\ istancia(O_i, O_j)}{||C_1|| * ||C_2||}$$

- Distancia mínima entre todos los pares formados por una observación del *cluster* C_1 y otra del *cluster* C_2:

$$Similitud\ (C_1, C_2) = \min_{O_i \in C_1, O_j \in C_2} (distancia(O_i, O_j))$$

- Distancia entre centroides:

$$Similitud\ (C_1, C_2) = distancia(\overline{O}_1, \overline{O}_2)$$

Dado que la homogeneidad y la separación siguen tendencias opuestas (a mayor número de *clusters* la homogeneidad aumenta, pero la separación disminuye), algunos de los índices

más frecuentemente empleados para la validación interna de *clusters* combinan ambas medidas, dos de ellos son: el *silhouette widht* y el índice *Dunn*.

Silhouette width

Cuantifica cómo de buena es la asignación que se ha hecho de una observación comparando su similitud con el resto de observaciones del mismo *cluster* frente a las de los otros *clusters*. Para cada observación i, el *silhouette coeficient* (s_i) se obtiene del siguiente modo:

- Calcular la media de las distancias (llámese a_i) entre la observación i y el resto de observaciones que pertenecen al mismo *cluster*. Cuanto menor sea a_i mayor la similitud que tiene con el resto de observaciones de su *cluster*.

- Calcular la distancia promedio entre la observación i y el resto de *clusters*. Entendiendo por distancia promedio entre i y un determinado *cluster* C como la media de las distancias entre i y las observaciones del *cluster* C.

- Identificar como b_i a la menor de las distancias promedio entre i y el resto de *clusters*, es decir, la distancia al *cluster* más próximo (*neighbouring cluster*).

- Calcular el valor de *silhouette* como:

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)}$$

Su valor puede estar entre -1 y 1, siendo valores altos un indicativo de que la observación se ha asignado al *cluster* correcto. Cuando su valor es próximo a cero significa que la observación se encuentra en un punto intermedio entre dos *clusters*. Valores negativos apuntan a una posible asignación incorrecta de la observación. Se trata por lo tanto de un método que permite evaluar el resultado del *clustering* a múltiples niveles:

- La calidad de asignación de cada observación por separado. Permitiendo identificar potenciales asignaciones erróneas (valores negativos de *silhouette*).

- La calidad de cada *cluster* a partir del promedio de los índices *silhouette* de todas las observaciones que lo forman. Si por ejemplo se han introducido demasiados *clusters*, es muy probable que algunos de ellos tengan un valor promedio mucho menor que el resto.

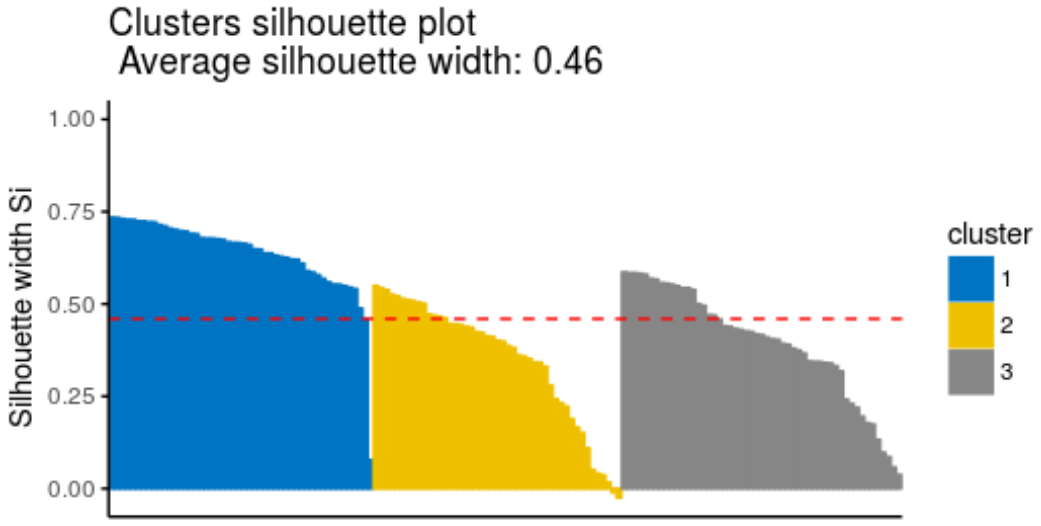
- La calidad de la estructura de *clusters* en su conjunto a partir del promedio de todos los índices *silhouette*.

El uso combinado de las funciones `eclust()` y `fviz_silhouette()` del paquete `factoextra()` permiten obtener los coeficientes *silhouette* de forma sencilla. La función

`eclust()`, gracias a su argumento `FUNcluster`, facilita el uso de múltiples algoritmos de *clustering* mediante una misma función (internamente llama a las funciones `kmeans`, `hclust`, `pam`, `clara`...).

```
library(factoextra)
# Se emplean los datos iris excluyendo la variable Species
datos <- scale(iris[, -5])
km_clusters <- eclust(x = datos, FUNcluster = "kmeans", k = 3, seed = 123,
                      hc_metric = "euclidean", nstart = 50, graph = FALSE)
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                ggtheme = theme_classic())
```

```
##   cluster size ave.sil.width
## 1       1   50          0.64
## 2       2   47          0.35
## 3       3   53          0.39
```



La función `eclust()` almacena, además de la información devuelta por la función de *clustering* empleada, en este caso *kmeans*, información sobre los coeficientes *silhouette* individuales y por *cluster*, el *cluster* al que se ha asignado cada observación y el *cluster* vecino más próximo (el segundo mejor candidato).

```
# Media silhouette por cluster
km_clusters$silinfo$clus.avg.widths
```

```
## [1] 0.6363162 0.3473922 0.3933772
```

```
# Coeficiente silhouette para cada observación
head(km_clusters$silinfo$widths)
```

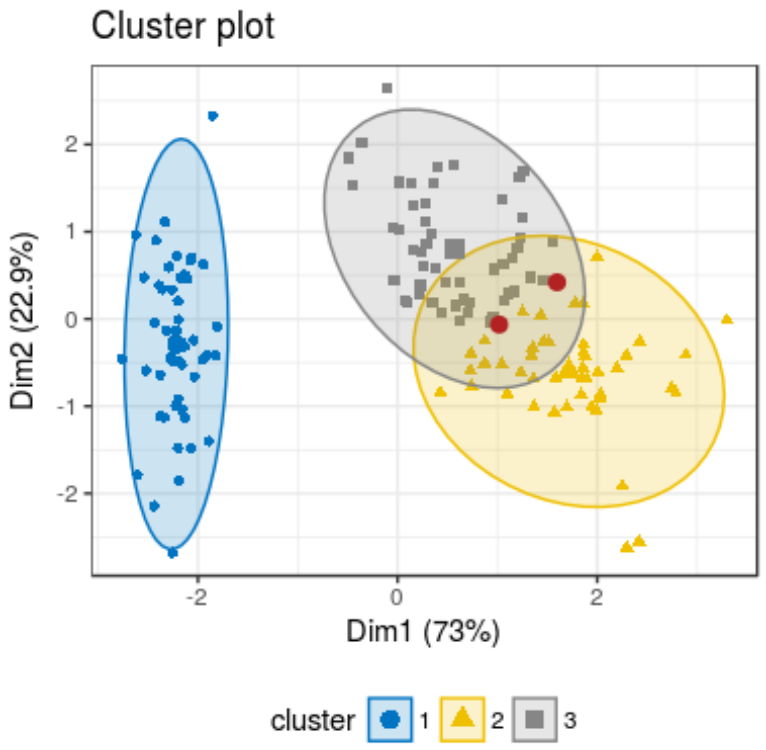
```
##    cluster neighbor sil_width
## 1        1        3 0.7341949
## 41       1        3 0.7333345
## 8        1        3 0.7308169
## 18       1        3 0.7287522
## 5        1        3 0.7284741
## 40       1        3 0.7247047
```

El *cluster* número 2 (amarillo) tiene observaciones con valores de *silhouette* próximos a 0 e incluso negativos, lo que indica que esas observaciones podrían estar mal clasificadas. Viendo la representación gráfica del *clustering*, cabe esperar que sean observaciones que están situadas en la frontera entre los *clusters* 2 y 3 ya que solapan.

```
library(dplyr)
km_clusters$silinfo$widths %>% filter(sil_width <= 0)
```

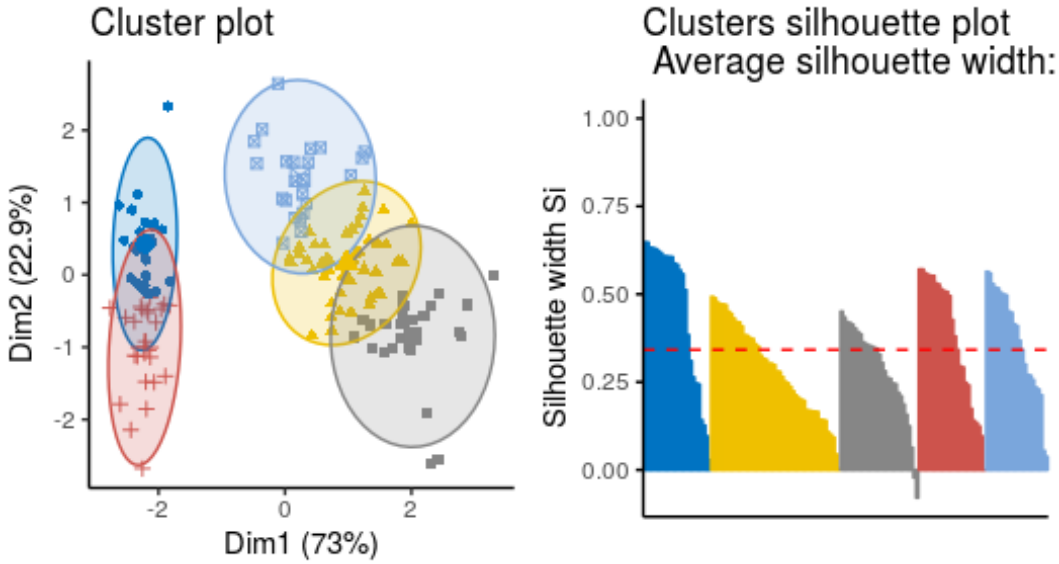
```
##   cluster neighbor   sil_width
## 1       2        3 -0.01058434
## 2       2        3 -0.02489394
```

```
p <- fviz_cluster(object = km_clusters, geom = "point", ellipse.type  = "norm",
                  palette = "jco")
p + geom_point(data = p$data[c(112, 128),], colour = "firebrick", size = 2.5) +
    theme_bw() + theme(legend.position = "bottom")
```



Véase cómo cambia el resultado si en lugar de 3 *clusters* (número correcto de especies), se crean 5.

```
library(ggpubr)
km_clusters <- eclust(x = datos, FUNcluster = "kmeans", k = 5, seed = 123,
                      hc_metric = "euclidean", nstart = 50, graph = FALSE)
p1 <- fviz_cluster(object = km_clusters, geom = "point", ellipse.type  = "norm",
                   palette = "jco") +
      theme_classic() + theme(legend.position = "none")
p2 <- fviz_silhouette(sil.obj = km_clusters, print.summary = FALSE,
                      palette = "jco", ggtheme = theme_classic()) +
      theme(legend.position = "none")
ggarrange(p1, p2)
```



Índice Dunn

El índice *Dunn* es otra medida de validación interna que se obtiene de la siguiente forma:

1. Para cada *cluster* calcular la distancia entre cada una de las observaciones que lo forman y las observaciones de los otros *clusters*.
2. Seleccionar como "representante" de la distancia entre *clusters* a la menor de todas las distancias calculadas en el paso anterior (separación mínima *inter-clusters*).
3. Para cada *cluster* calcular la distancia entre las observaciones que lo forman (*intra-cluster distance*).
4. Seleccionar como "representante" de la distancia *intra-cluster* a la mayor de todas las distancias calculadas en el paso anterior (separación máxima *intra-cluster*).

