Ciencia de datos, práctica 4

Juan Casado Ballesteros, Samuel García Gonzalez, Iván Anaya Martín

November 19, 2019

Abstract

A lo largo de esta práctica analizaremos y compararemos distintos métodos de clasificación no supervisada o clusterización.

Utilizaremos el algoritmo de k-mean para realizar el ejercicio propuesto y la primera parte del ejercicio libre. Nos centraremos principalmente en las técnicas de selección del número de clusters óptimo para lo cual exploraremos tres de ellas. Adicionalmente hemos relizado una implementación propia de k-means que validaremos comparándola con los resultados de la implementación de R. La ventaja de nuestra implementación frente a la de R es que nos permite visualizar paso a paso el estado de cada iteración del algoritmo. Esto nos permite ver cómo se desplazan los centroides así como ver en cada momento la clasificación actual de los elementos de la muestra.

En la segunda parte del ejercicio libre utilizaremos los algoritmos de clasificación jerárquica aglomerativo y divisivo. En el algoritmo aglomerativo partimos de que cada elemento de la muestra es una clases. En cada iteración elegiremos las dos clases más próximas y la uniremos en una nueva clase creando así una estructura de árbol. En el algoritmo divisivo todos los elementos serán consderados una única clase al inicio de la ejecución. En cada iteración separaremos la clase en dos pudiendo contener cada una de ellas uno o más elementos creando así una estructura de árbol. Las principales variaciones en estos métodos residen en la métrica utilizada para calcular la distancia entre clases a partir de la cual decidimos cuando separarlas o unirlas. Exploraremos distintas métricas, visualizaremos los árboles generados y compararemos las clasificaciones jerárquicas entre si y con los resultados de k-means.

Debido a que deseamos comparar los resultados de k-means con la clasificación jerarquica utilizaremos para explorar ambos algoritmos el mismo dataset. Se utilizará un dataset obtenido de kaggle en que se presentan las características de un conjunto de semillas.

Contents

1 K-means sobre la muestra proporcionada

Aplicaremos el algoritmo k-means sobre la muestra que se nos proporciona. Este algoritmo clasificará de forma no supervisada la muestra en tantas clases como indiquemos. En primer lugar deberemos cargar esta desde un archivo .txt.

```
> datos1 <- read.table("datos1.txt")
> datos1
```

```
  Teoria Laboratorio
1      4           4
2      3           5
3      1           2
4      5           5
5      0           1
6      2           2
7      4           5
8      2           1
```

Para esta muestra tal y como podemos ver y tal y como vimos en clase el número adecuado de centroides es 2. Ubicaremos dos puntos a partir de los cuales se ejecutará el algoritmo.

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

```
     [,1] [,2]
[1,]    0    1
[2,]    2    2
```

Realizamos la ejecución del algoritmo con la implementación que R nos proporciona.

```
> classkms<-kmeans(datos1,centroides,4)
```

1.1 Evaluación de la solución obtenida

Podemos ver a que clase pertenecen cada uno de los elementos de la muestra. Nuestra muestra tiene un total de ocho elementos que se han clasificado en dos clases (clase 1 y clase 2).

```
> classkms$cluster
```

```
1 2 3 4 5 6 7 8
2 2 1 2 1 1 2 1
```

Centroides de cada una de las dos clases. Son los puntos que minimizan las distancias a los elementos de la muestra, desde ellos a los elementos de su clase.

```
> classkms$centers
```

```
  Teoria Laboratorio
1   1.25        1.50
2   4.00        4.75
```

Cantidad de elementos que pertenecen a cada una de las clases obtenidas.

```
> classkms$size

[1] 4 4
```

Iteraciones necesarias para realizar la clasificación. Este valor no se refiere a las iteraciones del algoritmo, con una sola iteración no se hubiera podido llegar a la solución.

```
> classkms$iter

[1] 1
```

Indica si la clasificación obtenida es válida o no. K-means con unos centroides iniciales mal-intencionadamente elegidos puede no llegar a una solución válida

```
> classkms$ifault

[1] 0
```

Distancia cuadrática media entre los centroides, deseamos que este valor sea alto ya que querríamos que nuestros clusters fueran heterogéneos y bien diferenciados.

```
> classkms$betweenss

[1] 36.25
```

Distancia cuadrática media dentro de cada cluster, deseamos obtener valores bajos, es decir, dentro de cada cluster los elementos deben estar próximos, deben ser homogéneos.

```
> classkms$withinss

[1] 3.75 2.75
```

Suma de todas las distancias cuadráticas medias dentro de los clusters.

```
> classkms$tot.withinss

[1] 6.5
```

Suma de todas las distancias cuadráticas medias dentro de los clusters y la distancia cuadrática media entre clusters.

```
> classkms$totss

[1] 42.75
```

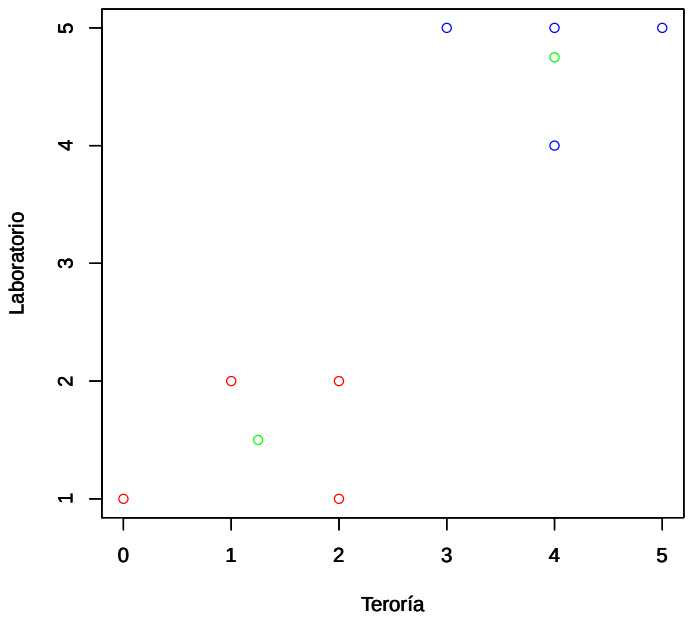
Estos últimos cuatro valores ayudan a decidir si el algoritmo ha realizado una clasificación correcta o no.

