Minería de Datos para el Análisis de Big Data

Por: Carlos Carreño

ccarrenovi@Gmail.com

Abril,2021

Modulo 9 Clustering Avanzado

- Model based clustering
- Density based clustering (DBSCAN)
- Validación del Clustering
- Heatmaps
- Comparación de Dendrogramas

Model based clustering

- 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.
- 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)*.
- MBC 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.

Ejemplo: Model Based Cluster

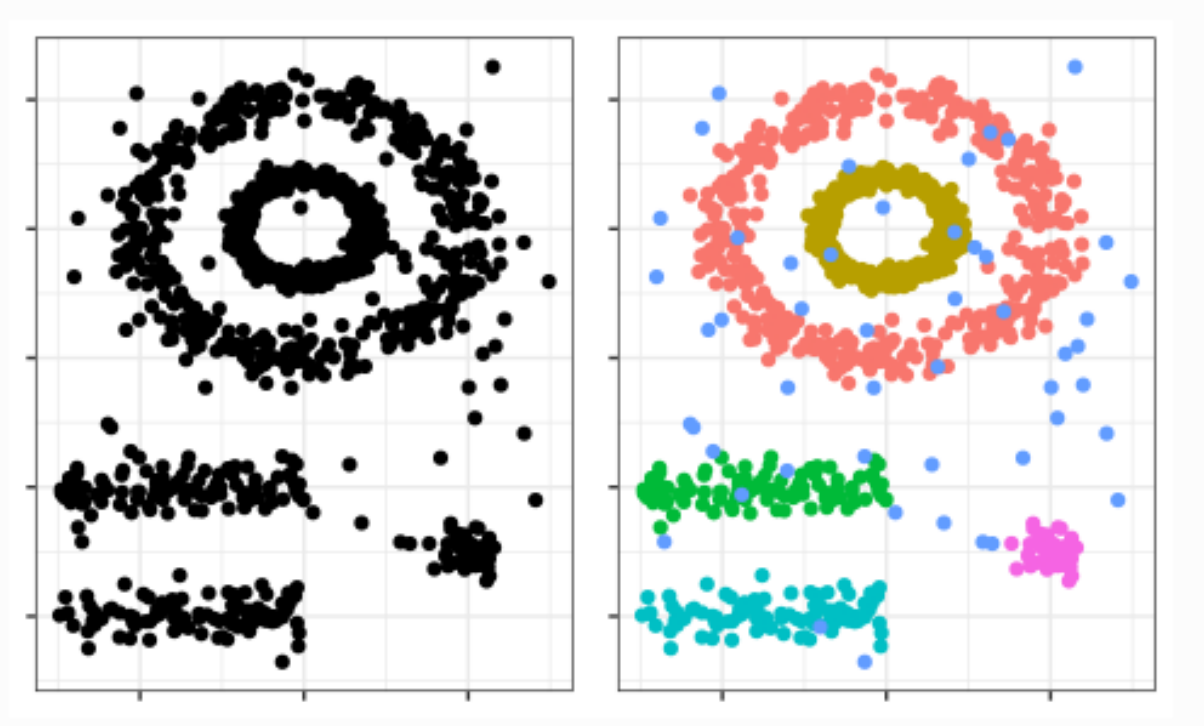
- El set de datos diabetes del paquete mclust contiene 3 parámetros sanguíneos medidos en 145 pacientes con 3 tipos distintos de diabetes. Se pretende emplear model-based-clustering para encontrar las agrupaciones.

Density based clustering (DBSCAN)

- *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.

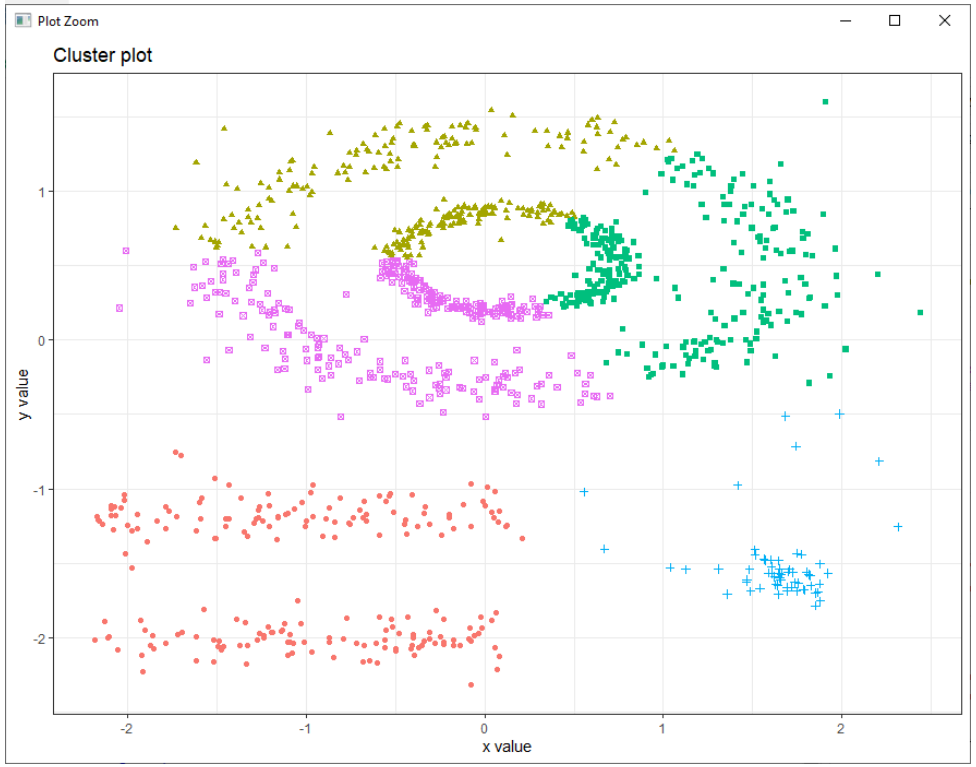
...