5. Calcular el índice *Dunn* como:

$$D = \frac{separacion\ minima\ interclusters}{separacion\ maxima\ intracluster}$$

Si la estructura contiene *clusters* compactos y bien separados, el numerador es grande y el denominador pequeño, dando lugar a valores altos de D. El objetivo por lo tanto es maximizar el índice *Dunn*. Esta forma de evaluar la calidad del *clustering* tiene un inconveniente. Si todos los *clusters* tienen un comportamiento ideal excepto uno, cuya calidad es baja, dado que el denominador emplea el máximo en lugar de la media, el índice estará totalmente influenciado por este *cluster* enmascarando al resto. Es importante tener en cuenta que se trata de un indicador de tipo "el peor de los casos".

Las funciones `cluster.stats()` del paquete `fpc` y `NbClust()` del paquete `NbClust` permiten calcular, entre muchos otros, el índice *Dunn*.

```
library(fpc)
# Se emplean los datos iris excluyendo la variable Species
datos <- scale(iris[, -5])

# K-means clustering con k = 3
set.seed(321)
km_clusters <- kmeans(x = dist(datos, method = "euclidean"), centers = 3,
                      nstart = 50)
# Cálculo de índices (se calculan un total de 34 índices y parámetros)
km_indices <- cluster.stats(d = dist(datos, method = "euclidean"),
                            clustering = km_clusters$cluster)

# Medidas de homogeneidad y separación
km_indices$average.within
```

```
## [1] 1.242226
```

```
km_indices$average.between
```

```
## [1] 3.168981
```

```
# Índice Dunn
km_indices$dunn
```

```
## [1] 0.0529933
```

Medidas de estabilidad

Las medidas de estabilidad son un tipo particular de validación interna que cuantifican el grado en que varían los resultados de un *clustering* como consecuencia de eliminar, de forma iterativa, una columna del set de datos. Todas ellas son relativamente costosas desde el punto de vista computacional ya que requieren repetir el *clustering* tantas veces como columnas tenga el set de datos. Dentro de esta familia de medidas se encuentran:

- *Average proportion of non-overlap (APN)*: mide la proporción media de observaciones que no se asignan al mismo *cluster* cuando se elimina una columna del set de datos en comparación a cuando se incluyen todas.
- *Average distance (AD)*: mide la media de las distancias promedio *intra-cluster* empleando todos los datos y eliminando una columna a la vez.
- *Average distance between means (ADM)*: mide la media de las distancias entre centroides empleando todos los datos y eliminando una columna a la vez.
- *Figure of merit (FOM)*: mide media de la varianza *intra-cluster* de la columna eliminada, empleando la estructura del *clustering* calcula con las columnas no eliminadas.

Los valores de *APN*, *ADM*, y *FOM* pueden ir desde 0 a 1, siendo valores pequeños un indicativo de alta estabilidad. En el caso de *AD* ocurre lo mismo pero sus valores pueden ir de 0 hasta infinito.

Validación externa de los clusters (ground truth)

Si se conoce la verdadera clasificación de las observaciones (*ground truth*), se puede evaluar la capacidad del proceso de *clustering* comparando las particiones predichas con las reales. Como el resultado de un *clustering* no es la predicción de clases sino de agrupaciones, no se pueden evaluar las coincidencias de la misma forma que en los métodos *supervised*, en los que se compara la clase predicha con la real. Por ejemplo, si el set de datos contiene observaciones que pertenecen a dos grupos diferentes (enfermos y sanos), idealmente, el *clustering* generará dos grupos, pero no les asignará ninguna identificación de tipo sanos/enfermos. En este tipo de escenarios lo que hay que contrastar es si las agrupaciones coinciden. Si la agrupación resultante es perfecta y se selecciona una observación cualquiera, el resto de observaciones que comparten el mismo *cluster* serán las mismas que comparten grupo real. Siguiendo esta idea, se puede comparar si la estructura predicha se asemeja a la estructura real de la siguiente forma:

- Dados unos resultados de *clustering* con n observaciones, se construye una matriz binaria C con dimensiones $n * n$. A cada posición C_{ij} se le asigna el valor 1 si las observaciones O_i y O_j se encuentran en el mismo *cluster*, y 0 de lo contrario.

- De forma idéntica, se construye una matriz binaria P, pero esta vez empleando la verdadera agrupación para determinar el valor de cada posición P_{ij}.

- Se resuelve la concordancia entre las matrices C y P generando los siguientes sumatorios:

  - n_{11} es el número de pares (O_i, O_j) para los que $C_{i,j} = 1$ y $P_{i,j} = 1$. Las observaciones i y j están en el mismo cluster predicho y en el mismo grupo real.

  - n_{10} es el número de pares (O_i, O_j) para los que $C_{i,j} = 1$ y $P_{i,j} = 0$. Mismo *cluster* pero distinto grupo real.

  - n_{01} es el número de pares (O_i, O_j) para los que $C_{i,j} = 0$ y $P_{i,j} = 1$. Distinto *cluster* pero mismo grupo real.

  - n_{00} es el número de pares (O_i, O_j) para los que $C_{i,j} = 0$ y $P_{i,j} = 0$. Distinto *cluster* y distinto grupo real.

- Utilizando los sumatorios anteriores se pueden calcular distintos índices de similitud entre las matrices C y P:

  - Rand index: $Rand = \frac{n_{11}+n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$

  - Jaccard coefficient: $JC = \frac{n_{11}}{n_{11}+n_{10}+n_{01}}$

  - Minkowski measure: $Minkowski = \sqrt{\frac{n_{10}+n_{01}}{n_{11}+n_{01}}}$

El *Rand index* y el *Jaccard coefficient* cuantifican las coincidencias entre las matrices C y P mientras que *Minkowski measure* cuantifica la proporción de no-concordancias respecto al total de concordancias en P. Cabe destacar que ni *Jaccard coefficient* ni *Minkowski measure* tienen en cuenta el término n_{00}, lo que evita que este sumatorio domine el resultado en escenarios en los que la mayoría de observaciones no pertenecen al mismo *cluster*, por ejemplo, estudios genéticos.

Significancia de los clusters

Estudiar la significancia de *clusters* consiste en calcular la probabilidad de que las agrupaciones se obtengan simplemente por azar. Se han desarrollado diferentes aproximaciones para cuantificar la significancia, algunas basadas en métodos de *resampling* y otras en la incorporación de información externa.

Significancia mediante información externa

La idea es añadir información adicional sobre las observaciones que permita evaluar la probabilidad del agrupamiento resultante. Por ejemplo, en el ámbito de la genómica, se puede asociar a cada observación (gen) su categoría funcional (el tipo de función que realiza en el organismo). Conociendo la cantidad total de genes que desempeñan cada una de las funciones se puede calcular, mediante un test exacto de Fisher, la probabilidad de que *m* genes con la misma función acaben por azar en un *cluster* de tamaño *n*. Ha este proceso se le conoce como *functional enrichment analysis*.

Significancia de hierarchical clusters por resampling

La idea detrás de esta estrategia es emplear *bootstrap-resampling* para simular pseudo-muestras con las que se repetir el *clustering* y luego evaluar la frecuencia con la que se repite cada *cluster*. El paquete `pvclust` automatiza este proceso para el caso particular de *hierarchical clustering*, calculando dos tipos de *p-value*: *AU (Approximately Unbiased) p-value* y *BP (Bootstrap Probability) value*, siendo el primero la opción recomendada por los creadores del paquete. *Clusters* con un valor de *AU* igual o por encima del 95% tienen fiabilidad muy alta. Se trata de un método que requiere muchos recursos computacionales ya que, para conseguir buena precisión, se necesitan al menos 1000 simulaciones. El paquete incluye la posibilidad de recurrir a computación paralela para reducir el tiempo de computación.

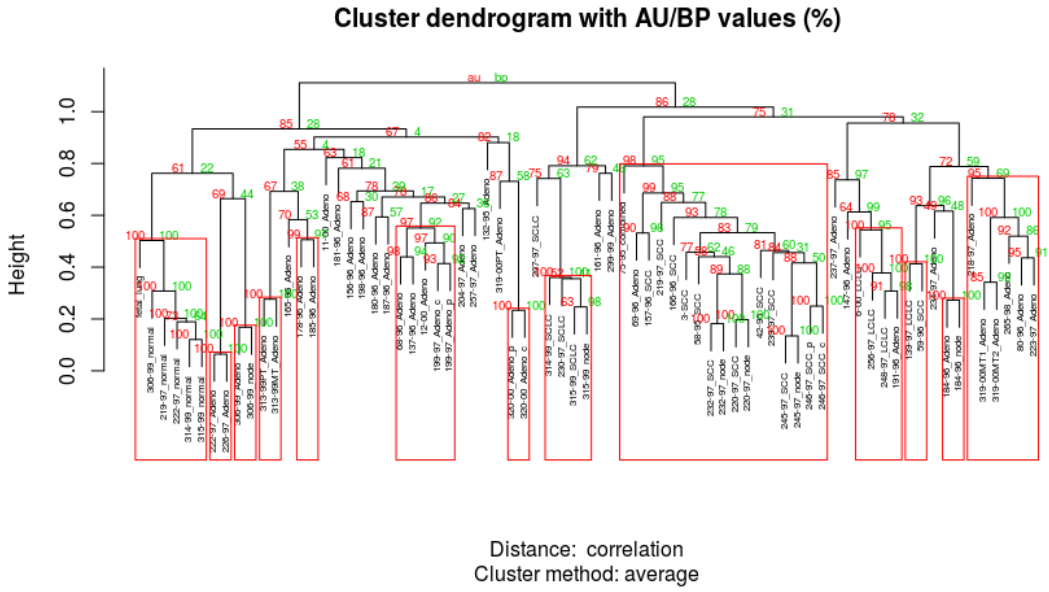
Para ilustrar el uso del paquete `pvclust` se emplea el set de datos `lung` que contiene los niveles de expresión de 916 genes en 73 muestras de pulmón, 67 de las cuales son tumores. El objetivo es aplicar hierarchical clustering y calcular la significancia de cada cluster

```
library(pvclust)
datos <- data("lung")
head(lung[, 1:5])
```

```
##               fetal_lung 232-97_SCC 232-97_node 68-96_Adeno 11-00_Adeno
## IMAGE:196992       -0.40       4.28        3.68       -1.35       -1.74
## IMAGE:587847       -2.22       5.21        4.75       -0.91       -0.33
## IMAGE:1049185      -1.35      -0.84       -2.88        3.35        3.02
## IMAGE:135221        0.68       0.56       -0.45       -0.20        1.14
## IMAGE:298560          NA       4.14        3.58       -0.40       -2.62
## IMAGE:119882       -3.23      -2.84       -2.72       -0.83       -0.02
```