1.2 Visualización del resultado

Dividiremos la muestra en las clases que la ejecución nos ha proporcionado. Dibujaremos dichas clases junto a sus representantes en una gráfica bidimensional. Primero deberemos separar los elemntos de la muestra según la clasificación btenida.

```
> clusters <- cbind(classkms$cluster,datos1)
> cluster1 <- subset(clusters,clusters[,1]==1)
> cluster2 <- subset(clusters,clusters[,1]==2)
> cluster1 <- cluster1[,-1]
> cluster2 <- cluster2[,-1]
> clusters <- list(cluster1, cluster2)
```

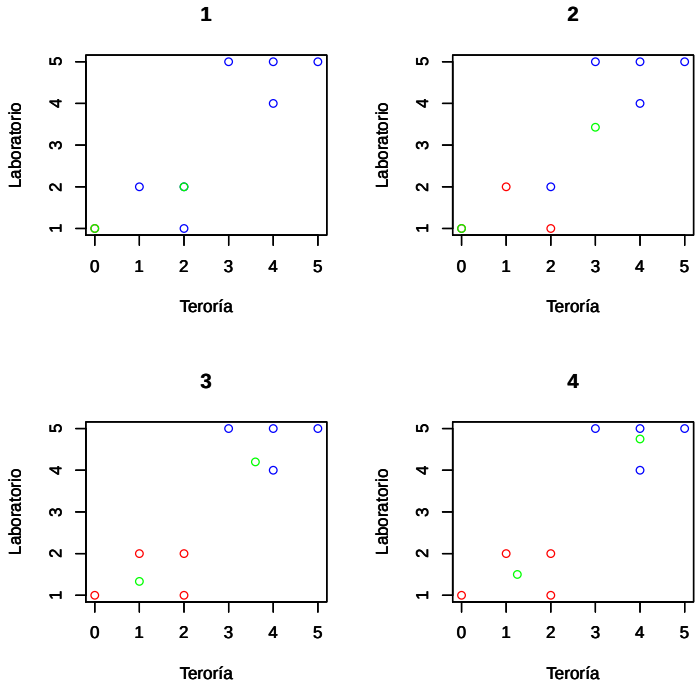
Como podemos ver la clasificación se ha realizado correctamente.



1.3 Pasos de la clasificación

Aunque establezcamos una iteración como número máximo de iteraciones al lanzar el algoritmo este siempre llega a la solución final sin proporcionar los pasos intermedios. Es por ello que decidimos implementar nuestra propia versión de k-means con la intención de poder mostrar estos pasos intermedios. A cada iteración del algoritmo extraemos y mostramos el estado de la ejecución en una gráfica distinta. Podemos ver la evolución de los centroides desde sus posiciones iniciales hasta las finales. Comprobamos que el resultado de la ejecución con nuestra implementación coincide con el de la implementación de R.

```
> current_centroides <- centroides
> par(mfrow=c(2,2))
> for (i in 1:4){
+   cluster <- kmeans_cluster (datos1, current_centroides)
+   split <- kmeans_split(datos1, cluster)
+   new_centroides <- kmeans_new_centroids(split)
+   plot_kmeans(split, current_centroides, main=toString(i),
+               xlab="Teroría", ylab="Laboratorio")
+   if (same_centroids(new_centroides, current_centroides)){
+     break
+   }else{
+     current_centroides <- new_centroides
+   }
+ }
> current_centroides

     [,1] [,2]
[1,] 1.25 1.50
[2,] 4.00 4.75
```



2 K-means

En este apartado vamos a realizar un ejemplo práctico de clustering sobre una muestra obtenida de kaggle. En dicha muestra se presentan 7 características sobre un conjunto de semillas.

Utilizaremos distintas funciones de las librerías: "tidyverse", "cluster", "factoextra" y "gridExtra" para mostrar los datos y calcular el número óptimo de clusters. El algoritmo encargado de realizar la clusterización será kmeans del que utilizaremos la implementación que R proporciona.

2.1 Preparar los datos

Deberemos eliminar los datos que estén incompletos ya que kmeans no podría manejarlos. Adicionalmente escalaremos las variables con la intención de hacerlas comparables entre ellas. Esto es necesario debido a que la escala de las variables de entrada adecta al algoritmo pues se utilizar distancias euclídeas para realizar la clusterización. Escalar las variables consiste en hacer que la media de cada una de ellas sea 0 y desviación estándar de 1. Tras haber realizado el escalado aparecerán por tanto valores negativos en las características, estas dejan de estar en sus unidades originales.

```
> df <- read.csv("Seed_Data.csv")
> df <- na.omit(df)
> df <- scale(df)
```

Mostraramos a continuación un extracto de los datos.

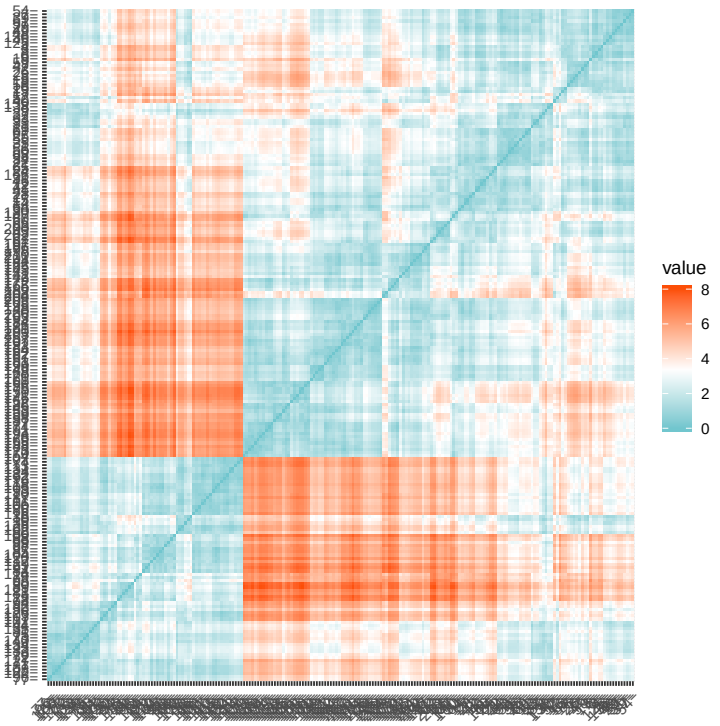
```
> head(df)
```

```
            A           P           C          LK        WK     A_Coef
1  0.14175904  0.21494882 6.045733e-05  0.30349301 0.1413640 -0.9838010
2  0.01116136  0.00820415 4.274938e-01 -0.16822270 0.1969616 -1.7839036
3 -0.19160873 -0.35934192 1.438945e+00 -0.76181710 0.2075516 -0.6658882
4 -0.34626388 -0.47420007 1.036904e+00 -0.68733567 0.3187467 -0.9585276
5  0.44419577  0.32980697 1.371233e+00  0.06650665 0.8032397 -1.5597684
6 -0.16067770 -0.26745540 1.019976e+00 -0.54740087 0.1413640 -0.8235144
        LKG
1 -0.3826631
2 -0.9198156
3 -1.1863572
4 -1.2270506
5 -0.4742231
6 -0.9198156
```