- El cerebro humano identifica fácilmente 5 agrupaciones y algunas observaciones aisladas (ruido).



Usando K-means

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



…

- Los *clusters* generados por k-means, 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.

DBSCAN Parámetros

- 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.

Conectividad

- 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.

Ejemplo: Conectividad entre observaciones

- 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.

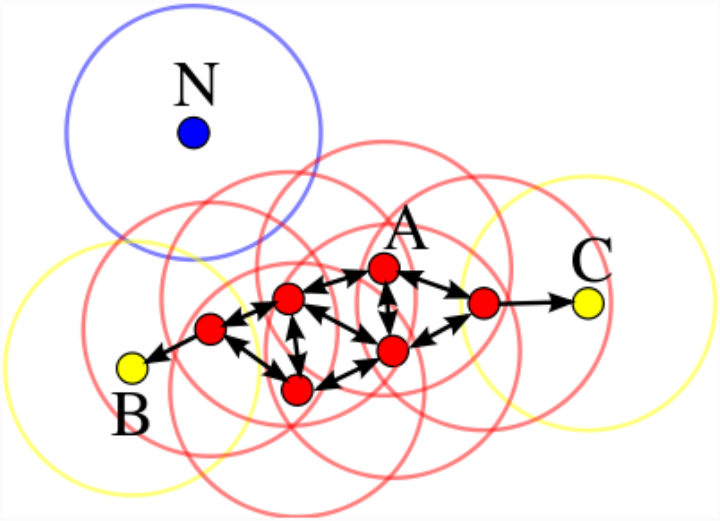


Imagen obtenida de Wikipedia

Algoritmo DBSCAN

1. Para cada observación Xi 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 Xi 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.

Nota:

- 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 y Desventajas de DBSCAN

- 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.

Validación del Clustering

- La validación de *clusters* es el proceso por el cual se evalúa la veracidad de los grupos obtenidos.
- El proceso de validación por lo general consta de tres partes:
  1. Estudio de la tendencia de *clustering*,
  2. Elección del número óptimo de *clusters* y
  3. 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*).

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*.

```
> hopkins(data = datos_iris, n = nrow(datos_iris) - 1)
$H
[1] 0.1797192

> # Estadístico H para el set de datos simulado
> hopkins(data = datos_simulados, n = nrow(datos_simulados) - 1)
$H
[1] 0.5030529
```

Nota:
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.

```r
library(clustertend)
set.seed(321)
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)
# Estadístico H para el set de datos iris
hopkins(data = datos_iris, n = nrow(datos_iris) - 1)

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

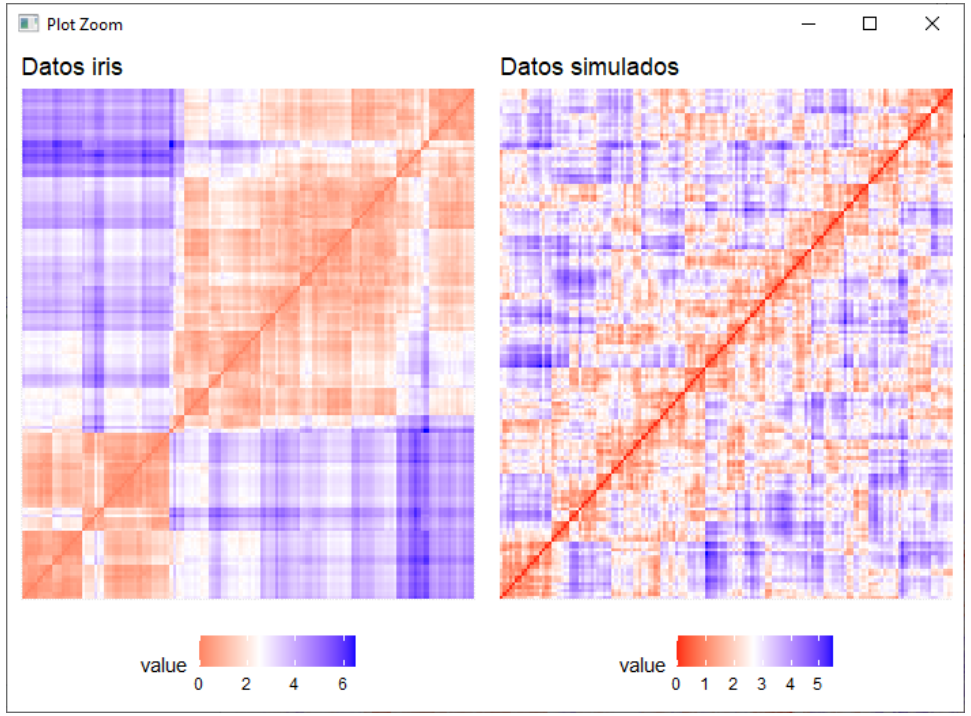
Visual Assessment of cluster Tendency (VAT)

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.

Ejemplo: VAT

```r
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.