Cada columna representa una muestra y cada columna un gen.

```
# Se generan solo 100 pseudo-muestras para agilizar el proceso, pero para casos
# reales los autores no recomiendan bajar de 10000
boot_hc_cluster <- pvclust(data = lung, method.dist = "cor",
                           method.hclust = "average",
                           nboot = 100, quiet = TRUE)
# Al representar un objeto pvclust se obtiene el dendrograma con los valores de
# AU-pvalue en rojo y BP-values en verde
plot(boot_hc_cluster, cex = 0.5, print.num = FALSE, cex.pv = 0.6)

# Con la función pvrect() se encuadran aquellos clusters cuyo p-value > 0.95,
# o lo que es lo mismo, su significancia está por debajo del 0.05.
pvrect(x = boot_hc_cluster, alpha = 0.95, pv = "au")
```



Aunque el *clustering* es en su naturaleza un método *unsupervised*, como se ha visto en los apartados anteriores, disponer de cierta información sobre la clasificación o agrupación real de las observaciones mejora mucho la posterior validación de los resultados.

Heatmaps

Los *heatmaps* son el resultado obtenido al representar una matriz de valores en la que, en lugar de números, se muestra un gradiente de color proporcional al valor de cada variable en cada posición. La combinación de un dendrograma con un *heatmap* permite ordenar por semejanza las filas y o columnas de la matriz, a la vez que se muestra con un código de colores el valor de las variables. Se consigue así representar más información que con un simple dendrograma y se facilita la identificación visual de posibles patrones característicos de cada *cluster*.

En `R` existen una amplia variedad de funciones desarrolladas para la creación de *heatmaps*. Algunas de ellas son:

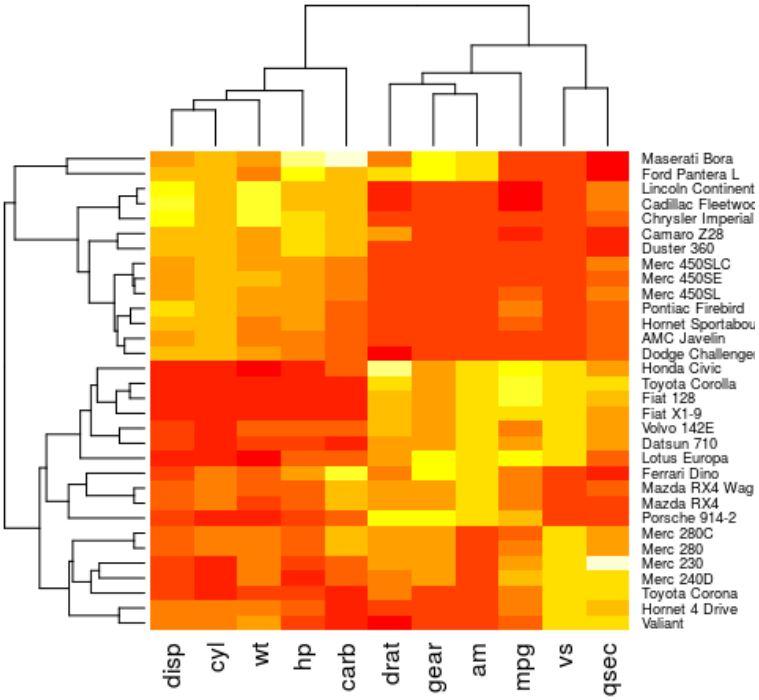
- `heatmap()[stats]`, `heatmap.2()[gplots]` y `pheatmap()[pheatmap]` para representar *heatmaps* estáticos.
- `d3heatmaps()[d3heatmaps]` para crear *heatmaps* interactivos.
- `Heatmap()[ComplexHeatmap Bioconductor]` permite un alto grado de personalización de los *heatmaps*, muy útil para datos genómicos.
- `viridis` es un paquete que contiene paletas de color muy adecuadas para generar gradientes.

A continuación, se muestran ejemplos con cada una de estas funciones:

El set de datos `mtcars` contiene información sobre 32 modelos de coche. Se pretende representar la información combinando un heatmap con un dendrograma.

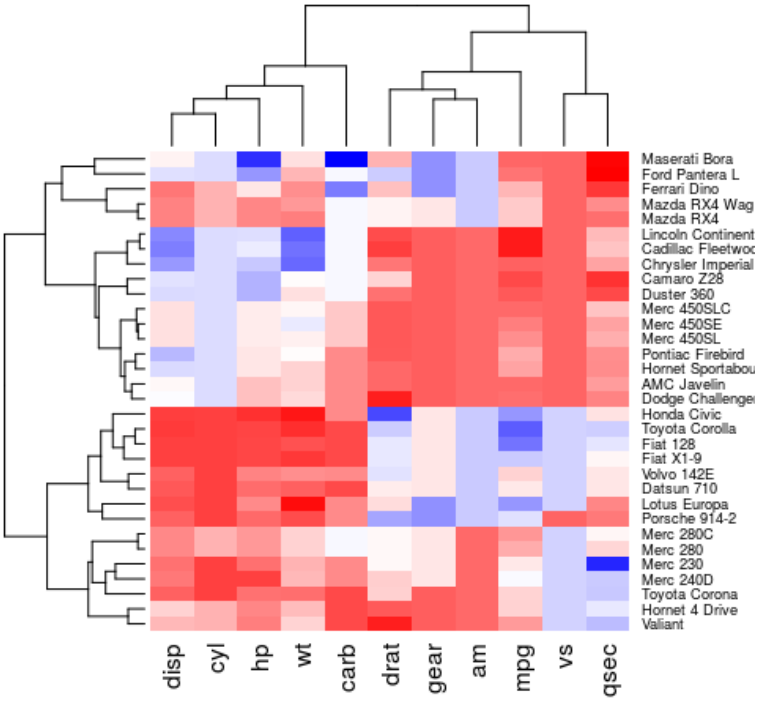
heatmap()[stats]

```
datos <- mtcars
# Para que las variables sean comparables bajo un mismo esquema de colores se
# estandarizan.
datos <- scale(datos)
heatmap(x = datos, scale = "none",
        distfun = function(x){dist(x, method = "euclidean")},
        hclustfun = function(x){hclust(x, method = "average")},
        cexRow = 0.7)
```

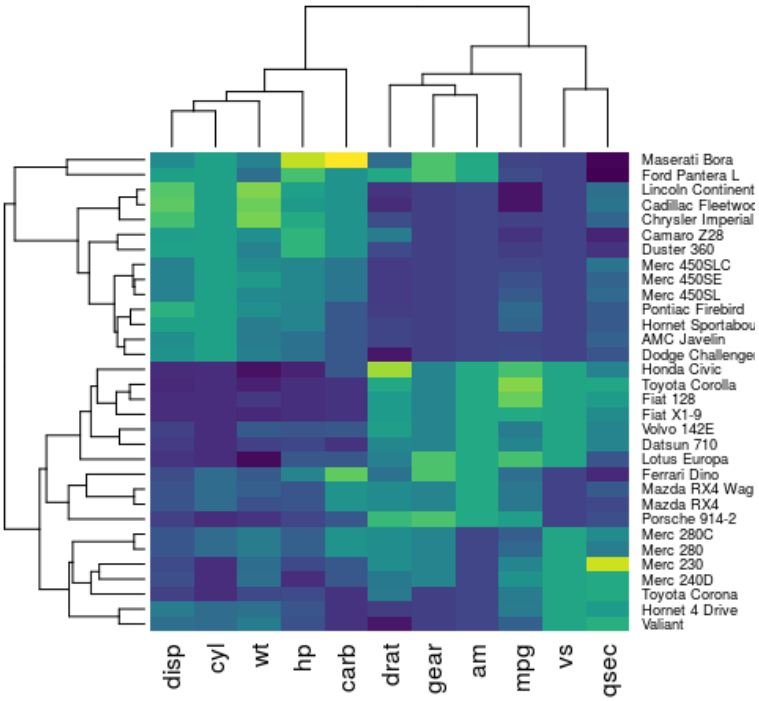


Se pueden especificar los colores mediante el argumento col.

```
colores <- colorRampPalette(c("red", "white", "blue"))(256)
heatmap(x = datos, scale = "none", col = colores, cexRow = 0.7)
```

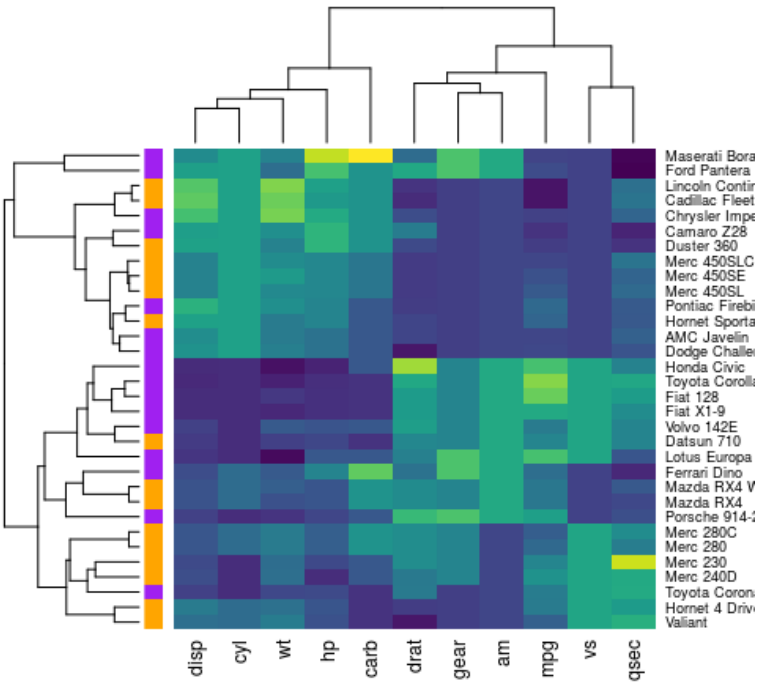


```
# Paleta de color viridis
library(viridis)
colores <- viridis(256)
heatmap(x = datos, scale = "none", col = colores,
        distfun = function(x){dist(x, method = "euclidean")},
        hclustfun = function(x){hclust(x, method = "average")},
        cexRow = 0.7)
```



Es posible añadir información adicional (*annotate*) en las filas o columnas con los argumentos `RowSideColors` y `ColSideColors`. Por ejemplo, supóngase que los primeros 16 coches proceden de China y los 16 últimos de América.