2.2 Conocer los datos

Antes de poder realizar cualquier análisis de datos debemos de conocerlos. En este caso utilizaremos una matriz de distancias para poder hacerlo. Dicha matriz nos mostrará las distancias entre los elementos que manejamos. En ella podemos observar que las distancias son heterogéneas entre los elementos, es decir, hay variedad de distancias. Entre otras cosas la matriz puede ayudarnos a elegir una distancia euclídea para realizar con ella la clasificación seleccionando una que acentúe las desigualdades entre clases.

```
> dist <- get_dist(df)
> fviz_dist(dist, gradient = list(low = "#00AFBB", mid = "white", high = "#FC4E07"))
```



2.3 Ejecución del algoritmo

Ejecutaremos el algoritmo kmeans implementado en R. En este caso en vez de fijar nosotros los centroides iniciales indicaremos que queremos obtener 2 clases distintas siendo el propio algoritmo quien los establezca. Adicionalmente indicaremos que queremos probar un total de 25 combinaciones de centroides iniciales. Dependidendo de las posiciones iniciales obtendremos distintos resultados en la clasificación, ejecuatando el algoritmo varias veces nos aseguramos de quedarnos con una clusterización adecuada.

```
> k2 <- kmeans(df, centers = 2, nstart = 25)
```

Obtenemos dos centroides, uno para cada clase, además visualizamos la cantidad de elementos en cada una de las clases.

```
> k2$centers
```

```
           A         P          C         LK         WK      A_Coef         LKG
1  1.1379346  1.145151  0.5424726  1.1260130  1.0640703 -0.14617607  1.1468793
2 -0.6588042 -0.662982 -0.3140631 -0.6519023 -0.6160407  0.08462825 -0.6639828
```

```
> k2$size
```

```
[1]  77 133
```

Adicionalmente mostramos las distancias cuadráticas medias entre los elementos de cada cluster y entre los clusters. Utilizaremos estas medidas posteriormente para compararlas con la clasificación mediante otros números de clusters.

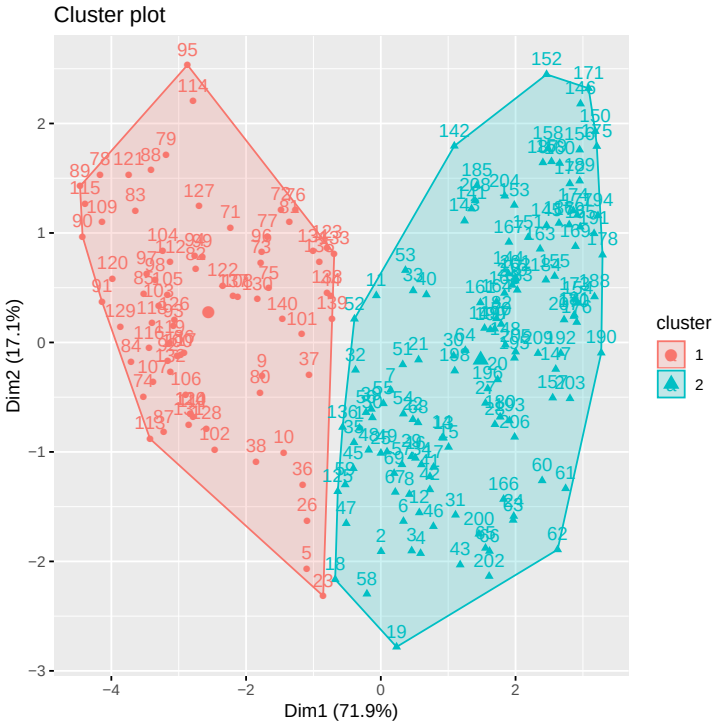
```
> k2$withinss
```

```
[1] 196.2357 459.7972
```

```
> k2$betweenss
```

```
[1] 806.9672
```

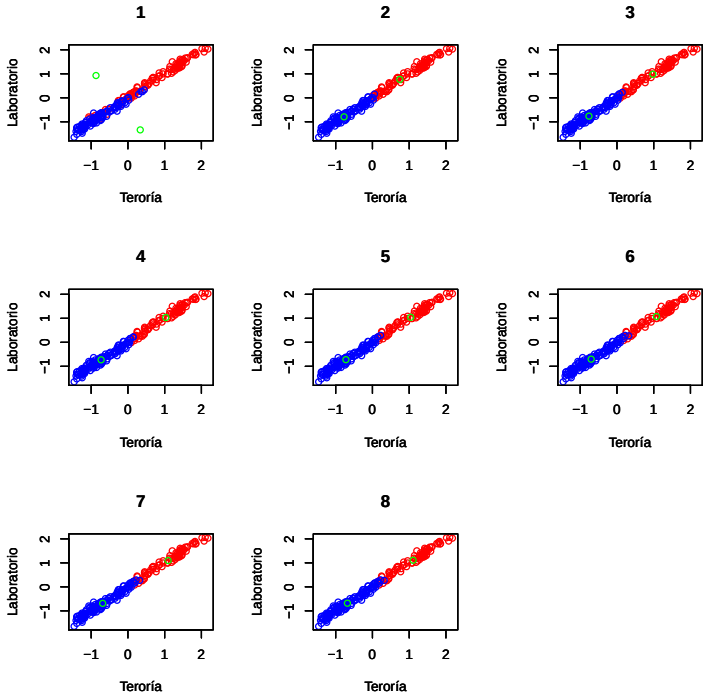
Mostramos la clasificación obtenida.