Numero Optimo de Cluster

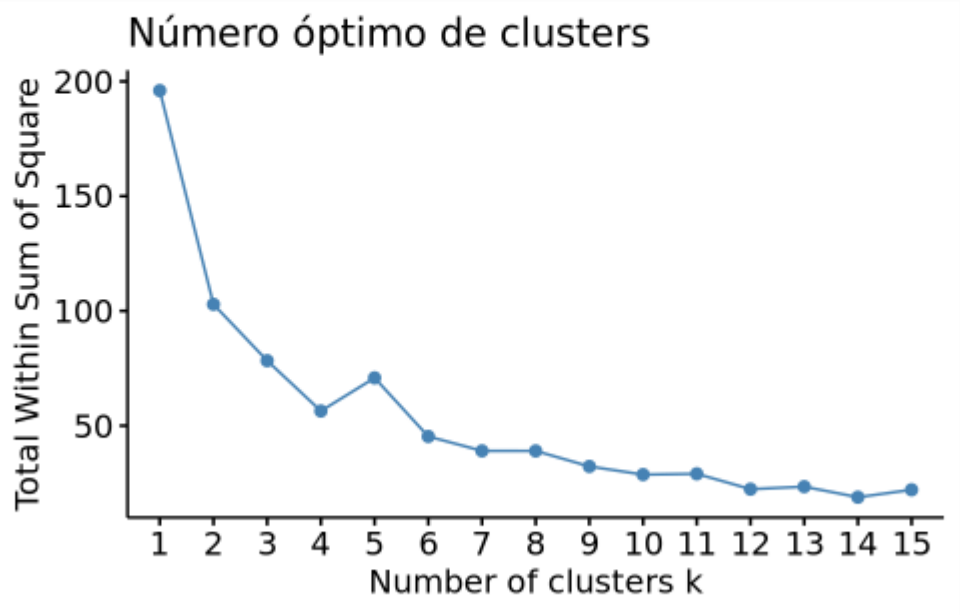
No existe una forma única de averiguar el número adecuado de *clusters*.

- Elbow method
- Average silhouette method
- Gap statistic method

Elbow Method

- El método Elbow sigue una estrategia comúnmente empleada para encontrar el valor óptimo de un hiperparámetro.
- La idea general es probar un rango de valores del hiperparámetro 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).

```r
library(factoextra)
datos <- scale(USArrests)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "wss", k.max = 15) +
  labs(title = "Número óptimo de clusters")
```



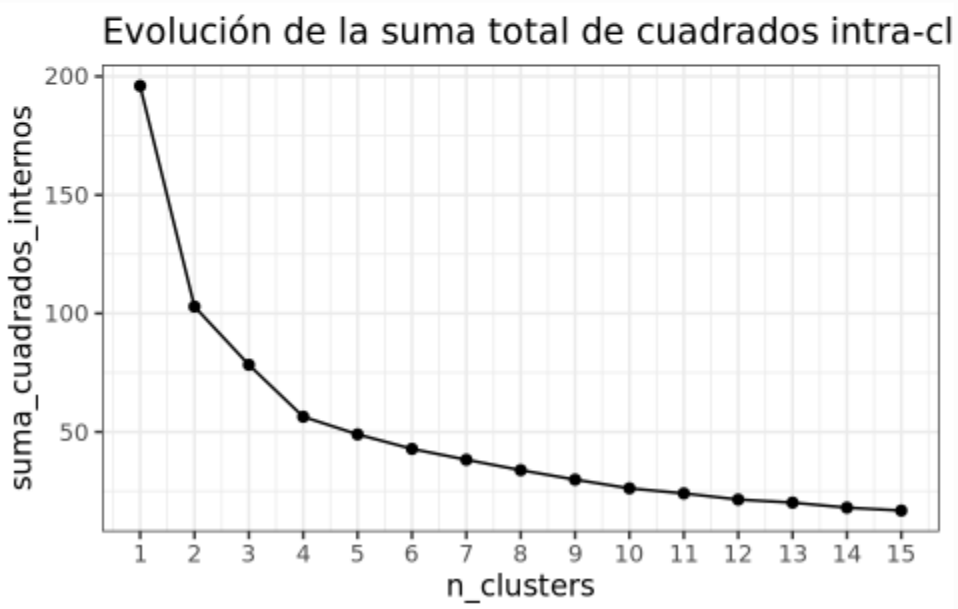
La curva indica que a partir de 4 *clusters* la mejora es mínima.

Sin fviz_nbclust()

```
calcular_totwithinss <- function(n_clusters, datos, iter.max=1000, nstart=50){
  # Esta función aplica el algoritmo kmeans y devuelve la suma total de
  # cuadrados internos.
  cluster_kmeans <- kmeans(centers = n_clusters, x = datos, iter.max = iter.max,
                           nstart = nstart)
  return(cluster_kmeans$tot.withinss)
}

# Se aplica esta función con para diferentes valores de k
total_withinss <- map_dbl(.x = 1:15,
                          .f = calcular_totwithinss,
                          datos = datos)
total_withinss
```

```
##  [1] 196.00000 102.86240  78.32327  56.40317  48.94420  42.83303  38.25764
##  [8]  33.85843  29.86789  26.18348  24.05222  21.47090  20.15762  18.04643
## [15]  16.81152
```

```r
data.frame(n_clusters = 1:15, suma_cuadrados_internos = total_withinss) %>%
  ggplot(aes(x = n_clusters, y = suma_cuadrados_internos)) +
    geom_line() +
    geom_point() +
    scale_x_continuous(breaks = 1:15) +
    labs(title = "Evolución de la suma total de cuadrados intra-cluster") +
    theme_bw()
```



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* o índices silueta (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.