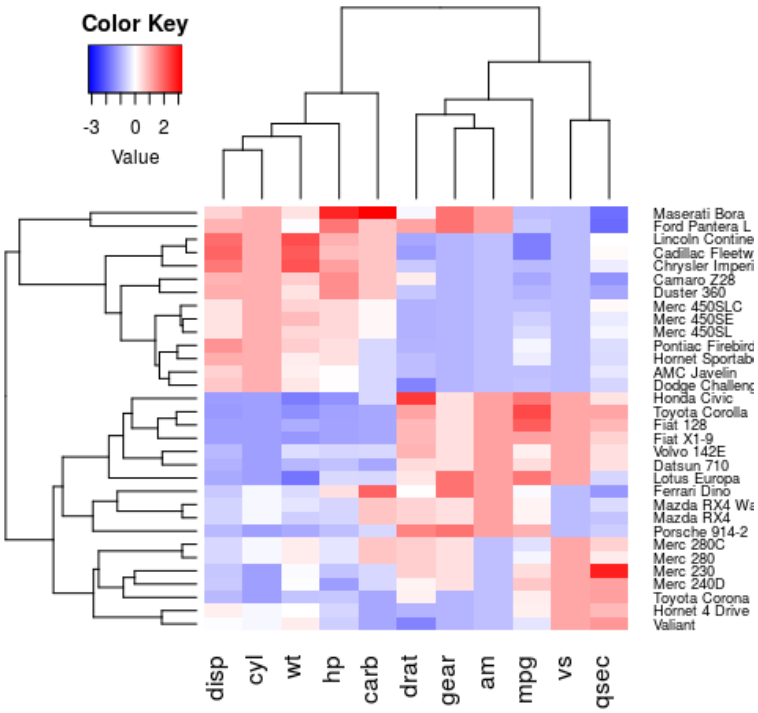
```
# Se codifica con color naranja a los coches procedentes de China y con morado a
# los de América
colores <- viridis(256)
heatmap(x = datos, scale = "none", col = colores,
        distfun = function(x){dist(x, method = "euclidean")},
        hclustfun = function(x){hclust(x, method = "average")},
        RowSideColors = rep(c("orange", "purple"), each = 16))
```



heatmap.2()[gplots]

La función `heatmap.2()` del paquete `gplots` permite expandir las capacidades básicas de la función `heatmap()`.

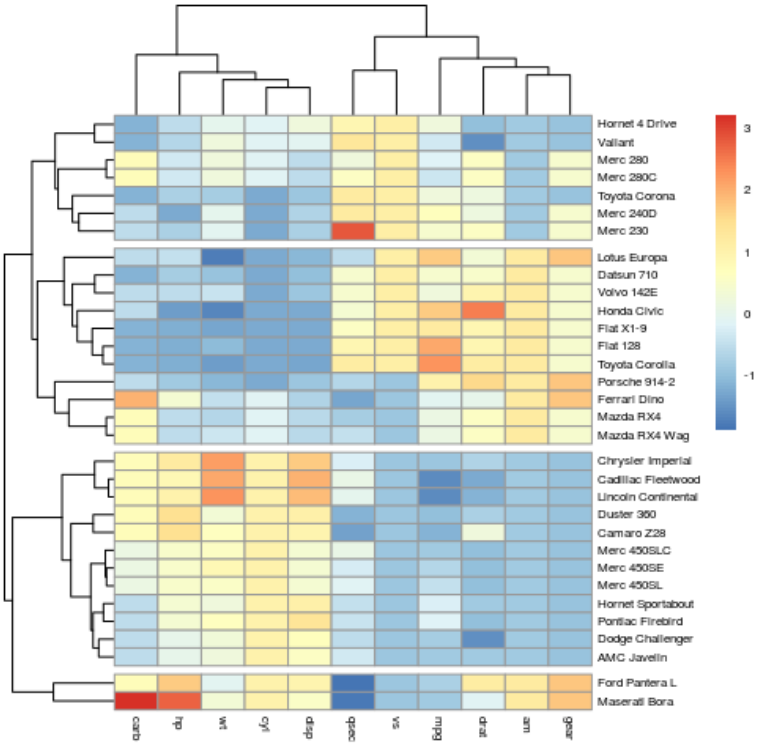
```
library(gplots)
heatmap.2(x = datos, scale = "none", col = bluered(256),
          distfun = function(x){dist(x, method = "euclidean")},
          hclustfun = function(x){hclust(x, method = "average")},
          density.info = "none",
          trace = "none", cexRow = 0.7)
```



pheatmap()[pheatmap]

Con la función `pheatmap()` del paquete `pheatmap`, se puede personalizar todavía más la representación. Por ejemplo, permite segmentar el *heatmap* por *clusters*.

```
library(pheatmap)
pheatmap(mat = datos, scale = "none", clustering_distance_rows = "euclidean",
         clustering_distance_cols = "euclidean", clustering_method = "average",
         cutree_rows = 4, fontsize = 6)
```



Heatmaps interactivos: d3heatmap()

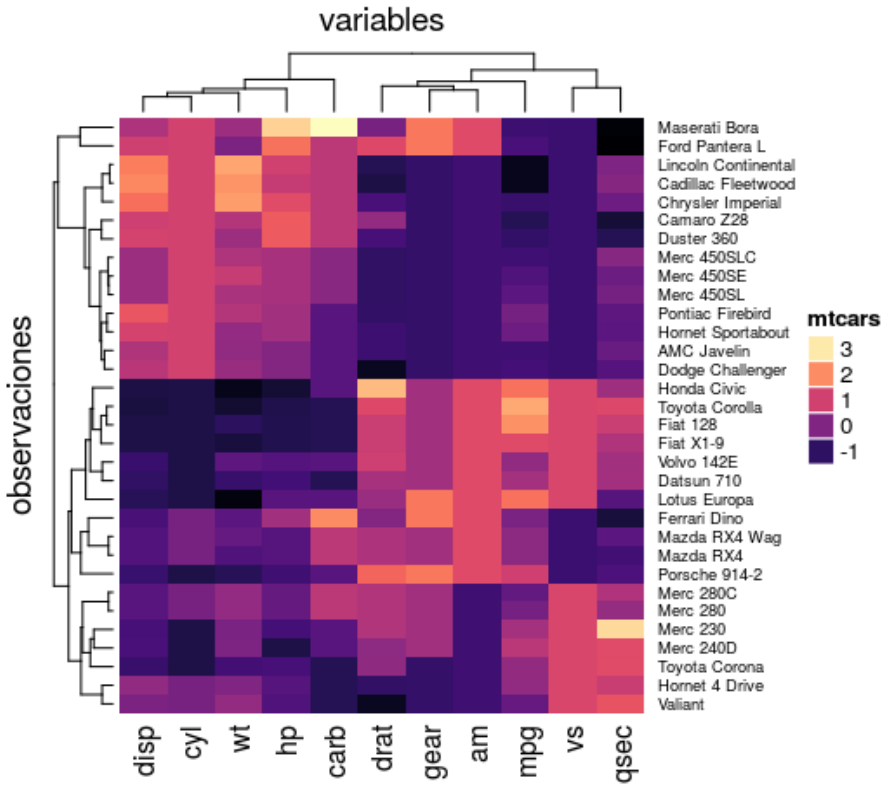
El paquete `d3heatmap` es lo que se conoce como un *htmlwidget*, contiene funciones que crean objetos *html* con los que se puede interactuar al visualizarlos en un navegador web.

```
library(d3heatmap)
# Al tratarse de html no puede visualizarse en word o PDF
d3heatmap(x = datos, k_row = 4, k_col = 2, scale = "none",
          distfun = function(x){dist(x, method = "euclidean")},
          hclustfun = function(x){hclust(x, method = "average")})
```

Complex heatmap

`ComplexHeatmap` es un paquete desarrollado por *Zuguang Gu* que permite una flexibilidad muy alta a la hora de crear *heatmaps*, anotarlos, combinarlos con otros gráficos, etc. Dadas las muchas posibilidades que ofrece el paquete, en este documento solo se muestran algunas posibilidades, para información más detallada consultar sus manuales.

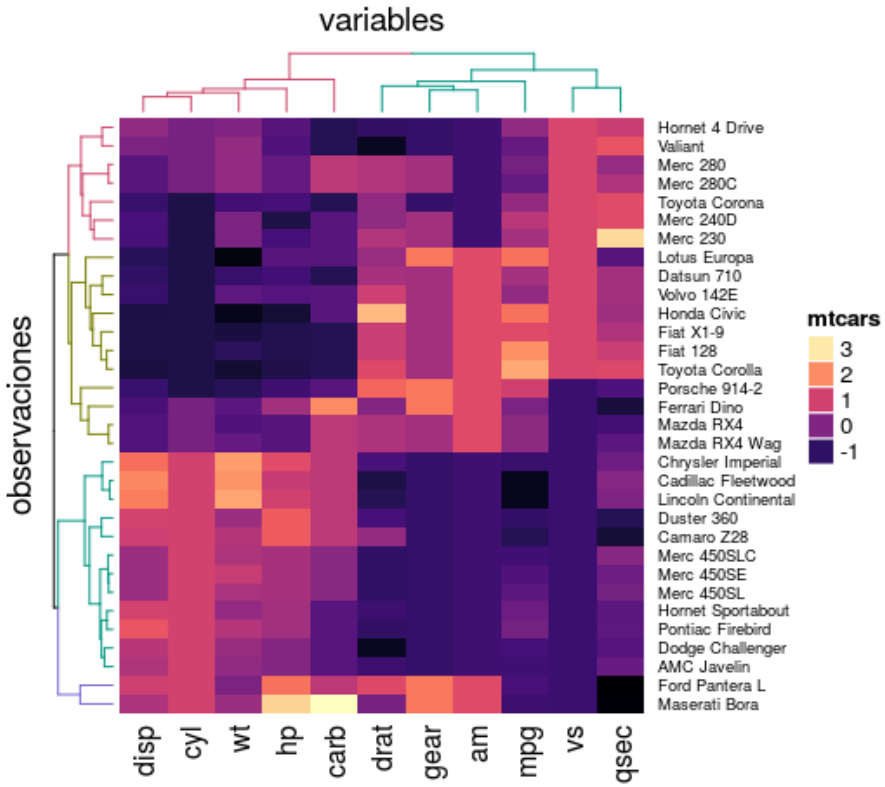
```
# El paquete se tiene que instalar desde bioconductors
# source("https://bioconductor.org/biocLite.R")
# biocLite("ComplexHeatmap")
library(ComplexHeatmap)
# Paquete viridis para la paleta de color
library(viridis)
colores <- magma(256)
Heatmap(matrix = datos, name = "mtcars",
        col = colores,
        row_title = "observaciones",
        column_title = "variables",
        row_names_gp = gpar(fontsize = 7),
        clustering_distance_columns = "euclidean",
        clustering_distance_rows = "euclidean",
        clustering_method_columns = "average",
        clustering_method_rows = "average")
```



Recurriendo a las funciones del paquete `dendextend` se puede cambiar el color de las ramas del dendrograma.