Cluster plot
Dim2 (17.1%)
Dim1 (71.9%)
cluster
1
2

2.4 Ejecución con implementación propia

Realizaremos la misma ejecución utilizando ahora la implementación de kmeans hecha por nosotros. Para realizar la ejecución tomaremos un vector aleatorio de tantos elementos como el número de clases multiplicado por la cantidad de variables asociadas a cada dato. Tras la ejecución podremos ver en las gráficas como los centroides logran clasificar los elementos en 5 iteraciones. Adicionalmente podemos comprobar que los centrides obtenidos con nuestra implementación y con la de R son los mismos.

```
> datos <- df
> current_centroides <- matrix(runif(14, -2, 2), 2,7)
> par(mfrow=c(3,3))
> for (i in 1:9){
+   cluster <- kmeans_cluster (datos, current_centroides)
+   split <- kmeans_split(datos, cluster)
+   new_centroides <- kmeans_new_centroids(split)
+   plot_kmeans(split, current_centroides, main=toString(i),
+               xlab="Teroría", ylab="Laboratorio")
+   if (same_centroids(new_centroides, current_centroides)){
+     break
+   }else{
+     current_centroides <- new_centroides
+   }
+ }
> current_centroides

           [,1]       [,2]       [,3]       [,4]       [,5]        [,6]
[1,]  1.1060769  1.1081623  0.5740801  1.0691564  1.0501524 -0.16202391
[2,] -0.6806627 -0.6819461 -0.3532801 -0.6579424 -0.6462476  0.09970702
           [,7]
[1,]  1.0887331
[2,] -0.6699896
```



Comprobamos adicionalmente que ambas clasificaciones, la de R y la nuestara coinciden como método de validación de nuestro algoritmo. El resultado es positivo.

```
> clusters <- cbind(k2$cluster,datos)
> cluster1 <- subset(clusters,clusters[,1]==1)
> cluster2 <- subset(clusters,clusters[,1]==2)
> cluster1 <- cluster1[,-1]
> cluster2 <- cluster2[,-1]
> clusters <- list(cluster1, cluster2)
> compare_clusters(clusters, split)

[1] 0.9857143
```

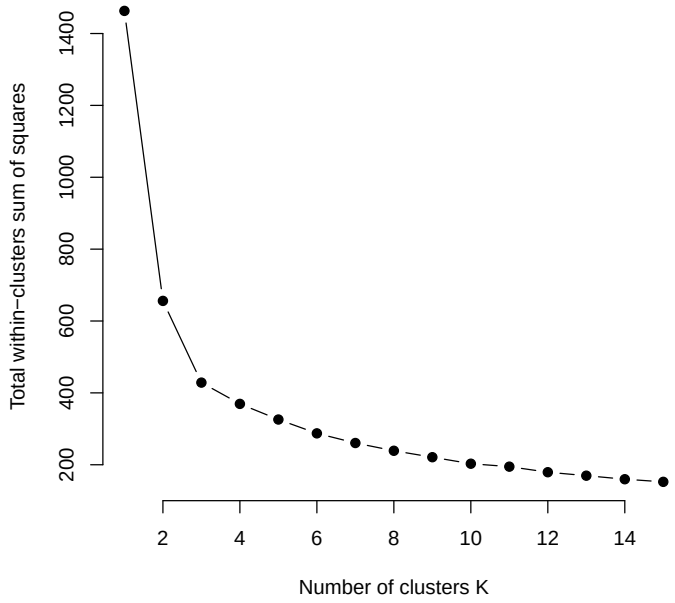
2.5 Elección del número de clusters

Hasta ahora hemos elegido siempre 2 como el número de clusters. No obstante esta decisión no debiera ser trivial. Expondremos a continuación una serie de métodos que pueden guiarnos a la hora de tomar esta decisión.

2.5.1 Elbow Method

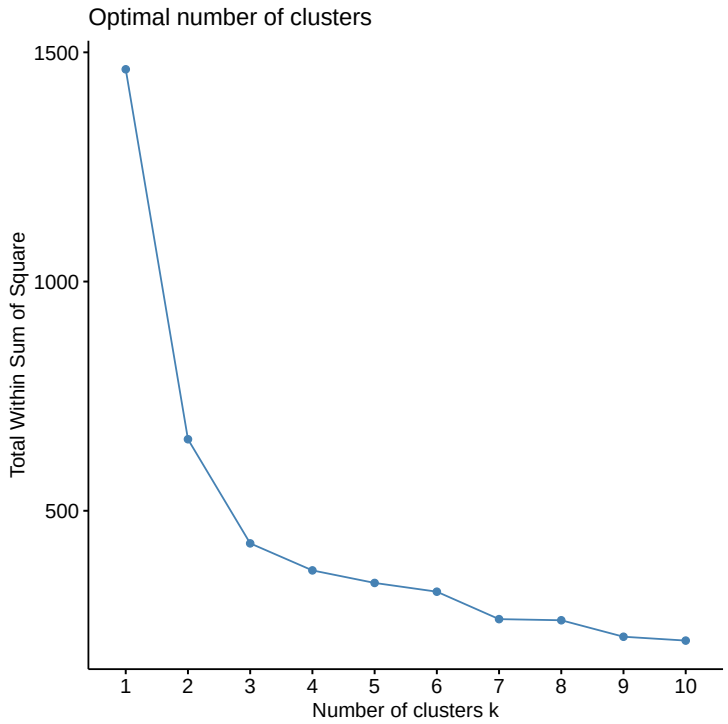
Este método se basa en calcular la suma de las distancias cuadráticas medias entre los elementos de los clusters y dibujarlas en una gráfica. Este valor es el proporcionado en "tot.withinss" al ejecutar kmeans tal y como ya explicamos en el primer apartado. En la gráfica el número de cluster óptimos será aquel en el que se produzca un codo en la gráfica. La suma de las distancias se irá reduciendo hasta llegar a 0 cuando haya tantas clases como elementos. El punto de nuestro interés está en el momento en el que la gráfica comienza a descender a un ritmo menor.

```
> wss <- function(k) {
+   kmeans(df, k, nstart = 10 )$tot.withinss
+ }
> k.values <- 1:15
> wss_values <- map_dbl(k.values, wss)
> plot(k.values, wss_values,
+        type="b", pch = 19, frame = FALSE,
+        xlab="Number of clusters K",
+        ylab="Total within-clusters sum of squares")
```