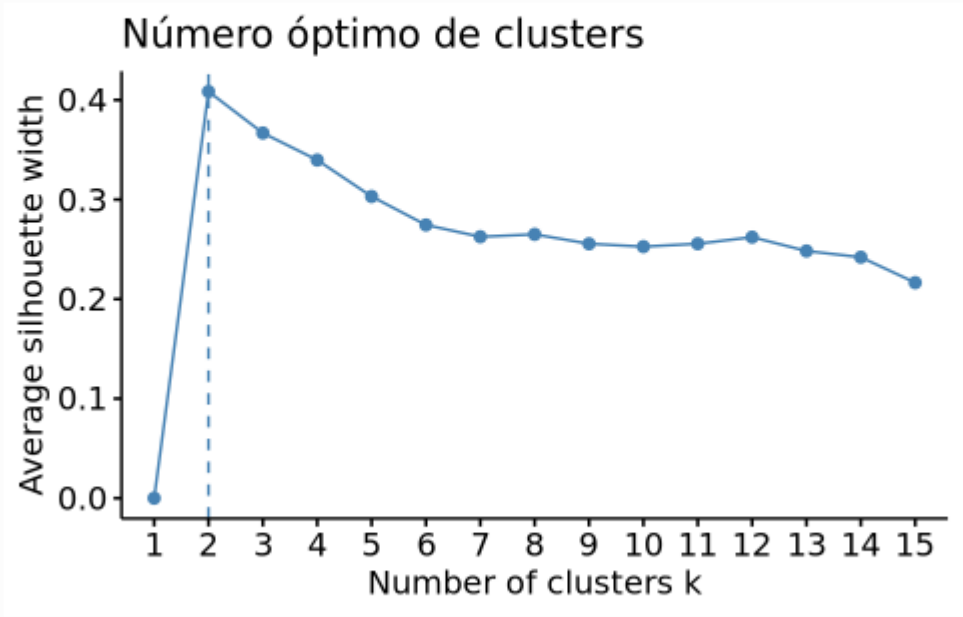
Para cada observación i, el *silhouette coeficient* (s_i) se obtiene del siguiente modo:

- 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:

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

Ejemplo: **Average silhouette method**

```
library(factoextra)
datos <- scale(USArrests)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "silhouette", k.max = 15) +
  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*.

Algoritmo *gap statistic method*

- Hacer *clustering* de los datos para un rango de valores de k ($k = 1, \ldots, K = n$) 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.

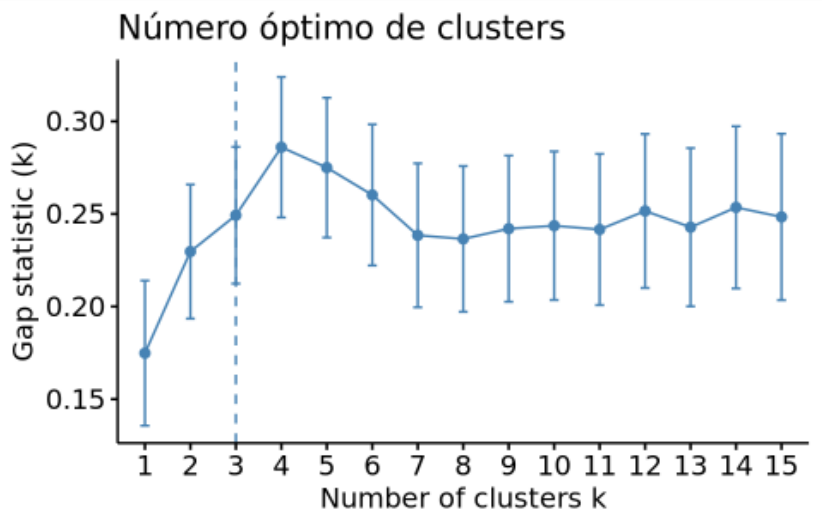
$$gap(k) = \frac{1}{B}\sum_{b=1}^{B} log(W_{kb}) - log(W_k)$$

- 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: $gap(k) \geq gap(k+1) - s_{k+1}$.

Estadístico Gap

- Se puede obtener el el estadístico gap con la función **fviz_nbclust()** o con la función **clusGap()** del paquete cluster.

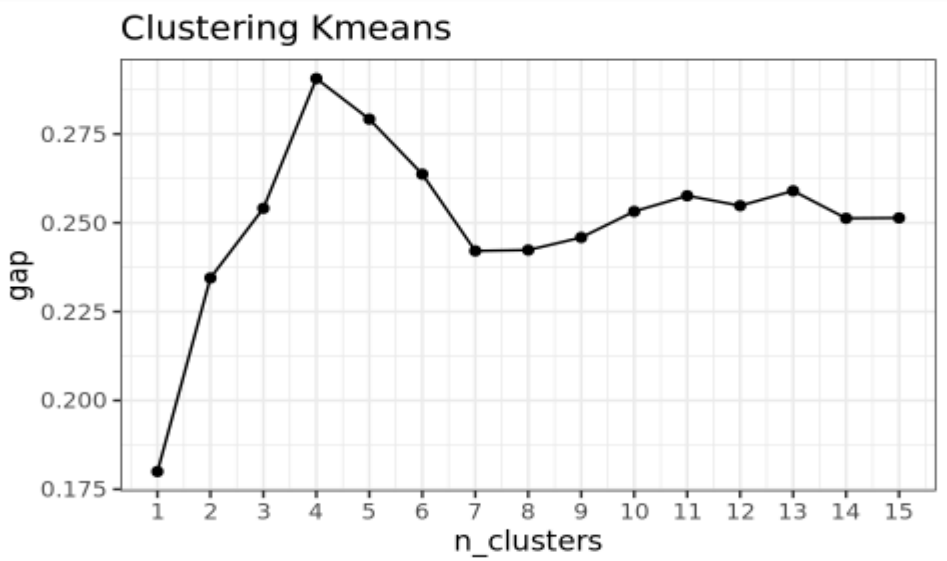
```
library(factoextra)
datos <- scale(USArrests)
set.seed(896)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "gap_stat", nboot = 500,
             k.max = 15, verbose = FALSE, nstart = 50) +
  labs(title = "Número óptimo de clusters")
```