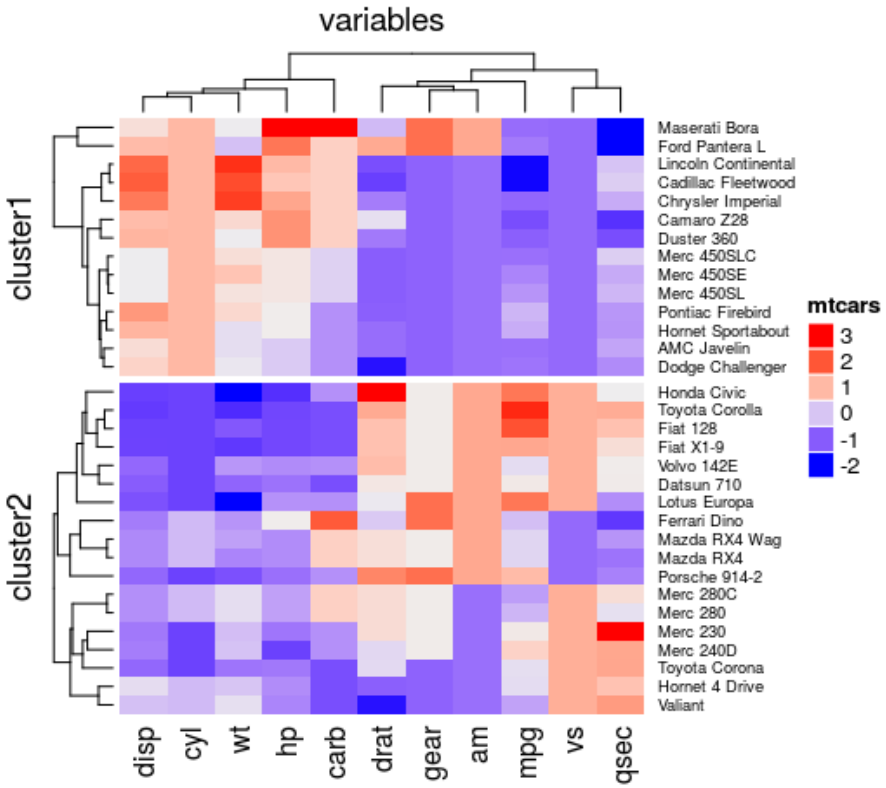
```
library(dendextend)
row_dend <- hclust(d = dist(datos, method = "euclidean"), method = "average")
# Se transpone la matriz de datos para que las observaciones estén como columnas
col_dend <- hclust(d = dist(t(datos), method = "euclidean"), method = "average")
colores <- magma(256)
```

```
Heatmap(matrix = datos, name = "mtcars",
        col = colores,
        row_title = "observaciones",
        column_title = "variables",
        row_names_gp = gpar(fontsize = 7),
        cluster_rows = color_branches(dend = row_dend, k = 4),
        cluster_columns = color_branches(dend = col_dend, k = 2))
```

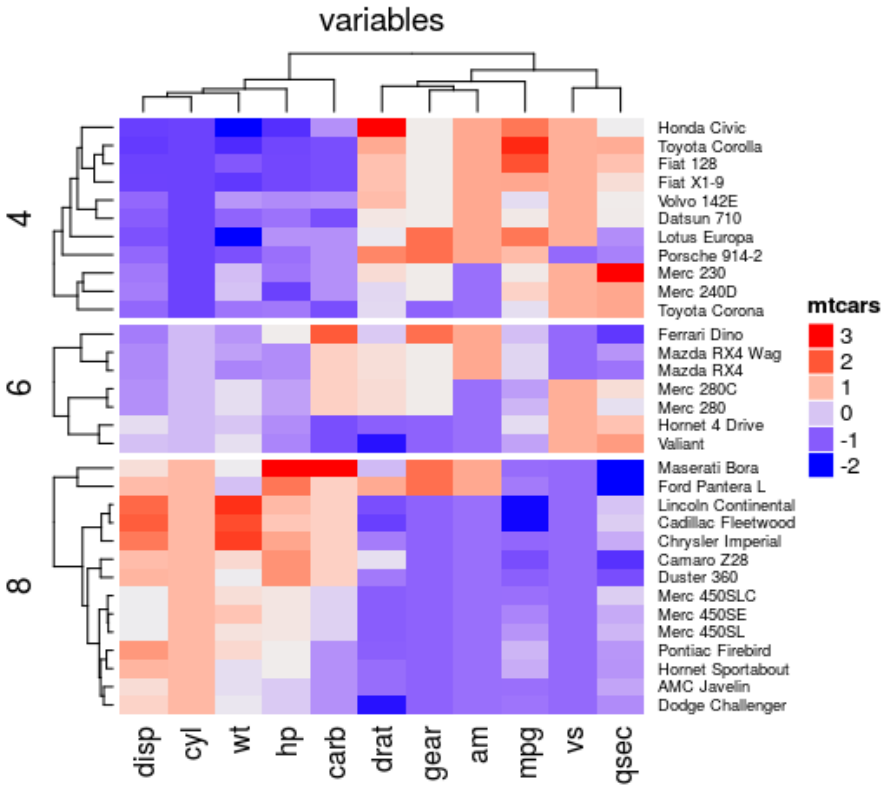


Los *heatmaps* creados mediante `Heatmap()` pueden dividirse bien en función de una variable discreta o de los *clusters* devueltos por el algoritmo *K-means*.

```
# División del heatmap acorde a un k-means con k = 2
set.seed(123)
Heatmap(matrix = datos, name = "mtcars",
        column_title = "variables",
        row_names_gp = gpar(fontsize = 7),
        clustering_distance_columns = "euclidean",
        clustering_distance_rows = "euclidean",
        clustering_method_columns = "average",
        clustering_method_rows = "average",
        km = 2)
```




```
# División del heatmap por cilindrada de los vehículos
Heatmap(matrix = datos, name = "mtcars",
        column_title = "variables",
        row_names_gp = gpar(fontsize = 7),
        clustering_distance_columns = "euclidean",
        clustering_distance_rows = "euclidean",
        clustering_method_columns = "average",
        clustering_method_rows = "average",
        split = mtcars$cyl)
```

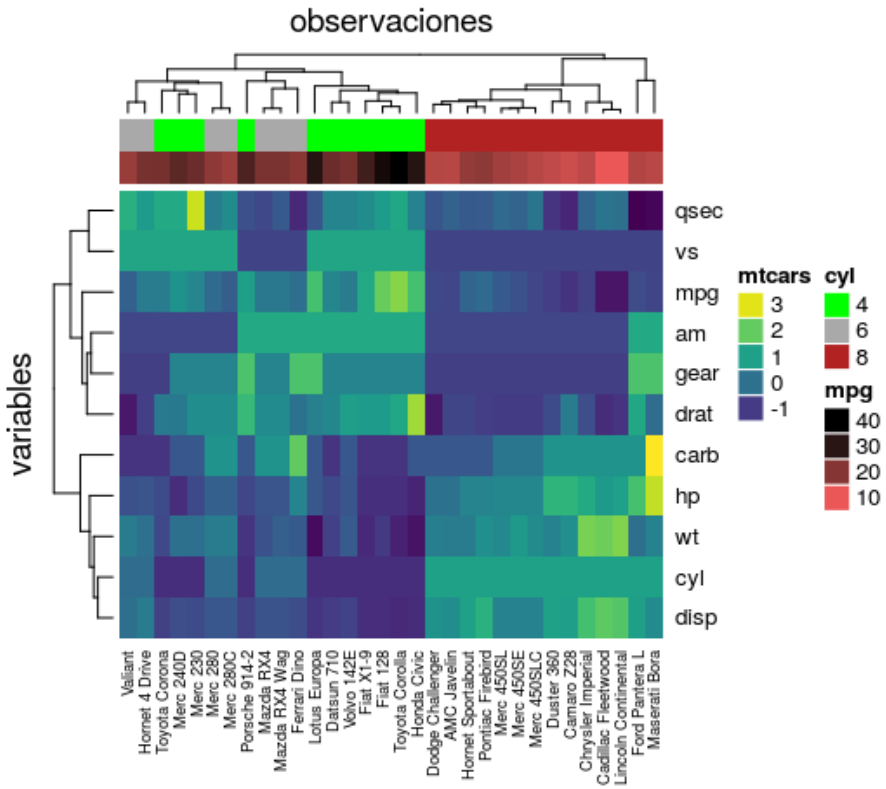


Anotar un *heatmap* consiste en añadirle información adicional a alguna de sus dimensiones (filas o columnas), bien sea en formato numérico, texto, gráfico o por código de colores. La función `HeatmapAnnotation()` del paquete `ComplexHeatmap` facilita en gran medida la anotación de *heatmaps*, controlando de forma automática muchos de los parámetros gráficos para que el resultado sea visualmente correcto.