La siguiente función ya implementa este método de modo que no es necesario realizarlo manualmente.

```
> fviz_nbclust(df, kmeans, method = "wss")
```

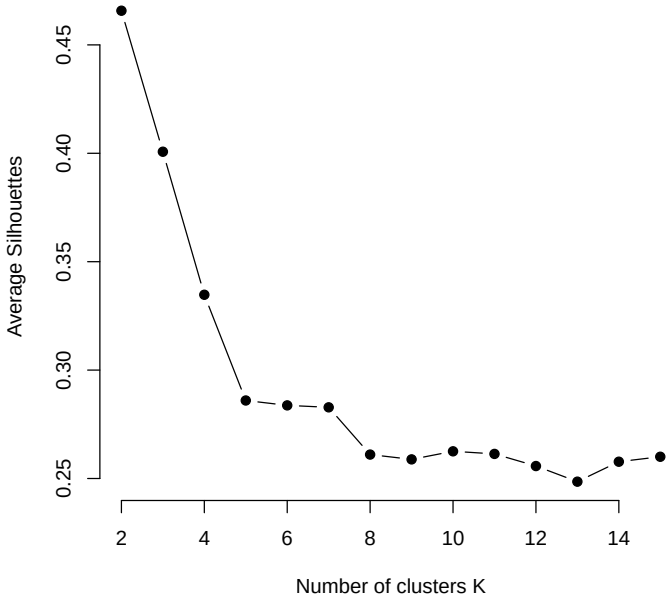


Podemos ver que el número de clusters óptimo estaría entre los valores de 2 y 4 clases.

2.5.2 Average Silhouette Method

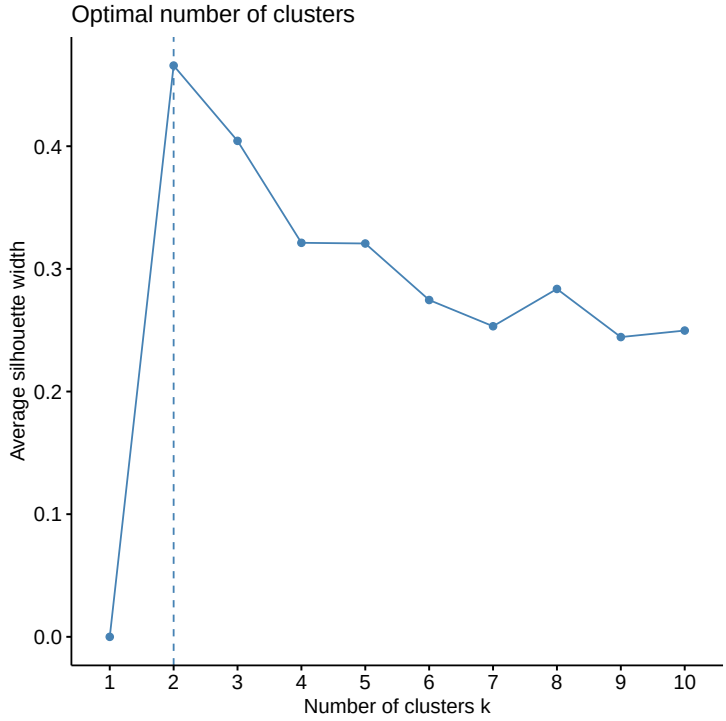
Este método trata de calcular la paertenencia de cada elemento al cluster en el que ha sido clasificado. Lo calcularemos manualmente para los valores de 2 a 15 clases. Buscamos en esta gráfica aquellos números de clusters que maximizan la pertencencia de sus elementos.

```
> avg_sil <- function(k) {
+   km.res <- kmeans(df, centers = k, nstart = 25)
+   ss <- silhouette(km.res$cluster, dist(df))
+   mean(ss[, 3])
+ }
> k.values <- 2:15
> avg_sil_values <- map_dbl(k.values, avg_sil)
> plot(k.values, avg_sil_values,
+        type = "b", pch = 19, frame = FALSE,
+        xlab = "Number of clusters K",
+        ylab = "Average Silhouettes")
```



El mismo método está ya implementado en una única función.

```
> fviz_nbclust(df, kmeans, method = "silhouette")
```

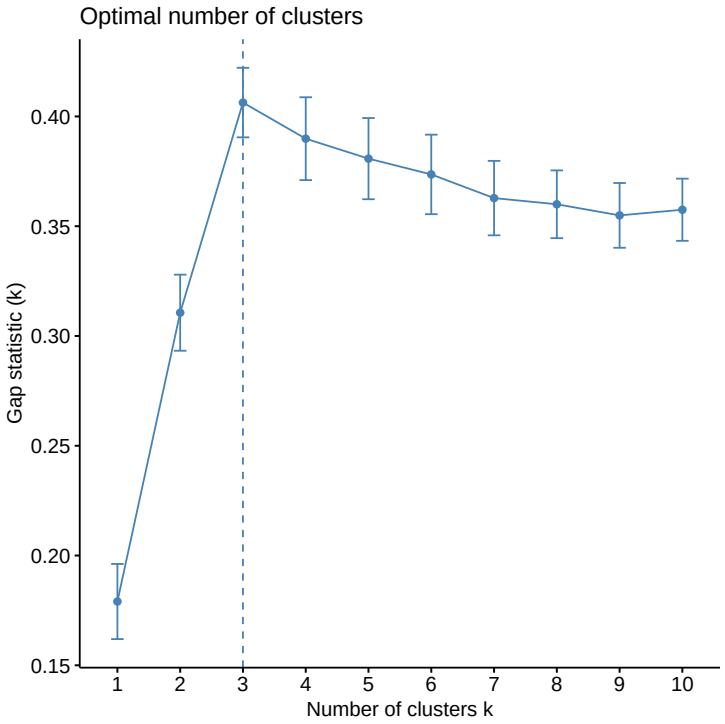


Podemos ver que 2 clases tiene la mayor pertenencia de los elementos clasificados a los clusters seguido de 4 clases. Estos valores coinciden con los obtenidos con el método anterior.

2.5.3 Gap Statistic Method

Este es un método más complejo que los anteriores. La explicación del mismo puede verse en: http://web.stanford.edu/ hastie/Papers/gap.pdf. Como resultados se obtiene una gráfica cuyo valor más alto será el mejor número de clusters. Junto a cada valor se nos indica la calidad de la estimación. Buscamos la combinación de un valor más alto en la gráfica con unos bigotes de menor tamaño.