```r
set.seed(896)
kmeans_gap <- clusGap(x = datos,
                      FUNcluster = kmeans,
                      K.max = 15,
                      B = 500,
                      verbose = FALSE,
                      nstart = 50)
kmeans_gap
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = datos, FUNcluster = kmeans, K.max = 15, B = 500,     verbose = FALSE, nstart = 50)
## B=500 simulated reference sets, k = 1..15; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstSEmax', SE.factor=1): 3
##           logW   E.logW       gap     SE.sim
##  [1,] 3.458369 3.638351 0.1799816 0.03934034
##  [2,] 3.135112 3.369586 0.2344744 0.03538104
##  [3,] 2.977727 3.231814 0.2540873 0.03622365
##  [4,] 2.826221 3.116833 0.2906128 0.03679382
##  [5,] 2.738868 3.018052 0.2791840 0.03739620
```

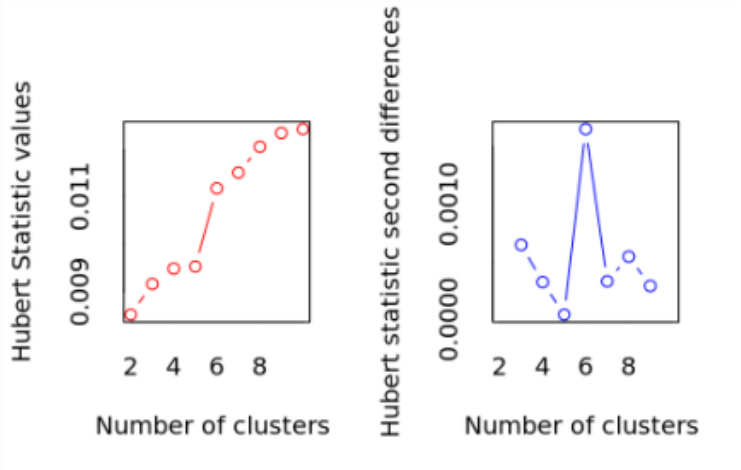
```r
kmeans_gap$Tab %>%
  as.data.frame() %>%
  rowid_to_column(var = "n_clusters") %>%
  ggplot(aes(x = n_clusters, y = gap)) +
    geom_line() +
    geom_point() +
    labs(title = "Clustering Kmeans") +
    scale_x_continuous(breaks = 1:20) +
    theme_bw()
```



...

- 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.

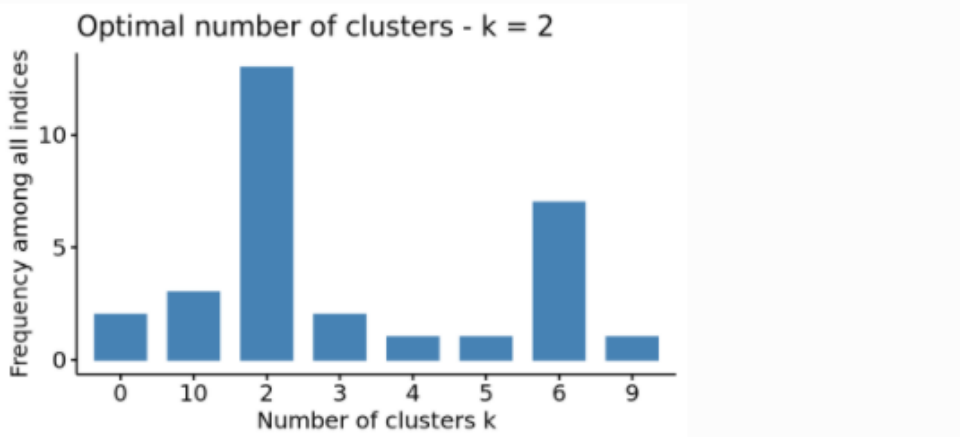
```r
library(factoextra)
library(NbClust)

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 Hube
rt
##                 index second differences plot.
##
```

```
fviz_nbclust(numero_clusters)
```

```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 13 proposed  2 as the best number of clusters
## * 2 proposed  3 as the best number of clusters
## * 1 proposed  4 as the best number of clusters
## * 1 proposed  5 as the best number of clusters
## * 7 proposed  6 as the best number of clusters
## * 1 proposed  9 as the best number of clusters
## * 3 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

- ***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.

Heatmaps (Mapas de Calor)

- 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*.