Con frecuencia se emplea la anotación de *heatmaps* combinados con dendrogramas para identificar si las observaciones con una determinada característica se agrupan en la misma región. Por ejemplo, empleando el set de datos `mtcars` se crea un *heatmap* con dendrograma y se anota la información sobre el número de cilindros (*cyl*) y la autonomía (*mpg*).

```
# Suele ser visualmente más agradable tener las anotaciones en las columnas. Como
# la anotación se hace sobre las observaciones, se transpone el set de datos.
datos <- t(scale(mtcars))

# La función HeatmapAnnotation() recibe como argumentos (entre otros) un
# dataframe con las variables empleadas para la anotación y el código de colores
# o gradiente para cada variable anotada.

# Dataframe con las variables de interés
df_anotacion <- data.frame(cyl = mtcars$cyl, mpg = mtcars$mpg)
```

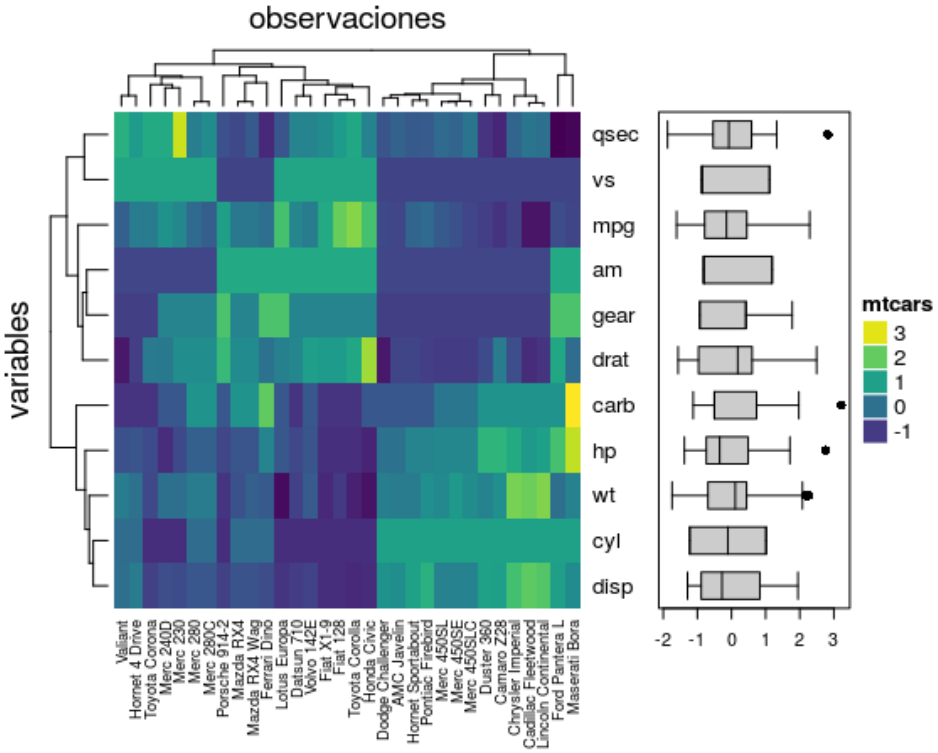
```
# Colores para cada nivel de la variable cyl y gradiente para mpg
# Para especificar en las anotaciones los colores del gradiente se tiene que
# emplear la función colorRamp2 del paquete circlize.
library(circlize)
colores <- list(cyl = c("4" = "green", "6" = "darkgrey", "8" = "firebrick"),
                mpg = circlize::colorRamp2(breaks = c(min(mtcars$mpg),
                                                      max(mtcars$mpg)),
                                           colors = c("#eb5757", "#000000")))
# Crear anotaciones
anotacion <- HeatmapAnnotation(df = df_anotacion, col = colores)

# Combinar el heatmap con las anotaciones
Heatmap(matrix = datos, name = "mtcars",
        col = viridis(256),
        column_title = "observaciones",
        row_title = "variables",
        column_names_gp = gpar(fontsize = 7),
        row_names_gp = gpar(fontsize = 10),
        clustering_distance_columns = "euclidean",
        clustering_distance_rows = "euclidean",
        clustering_method_columns = "average",
        clustering_method_rows = "average",
        top_annotation = anotacion)
```



Gracias a la anotación se puede ver de forma inmediata que el *clustering* separa correctamente los coches con 8 cilindros de los de 4 y 6.

La anotación con gráficos de distribución tales como *boxplots*, *density curves*, histogramas... se consigue de forma muy similar gracias a funciones integradas en `ComplexHeatmap`.

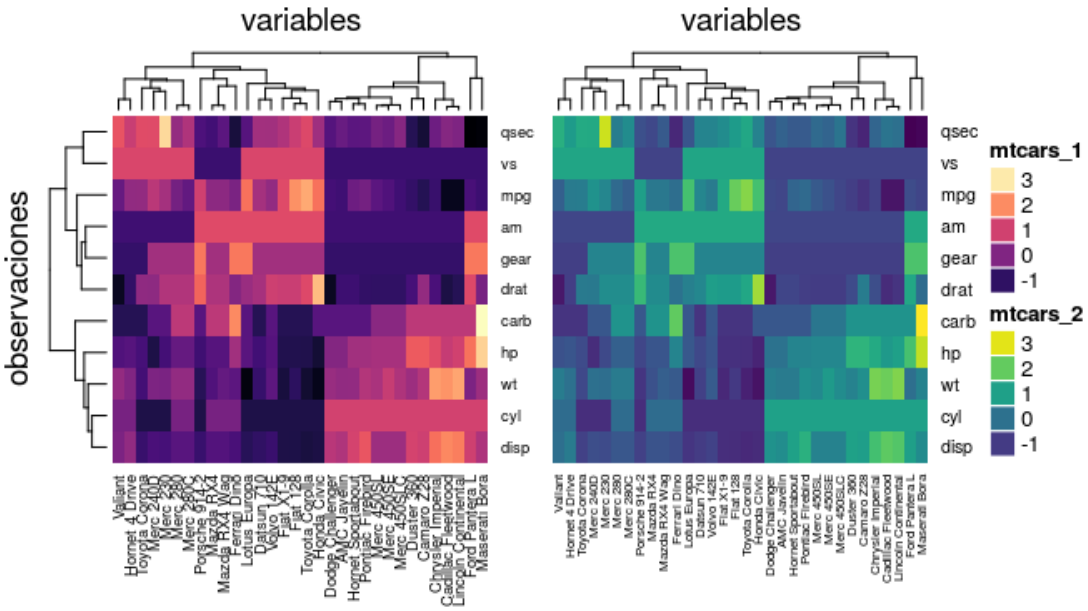
```
datos <- t(scale(mtcars))
# Anotar la distribución de las variables mediante boxplots
boxplot_anotacion <- anno_boxplot(x = datos, which = "row")

# Crear anotación
anotacion <- HeatmapAnnotation(boxplot = boxplot_anotacion, which = "row",
                               width = unit(3,"cm"))
# Combinar el heatmap con las anotaciones. Esta vez se combinan como si fuesen
# dos gráficos ggplot.
Heatmap(matrix = datos, name = "mtcars",
        col = viridis(256),
        column_title = "observaciones",
        row_title = "variables",
        column_names_gp = gpar(fontsize = 7),
        row_names_gp = gpar(fontsize = 10),
        clustering_distance_columns = "euclidean",
        clustering_distance_rows = "euclidean",
        clustering_method_columns = "average",
        clustering_method_rows = "average") +
  anotacion
```



También se pueden combinar varios *heatmaps* si se almacena cada uno en una variable y se indica que se muestren a la vez. El primer *heatmap* se emplea como referencia para ajustar el segundo.

```
colores_1 <- magma(256)
colores_2 <- viridis(256)
datos <- scale(mtcars)
datos <- t(datos)
hmap_1 <- Heatmap(matrix = datos, name = "mtcars_1",
                  col = colores_1,
                  row_title = "observaciones",
                  column_title = "variables",
                  row_names_gp = gpar(fontsize = 7),
                  column_names_gp = gpar(fontsize = 7),
                  clustering_distance_columns = "euclidean",
                  clustering_distance_rows = "euclidean",
                  clustering_method_columns = "average",
                  clustering_method_rows = "average")
hmap_2 <- Heatmap(matrix = datos, name = "mtcars_2",
                  col = colores_2,
                  row_title = "observaciones",
                  column_title = "variables",
                  row_names_gp = gpar(fontsize = 8),
                  column_names_gp = gpar(fontsize = 6),
                  clustering_distance_columns = "euclidean",
                  clustering_distance_rows = "euclidean",
```

```
                    clustering_method_columns = "average",
                    clustering_method_rows = "average")
draw(hmap_1 + hmap_2)
```



Limitaciones del clustering

El *clustering* puede ser una herramienta muy útil para encontrar agrupaciones en los datos, sobre todo a medida que el volumen de los mismos aumenta. Sin embargo, es importante recordar algunas de sus limitaciones o problemas que pueden surgir al aplicarlo.

- **Pequeñas decisiones pueden tener grandes consecuencias:** A la hora de utilizar los métodos de *clustering* se tienen que tomar decisiones que influyen en gran medida en los resultados obtenidos. No existe una única respuesta correcta, por lo que en la práctica se prueban diferentes opciones.
  - Escalado y centrado de las variables
  - Qué medida de distancia/similitud emplear
  - Número de *clusters*
  - Tipo de *linkage* empleado en *hierarchical clustering*
  - A que altura establecer el corte de un dendrograma

- **Validación de los clusters obtenidos:** No es fácil comprobar la validez de los resultados ya que en la mayoría de escenarios se desconoce la verdadera agrupación.
- **Falta de robustez:** Los métodos de *K-means-clustering* e *hierarchical clustering* asignan obligatoriamente cada observación a un grupo. Si existe en la muestra algún *outlier*, a pesar de que realmente no pertenezca a ningún grupo, el algoritmo lo asignará a uno de ellos provocando una distorsión significativa del *cluster* en cuestión. Algunas alternativas son *k-medoids* y *DBSCAN*.
- La naturaleza del algoritmo de *hierarchical clustering* conlleva que, si se realiza una mala división en los pasos iniciales, no se pueda corregir en los pasos siguientes.

Apuntes varios (miscellaneous)

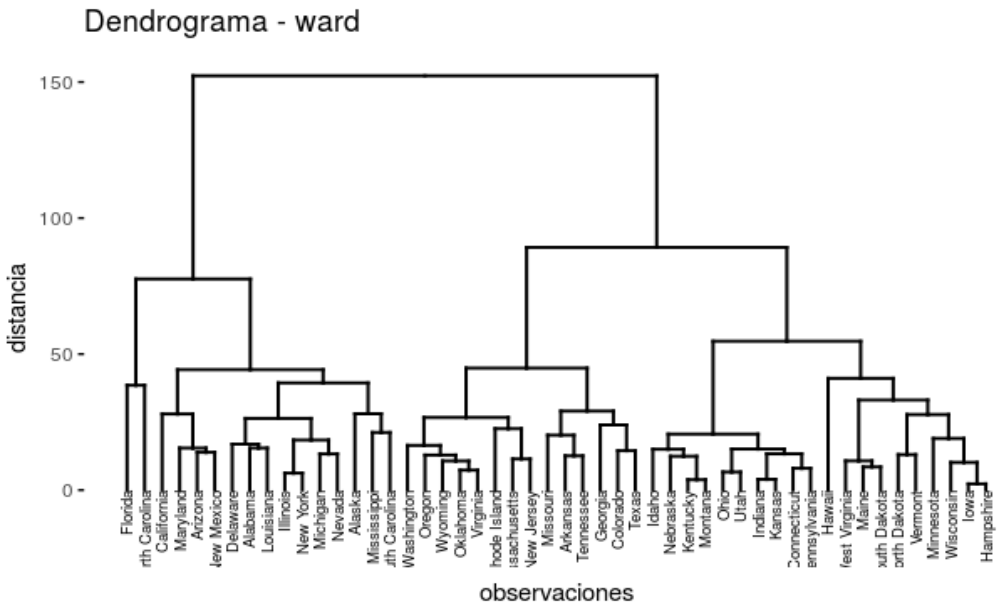
En este apartado recojo comentarios, definiciones y puntualizaciones que he ido encontrando en diferentes fuentes y que, o bien no he tenido tiempo de introducir en el cuerpo principal del documento, o que he considerado que es mejor mantenerlos al margen como información complementaria.