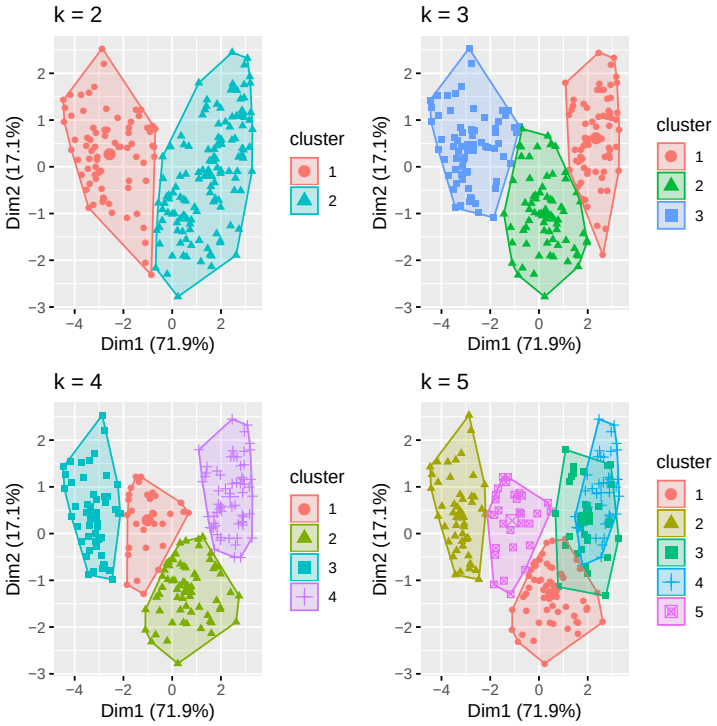
```
> gap_stat <- clusGap(df, FUN = kmeans, nstart = 25, K.max = 10, B = 50, verbose=FALSE)
> fviz_gap_stat(gap_stat)
```



Podemos ver que en este caso 3 sería el mejor número de cluster en los que realizar nuestra clasificación.

Mostramos a continuación las distintas clasificciones que se obtienen cuando tenemos de 3 a 5 clases.

```
> k3 <- kmeans(df, centers = 3, nstart = 25)
> k4 <- kmeans(df, centers = 4, nstart = 25)
> k5 <- kmeans(df, centers = 5, nstart = 25)
> p1 <- fviz_cluster(k2, geom = "point", data = df) + ggtitle("k = 2")
> p2 <- fviz_cluster(k3, geom = "point",  data = df) + ggtitle("k = 3")
> p3 <- fviz_cluster(k4, geom = "point",  data = df) + ggtitle("k = 4")
> p4 <- fviz_cluster(k5, geom = "point",  data = df) + ggtitle("k = 5")
> grid.arrange(p1, p2, p3, p4, nrow = 2)
```



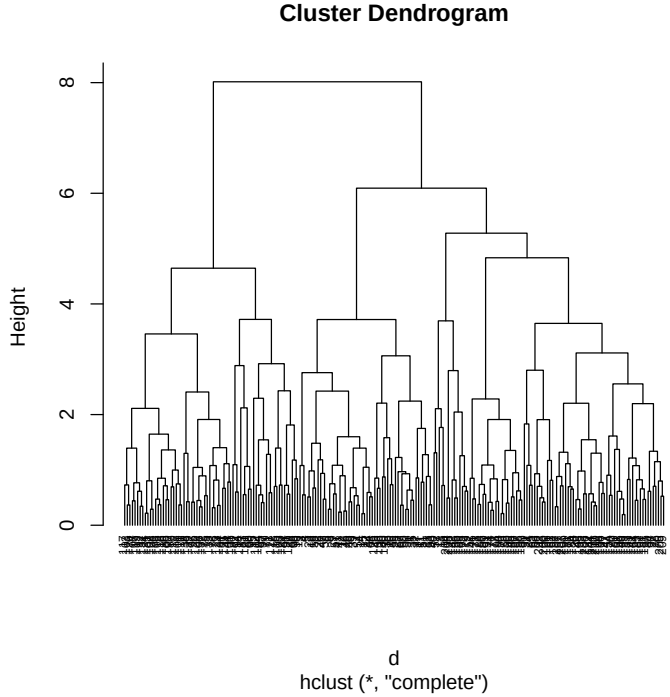
Finalmente mostramos la suma de las distancias entre los elementos de los clusters y entre los clusters. Vemos como al aumentar el número de clusters la calidad de la clasificación dentro de los clusters aumenta mientra que la calidad de la clasificación entre cluster disminuye. Es por tanto que la elección del número de cluster es una optimización de estas dos variables teniendo en cuenta que puede que nos interese favorecer a una frente a la otra.

```
> c(k2$betweens, k3$betweens, k4$betweens, k5$betweens)

[1]  806.9672 1034.3918 1093.5829 1138.0440

> c(k2$tot.withinss, k3$tot.withinss, k4$tot.withinss, k5$tot.withinss)

[1] 656.0328 428.6082 369.4171 324.9560
```

3 Clusterización jerárquica

Utilizaremos y compararemos el método de clusterización jerárquica dependiendo de la medida de distancia que utilicemos para ejecuatarlo. Utilizaremos funciones de las siguiente librerías: "cluster", "purrr", "dendextend" y "factoextra".