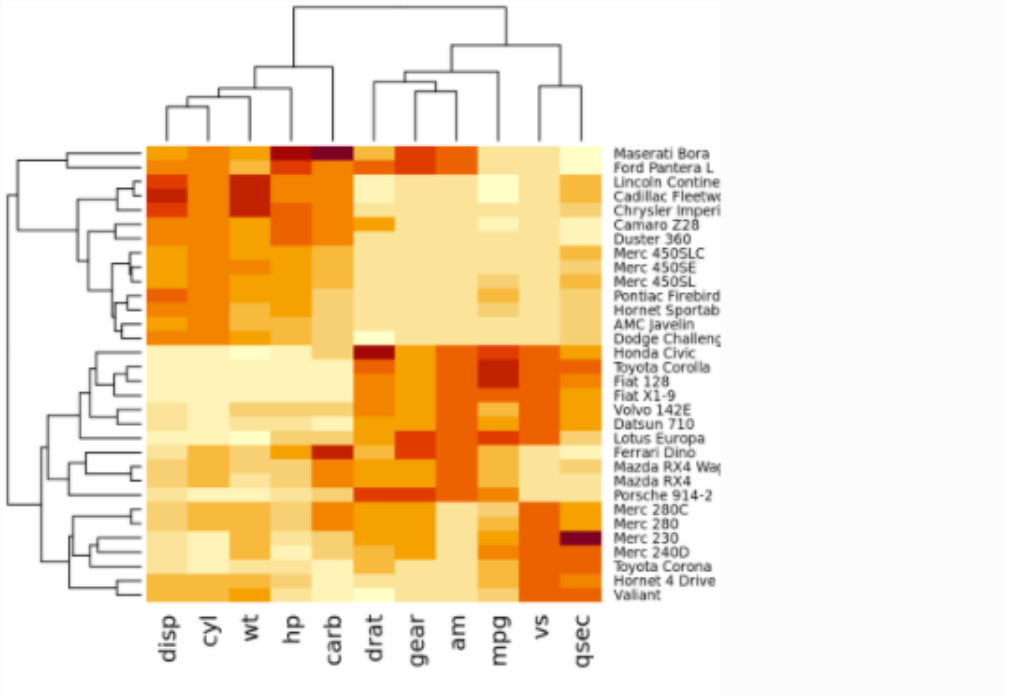
...

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.

heatmap()[stats]

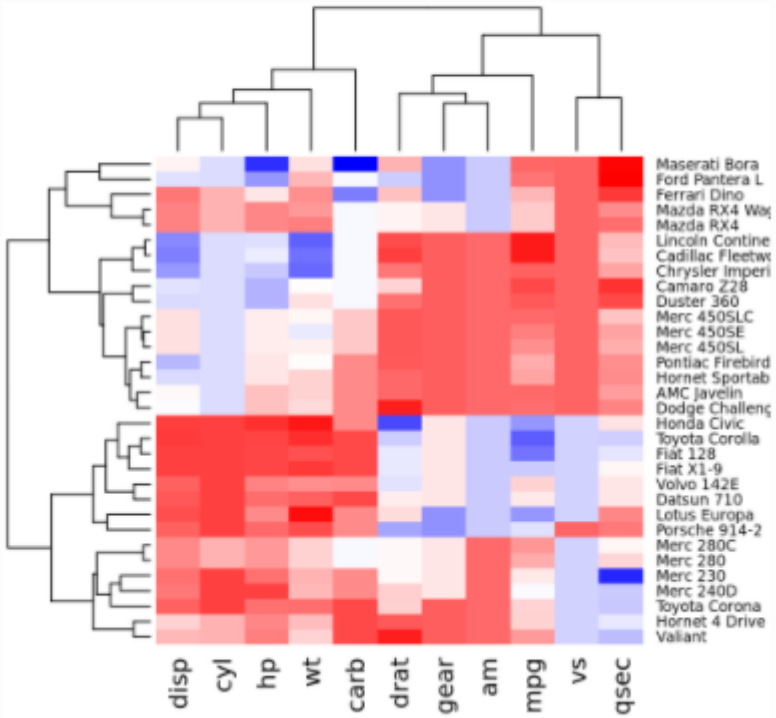
```
datos <- mtcars
# Para que las variables sean comparables bajo un mismo esquema de colores se
# estandarizan.
datos <- scale(datos)
heatmap(x = datos, scale = "none",
        distfun = function(x){dist(x, method = "euclidean")},
        hclustfun = function(x){hclust(x, method = "average")},
   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



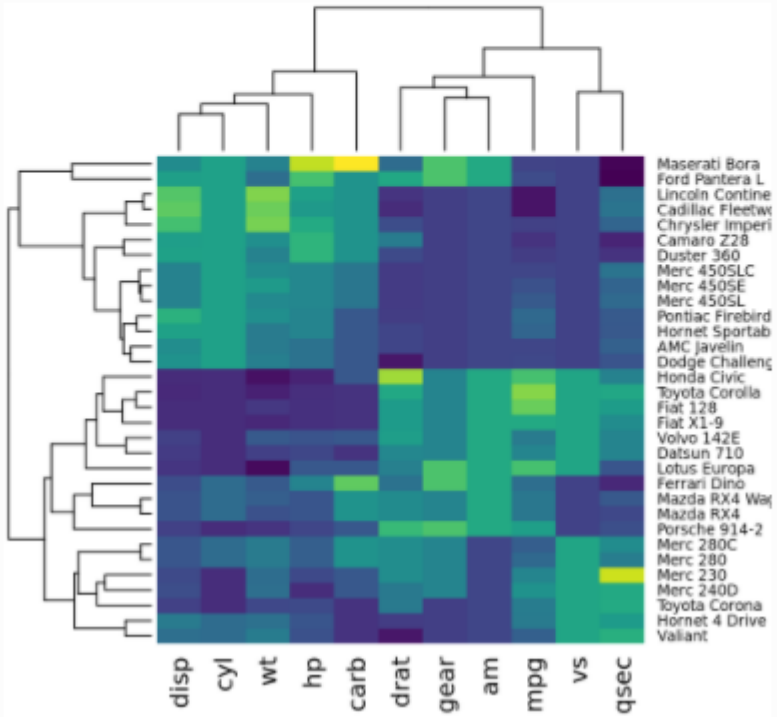
...

- 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)
```