Customización de dendrogramas

Practical Guide to Cluster Analysis in R, Alboukadel kassambara

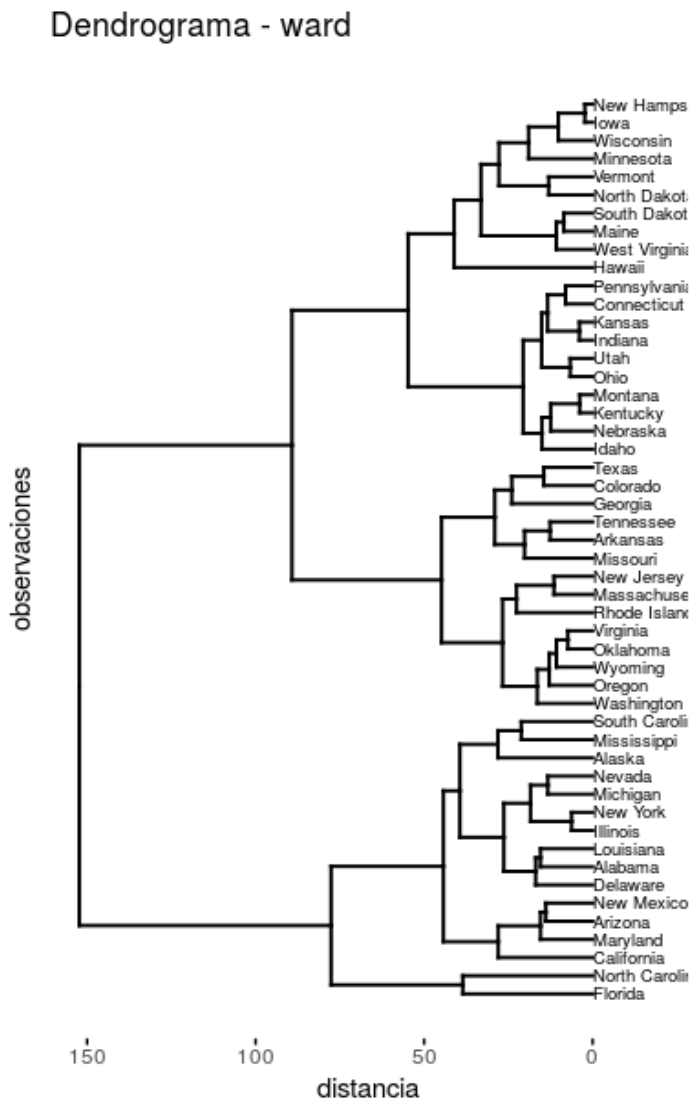
A continuación se muestra cómo visualizar y customizar dendrogramas utilizando las funciones `fviz_dend()` del paquete `factoextra` y varias funciones del paquete `dendextend`.

```
library(factoextra)
library(dendextend)
# Creación de un dendrograma con los datos de USArrests
datos <- USArrests
mat_distancia <- dist(datos, method = "euclidean")
hc_average <- hclust(d = mat_distancia, method = "average")

# Representación básica del dendrograma
set.seed(5665)
fviz_dend(x = hc_average,
          cex = 0.5,
          main = "Dendrograma - ward",
          xlab = "observaciones",
          ylab = "distancia",
          sub = "")
```



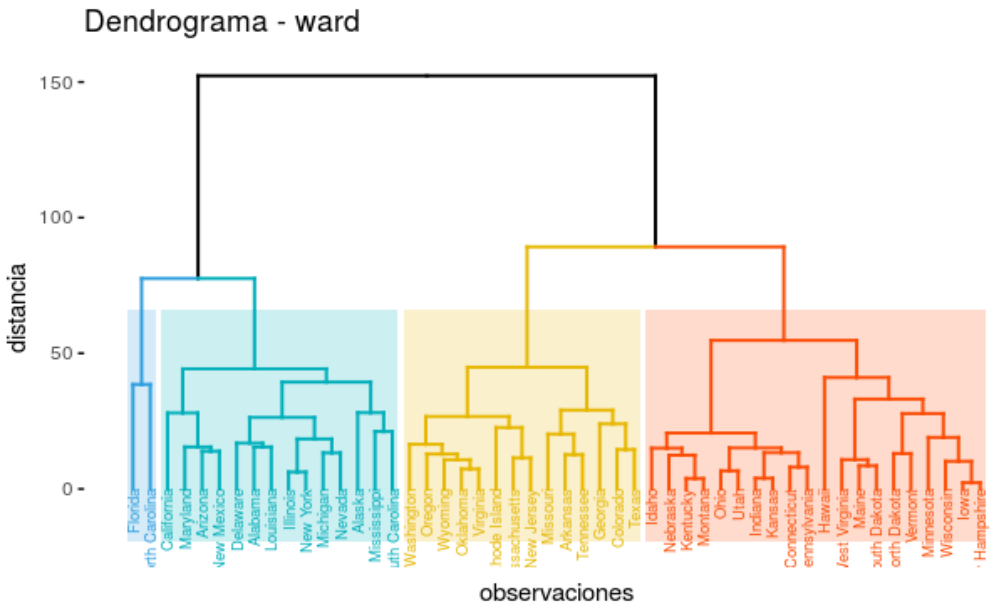
Representación horizontal

```
fviz_dend(x = hc_average,
          cex = 0.5,
          main = "Dendrograma - ward",
          xlab = "observaciones",
          ylab = "distancia",
          sub = "",
          horiz = TRUE)
```



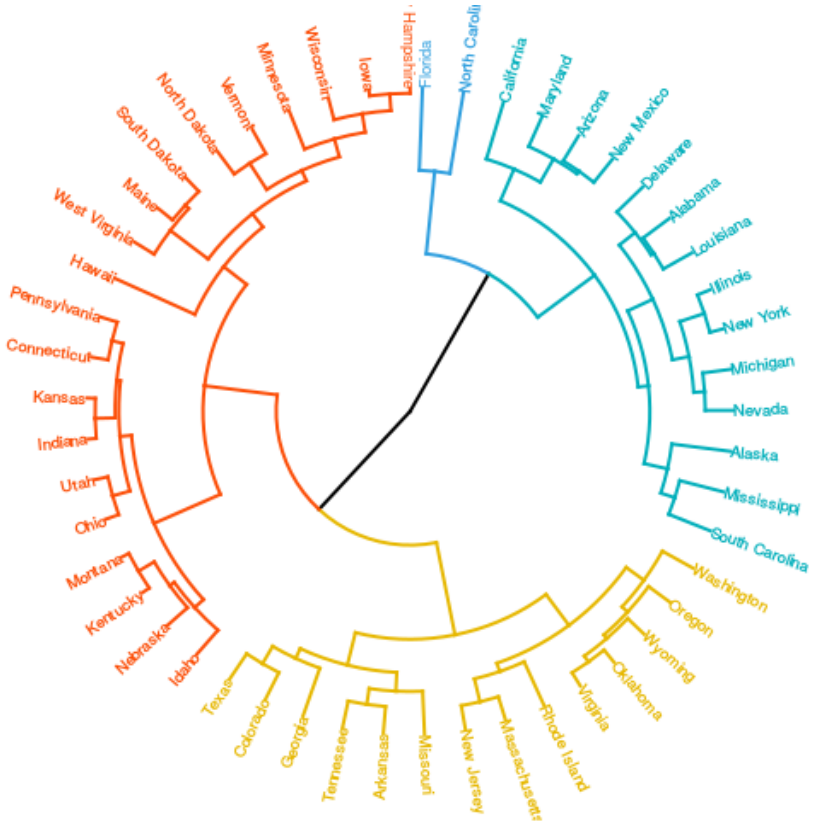
Cortar el dendrograma y asignar un color distinto a cada *cluster*.

```
set.seed(5665)
fviz_dend(x = hc_average,
          k = 4,
          k_colors = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
          color_labels_by_k = TRUE,
          rect = TRUE,
          rect_border = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
          rect_fill = TRUE,
          cex = 0.5,
          main = "Dendrograma - ward",
          xlab = "observaciones",
          ylab = "distancia",
          sub = "")
```



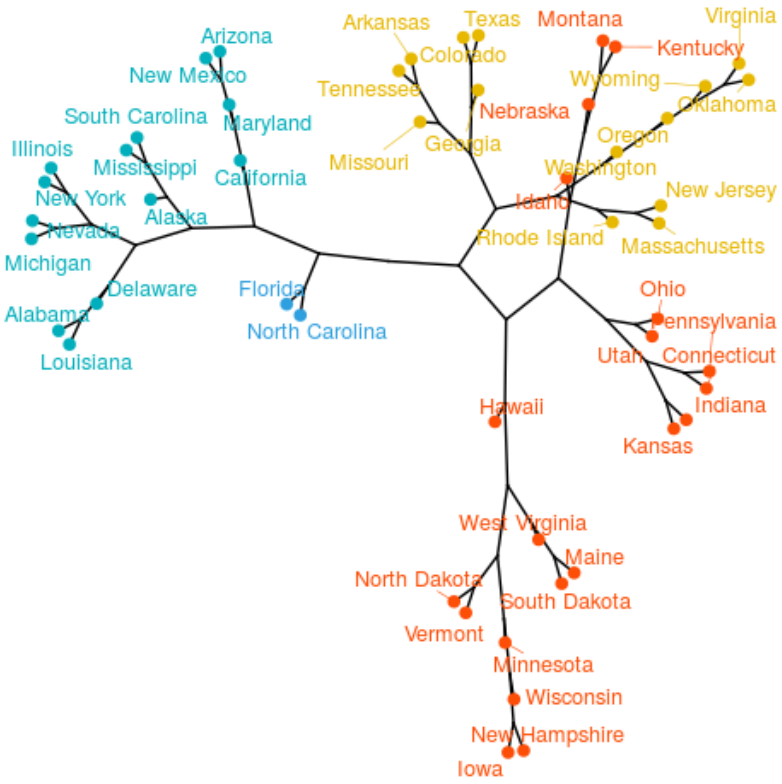
Dendrograma circular.

```
set.seed(5665)
fviz_dend(x = hc_average, k = 4,
          k_colors = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
          color_labels_by_k = TRUE, cex = 0.5, type = "circular")
```



Dendrograma en forma de árbol filogenético.