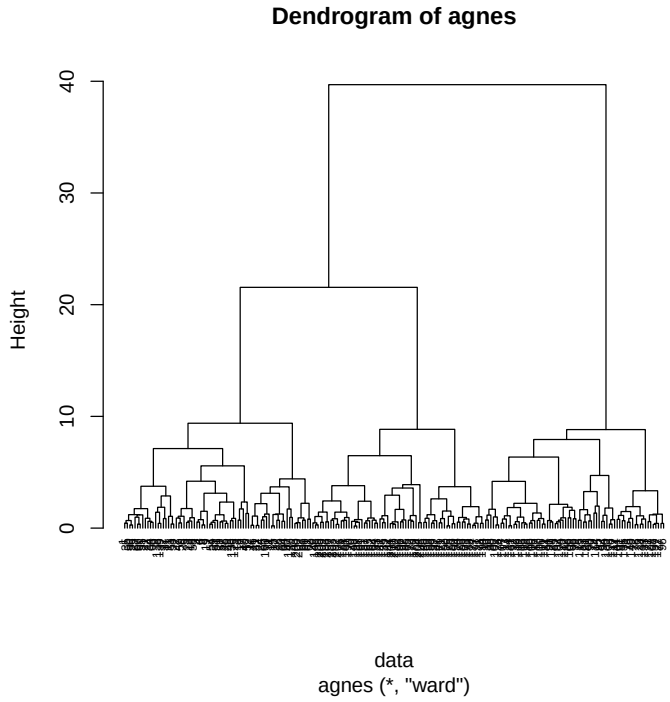
```
> data <- read.csv("Seed_Data.csv")
> data <- na.omit(data)
> data <- scale(data)

> d <- dist(data, method = "euclidean")
> hc1 <- hclust(d, method = "complete")
> plot(hc1, cex = 0.6, hang = -1)
```




```
> hc2 <- agnes(data, method = "complete")
> hc2$ac

[1] 0.9231512

> m <- c( "average", "single", "complete", "ward")
> names(m) <- c( "average", "single", "complete", "ward")
> ac <- function(x) {
+   agnes(data, method = x)$ac
+ }
> map_dbl(m, ac)
```

```
  average    single  complete      ward
0.8649547 0.6055379 0.9231512 0.9846264
```

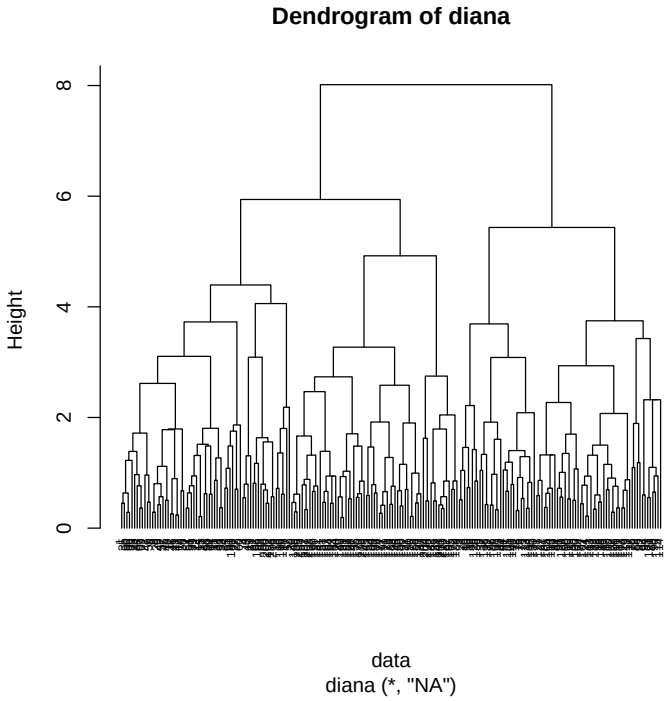
```
> hc3 <- agnes(data, method = "ward")
> pltree(hc3, cex = 0.6, hang = -1, main = "Dendrogram of agnes")
```



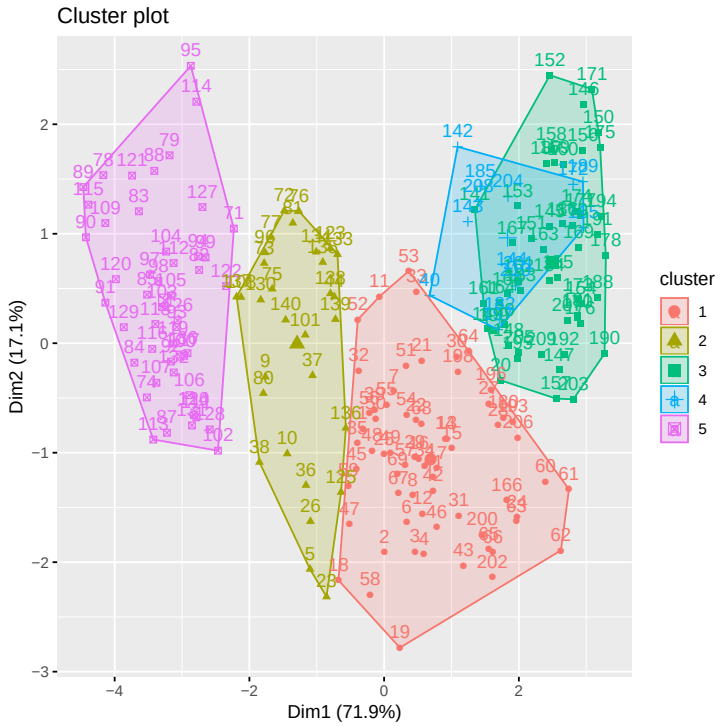

```
> hc4 <- diana(data)
> hc4$dc
```

```
[1] 0.918148
```

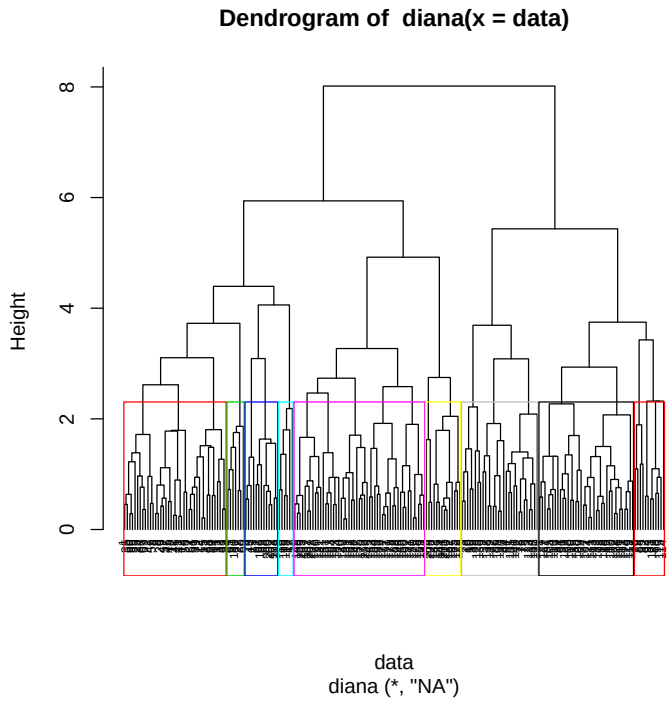
```
> pltree(hc4, cex = 0.6, hang = -1, main = "Dendrogram of diana")
```