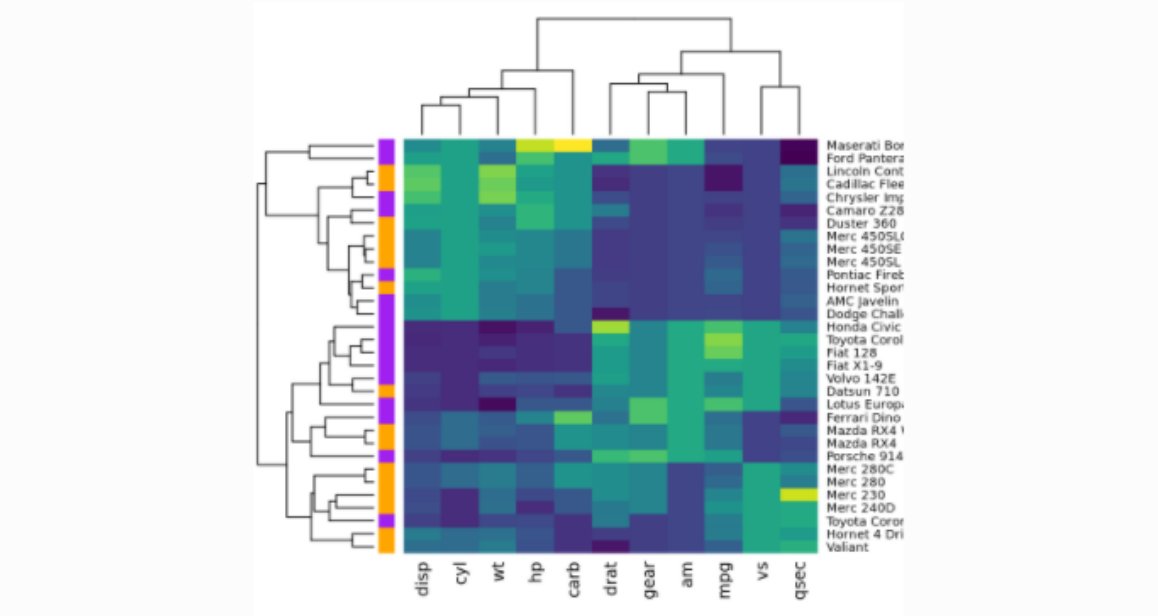
```r
# Paleta de color viridis
library(viridis)
colores <- viridis(256)
heatmap(x = datos, scale = "none", col = colores,
        distfun = function(x){dist(x, method = "euclidean")},
        hclustfun = function(x){hclust(x, method = "average")},
        cexRow = 0.7)
```



...

- 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.

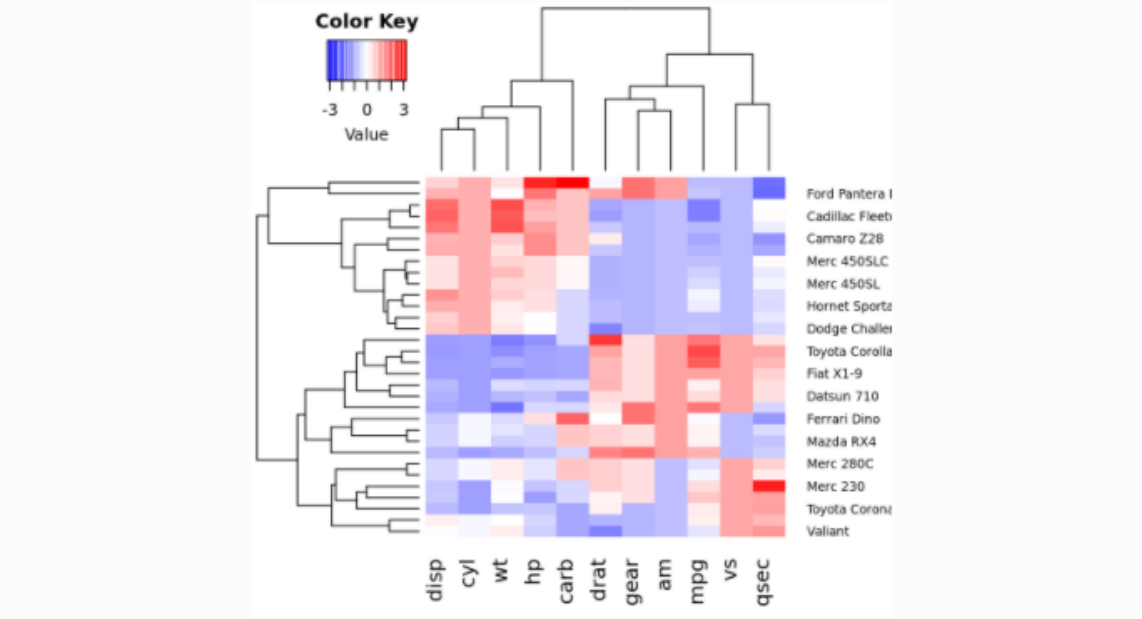
```
# Se codifica con color naranja a los coches procedentes de China y con morado a
# los de América
colores <- viridis(256)
heatmap(x = datos, scale = "none", col = colores,
        distfun = function(x){dist(x, method = "euclidean")},
        hclustfun = function(x){hclust(x, method = "average")},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

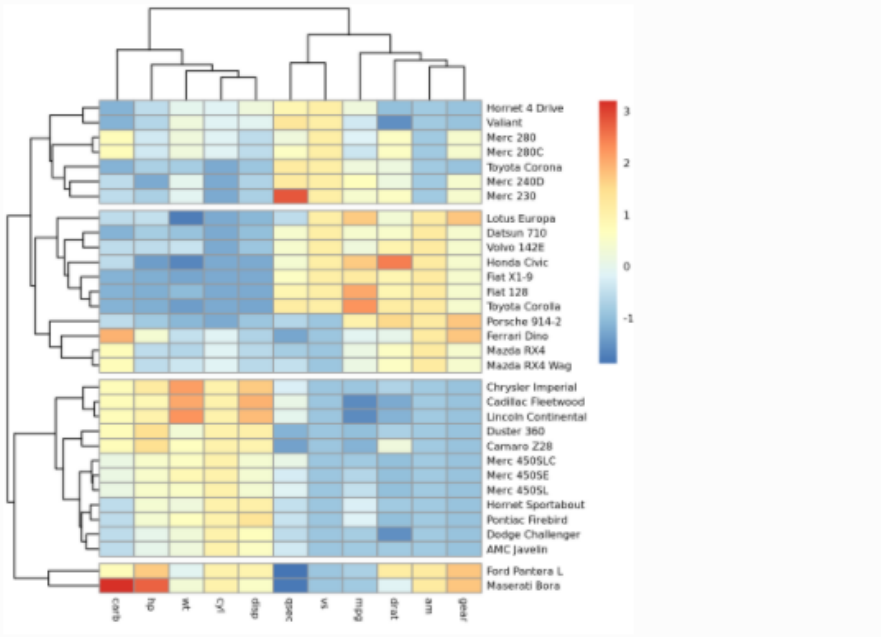
```
library(gplots)
heatmap.2(x = datos, scale = "none", col = bluered(256),
          distfun = function(x){dist(x, method = "euclidean")},
          hclustfun = function(x){hclust(x, method = "average")},
          density.info = "none",
          trace = "none", cexRow = 0.7)
```



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")})
```