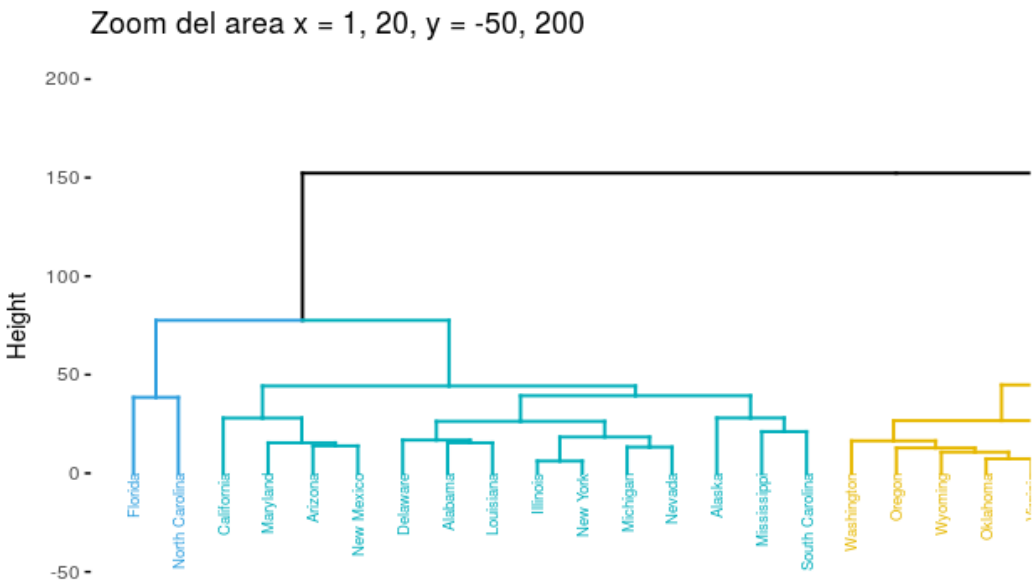
```
require("igraph")
set.seed(5665)
fviz_dend(x = hc_average,
          k = 4,
          k_colors = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
          color_labels_by_k = TRUE,
          cex = 0.8,
          type = "phylogenic",
          repel = TRUE)
```



Cuando se trabaja con dendrogramas de gran tamaño, puede resultar útil hacer *zoom* en una zona determinada, o representar únicamente algunas ramas.

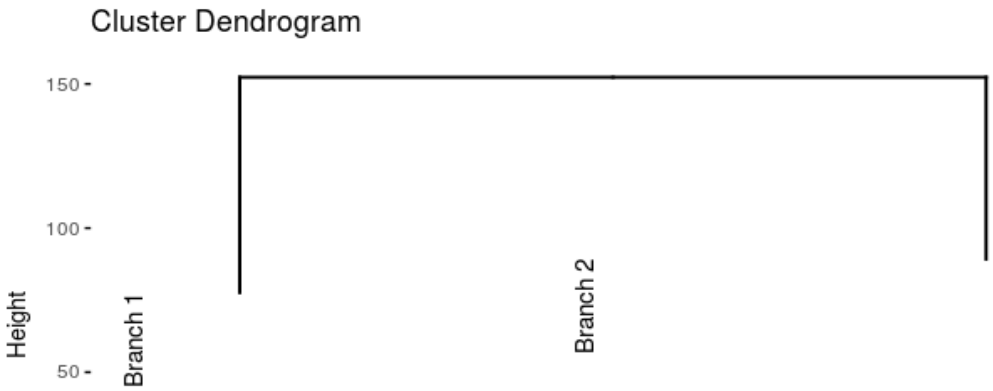
Zoom en un área determinada.

```
set.seed(5665)
fviz_dend(x = hc_average,
          k = 4,
          k_colors = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
          color_labels_by_k = TRUE,
          cex = 0.5,
          main = "Zoom del area x = 1, 20, y = -50, 200",
          xlim = c(1,20), ylim = c(-50,200))
```

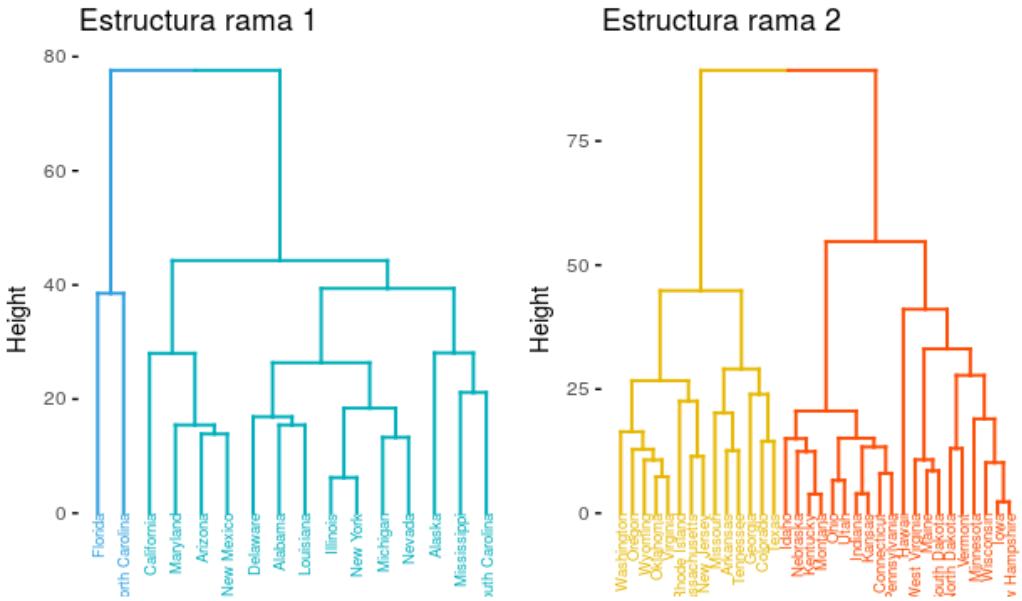


Representar determinadas ramas de un dendrograma.

```
library(ggpubr)
set.seed(5665)
# Crear la visualización del dendrograma y almacenarla en un objeto
grafico_dendrograma <- fviz_dend(x = hc_average,
                      k = 4,
                      k_colors = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
                      cex = 0.5)
# Extraer la información de la representación gráfica
datos_dendrograma <- attr(x = grafico_dendrograma, "dendrogram")

# Cortar el dendrograma a una determinada altura, por ejemplo 100
dendrograma_cortado <- cut(x = datos_dendrograma, h = 100)

# Representar cada rama que hay por encima de la altura de corte
fviz_dend(dendrograma_cortado$upper)
```



```
# Representar cada rama que hay por debajo de la altura de corte
rama_1 <- fviz_dend(dendrograma_cortado$lower[[1]], main = "Estructura rama 1")
rama_2 <- fviz_dend(dendrograma_cortado$lower[[2]], main = "Estructura rama 2")
ggarrange(rama_1, rama_2, ncol = 2)
```



Elección del mejor algoritmo de clustering

clValid, an R package for cluster validation. Guy Brock, Vasyl Pihur, Susmita Datta, and Somnath Datta

Decidir cuál es el método de *clustering* más adecuado para un determinado set de datos es un proceso complejo ya que se tienen que analizar uno a uno múltiples índices, estadísticos y parámetros (número de *clusters*, homogeneidad, separación, significancia...). El paquete clValid agiliza el proceso ofreciendo la posibilidad de comparar, de forma simultánea, múltiples algoritmos de *clustering* en una única función.

Empleando el set de datos `iris`*, se evalúan los métodos de clustering (K-means, hierarchical y PAM), empleando medidas de validación internas (conectividad, silhouette, Dunn y estabilidad), para un rango de cluster de 2 a 6.*

```
library(clValid)
datos <- scale(iris[, -5])
comparacion <- clValid(obj = datos, nClust = 2:6,
                       clMethods = c("hierarchical", "kmeans", "pam"),
                       validation = c("stability", "internal"))

summary(comparacion)
```

```
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6
##
## Validation Measures:
##                                  2       3       4       5       6
##
## hierarchical APN            0.0033  0.0370  0.0859  0.1088  0.1342
##              AD             1.4924  1.4601  1.4015  1.3269  1.2251
##              ADM            0.0161  0.1451  0.1448  0.3919  0.3176
##              FOM            0.5957  0.5809  0.5571  0.5334  0.5101
##              Connectivity   0.9762  5.5964  7.5492 18.0508 24.7306
##              Dunn           0.2674  0.1874  0.2060  0.0700  0.0762
##              Silhouette     0.5818  0.4803  0.4067  0.3746  0.3248
## kmeans       APN            0.0128  0.1034  0.1290  0.2130  0.2950
##              AD             1.5060  1.2685  1.1568  1.1269  1.0949
##              ADM            0.0555  0.2152  0.2147  0.4499  0.5147
##              FOM            0.6049  0.5206  0.4888  0.4805  0.4910
##              Connectivity   0.9762 23.8151 25.9044 40.3060 40.1385
##              Dunn           0.2674  0.0265  0.0700  0.0808  0.0808
##              Silhouette     0.5818  0.4599  0.4189  0.3455  0.3441
## pam          APN            0.0128  0.1162  0.1420  0.1655  0.1886
##              AD             1.5060  1.2721  1.1665  1.0726  1.0043
##              ADM            0.0555  0.2266  0.2500  0.2834  0.2814
##              FOM            0.6049  0.5032  0.4828  0.4789  0.4558
##              Connectivity   0.9762 23.0726 31.8067 35.7964 44.5413
##              Dunn           0.2674  0.0571  0.0566  0.0642  0.0361
##              Silhouette     0.5818  0.4566  0.4091  0.3574  0.3400
##
## Optimal Scores:
##
##              Score  Method       Clusters
## APN          0.0033 hierarchical 2
## AD           1.0043 pam          6
## ADM          0.0161 hierarchical 2
## FOM          0.4558 pam          6
## Connectivity 0.9762 hierarchical 2
## Dunn         0.2674 hierarchical 2
## Silhouette   0.5818 hierarchical 2
```

La mayoría de índices coinciden en que el mejor método es el *hierarchical clustering* con 2 *clusters*. Sin embargo, dado que se conoce la verdadera clasificación de las observaciones (*Species*), se sabe que realmente existen 3 grupos en la población.

Bibliografía

Introduction to Statistical Learning, Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani

Points of Significance: Clustering, Nature Methods, Martin Krzywinski & Naomi Altman

Practical Guide to Cluster Analysis in R, Alboukadel kassambara

Cluster Analysis for Gene Expression Data: A Survey. Daxin Jiang, Chun Tang, Aidong Zhang, Department of Computer Science and Engineering

Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? by Fionn Murtagh y Pierre Legendre

clValid, an R package for cluster validation. Guy Brock, Vasyl Pihur, Susmita Datta, and Somnath Datta Department of Bioinformatics and Biostatistics, University of Louisville

https://en.wikipedia.org/wiki/Jaccard_index

How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis. C. Fraley and A. E. Raftery

https://en.wikipedia.org/wiki/DBSCAN

This work by Joaquín Amat Rodrigo is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.