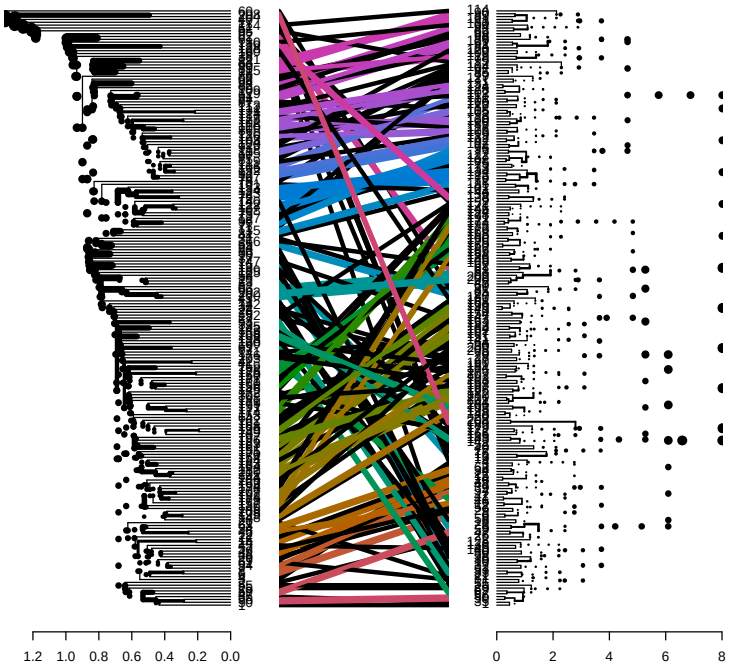

```
> clust <- cutree(hc4, k = 5)
> fviz_cluster(list(data = data, cluster = clust))
```



```
> pltree(hc4, hang=-1, cex = 0.6)
> rect.hclust(hc4, k = 9, border = 2:10)
```




```
> hc_single <- agnes(data, method = "single")
> hc_complete <- agnes(data, method = "complete")
> hc_single <- as.dendrogram(hc_single)
> hc_complete <- as.dendrogram(hc_complete)
> tanglegram(hc_single,hc_complete)
```


1.2 1.0 0.8 0.6 0.4 0.2 0.0
0 2 4 6 8

4 Funciones implementadas

Las funcines creadas son las que implementan k-means así como una función adicional para crear las gráficas con los resultados.

```
> distance
```

```
function (point1, point2) {
  acc <- 0
  len <- length(point1)
  for (i in 1:len){
    acc <- acc + (point1[i]-point2[i])^2
  }
  acc^(1/2)
}
<bytecode: 0x7ff9c3d663f8>
```

```
> kmeans_cluster
```

```
function (data, centroids) {
  d_default <- distance(c(max(data), max(data)), c(min(data), min(data)))
  classification <- c()
  for (j in 1:nrow(data)){
      point <- data[j,]
      best_c <- 0
      best_d <- d_default
      for (i in 1:nrow(centroids)){
        centroid <- centroids[i,]
        d <- distance (point, centroid)
        if (d < best_d){
          best_d <- d
          best_c <- i
        }
    }
    classification <- c(classification, best_c)
  }
  classification <- as.data.frame(classification)
  rownames(classification) <- rownames(data)
  classification
}
<bytecode: 0x7ff9ca8805b8>
```

```
> kmeans_split
```

```
function (data, cluster) {
  clusters=cbind(cluster,data)
  cluster1=subset(clusters,clusters[,1]==1)
  cluster2=subset(clusters,clusters[,1]==2)
  cluster1=cluster1[,-1]
  cluster2=cluster2[,-1]
  list(cluster1, cluster2)
}
<bytecode: 0x7ff9c4417c90>
```

```
> kmeans_new_centroids

function(split){
  centroids <- c()
  for (cluster in split) {
    for (colum in cluster) {
      acc <- 0
      for (element in colum){
        acc <- acc + element/length(colum)
      }
      centroids <- c(centroids, acc)
    }
  }
  t(matrix(centroids,length(split[[1]]),length(split)))
}
<bytecode: 0x7ff9c6a2a188>

> same_centroids

function(c1, c2){
  len <- length(c1)
  ans <- T
  for (i in 1:len){
    if (c1[i]!=c2[i]){
      ans <- F
    }
  }
  ans
}
<bytecode: 0x7ff9c24f5988>

> plot_kmeans

function (clusters, centers, main="", xlab="", ylab="") {
  maxx <- max(clusters[[1]][,1])
  maxy <- max(clusters[[1]][,2])
  minx <- min(clusters[[1]][,1])
  miny <- min(clusters[[1]][,2])
  for (cluster in clusters){
    maxx <- max(maxx, max(cluster[,1]))
    maxy <- max(maxy, max(cluster[,2]))
    minx <- min(minx, min(cluster[,1]))
    miny <- min(miny, min(cluster[,2]))
  }
  color_i <- 1
  colors = c("red", "blue", "pink", "yellow", "black", "brown")
  for (cluster in clusters){
    plot( cluster[,1], cluster[,2], type="p", col=colors[color_i],main=main,
          xlim=c(minx, maxx), ylim=c(miny, maxy), xlab=xlab, ylab=ylab)
    par(new=TRUE)
    color_i <- (color_i%%(length(colors)+1))+1
```

```
  }
  plot( centers[,1], centers[,2], type="p", col="green",
        xlim=c(minx, maxx), ylim=c(miny, maxy), xlab=xlab, ylab=ylab)
}
<bytecode: 0x7ff9cc756400>

> compare_clusters

function(c1, c2) {
  len <- length(c1)
  number_of_elements <- 0
  for (i in 1:len){
    number_of_elements <- number_of_elements + nrow(c1[[i]])
  }
  same <- 0
  for (j in 1:len){
    current_same = 0
    for (i in 1:len){
      r_results = clusters[[i]]
      our_results = split[[((i+j)%%len)+1]]
      if (nrow(r_results) < nrow(our_results)){
        temp <- r_results
        r_results <- our_results
        our_results <- temp
      }
      for (name in rownames(r_results)){
        if (name %in% rownames(our_results)){
          current_same <- current_same+1
        }
      }
    }
    same <- max(same, current_same)
  }
  same/number_of_elements
}
<bytecode: 0x7ff9cac9a278>
```