Comparación de Dendrogramas

Existen varias formas de estudiar las diferencias entre dendrogramas, dos de las más utilizadas son:

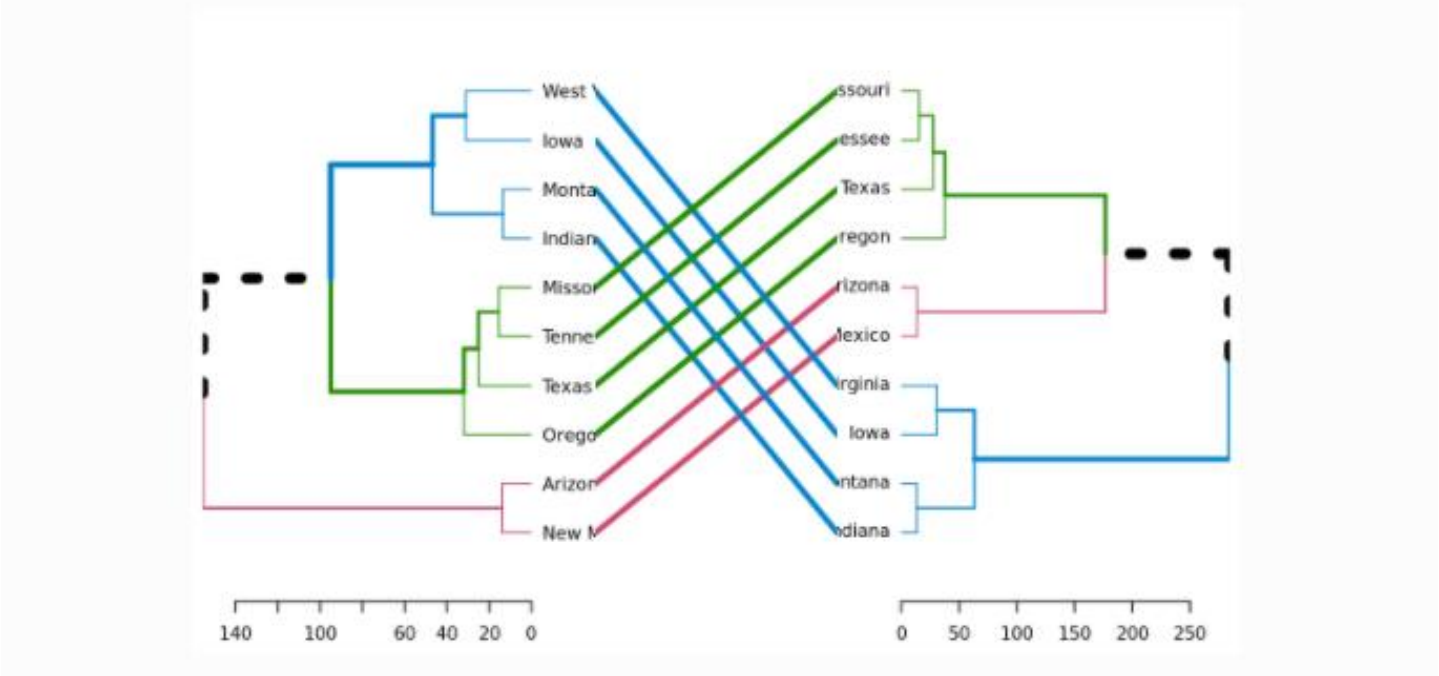
- la comparación visual y el cálculo de correlación entre dendrogramas. Ambos métodos pueden aplicarse con las funciones tanglegram() y cor.dendlist() del paquete dendextend.

```
# 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.

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



Comparación por correlación

- La función tanglegram() representa dos dendrogramas a la vez, enfrentados uno al otro, y conecta las hojas terminales con líneas.Con la función cor.dendlist() se puede calcular la matriz de correlación entre dendrogramas basada en las distancias de Cophenetic o Baker

```r
# 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 0.6915129
## [2,] 0.6915129 1.0000000
```

```r
# 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   0.692  0.984    0.650
## ward.D2    0.692   1.000  0.574    0.997
## single     0.984   0.574  1.000    0.529
## complete   0.650   0.997  0.529    1.000
```

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

```
## [1] 0.6915129
```

Preguntas

- Alguna pregunta?