Clustering

Minería de datos: aprendizaje no supervisado y detección de anomalías

Francisco Luque Sánchez

21/12/2019

1 Introducción

En este trabajo se van a estudiar técnicas de agrupamiento de ejemplos de una base de datos desde el punto de vista del aprendizaje no supervisado. En el paradigma de aprendizaje no supervisado no se dispone de un conjunto de etiquetas o categorías en las que agrupar los datos y que son conocidas a priori, si no que se tratan de formar grupos de elementos (*clusters*) en función de la similaridad que hay entre ellos. Esta similaridad está usualmente definida en función de la distancia que existe entre ellos dentro del espacio de características. Cuanto menor sea la distancia entre dos elementos de nuestro conjunto, más probable es que exista alguna relación entre los mismos, y por tanto será deseable que el algoritmo de agrupamiento los coloque dentro del mismo grupo. Por el contrario, cuanto más alejados estén los elementos en el espacio de características, más diferencia existirá entre ellos, y por tanto nos interesará que se emplacen en *clusters* diferentes.

Por tanto, para tener bien definido un algoritmo de agrupamiento, tendremos que definir dos cosas. Por un lado, habrá que establecer una medida de distancia o similaridad entre cualesquiera dos elementos de nuestro conjunto de datos. Esta distancia dependerá del tipo de datos con los que estemos trabajando. Si los datos con los que estamos trabajando están compuestos por atributos numéricos continuos, la distancia a utilizar será usualmente la distancia euclídea, o una distancia de Minkowski (la distancia euclídea es un caso particular de una distancia de Minkowski, cuando k=2). En cambio, si la información está codificada como un conjunto de variables binarias o nominales, es posible que estas distancias no estén bien definidas, o no tengan sentido, y habrá que recurrir a otras funciones de distancia. Otros tipos de datos, como textos, imágenes o series temporales, necesitarán también de medidas de distancia propias que nos permitan comparar dos elementos. En el desarrollo de esta práctica se estudiará cómo el uso de distintas medidas de distancia pueden dar lugar a resultados distintos. Por otro lado, habrá de definir qué estrategia se utiliza para agrupar los puntos una vez que conocemos la distancia que existe entre ellos. En esta práctica se estudiarán distintas políticas de agrupamiento, concretamente las siguientes:

- k-means
- DBSCAN
- Clustering jerárquico
- k-medioides
- k-means difuso

Para cada algoritmo de los dictados anteriormente, se hará una breve descripción teórica del mismo, y se harán pruebas para observar los resultados que obtiene el mismo sobre un determinado dataset. Además, para algunos de los algoritmos, estudiaremos cómo el cambio en la distancia utilizada produce distintos resultados.

1.1 Conjunto de datos empleado - Bankloan

El conjunto de datos sobre el que trabajaremos tiene el nombre de Bankloan dataset. Es uno de los conjuntos de datos de ejemplo que se distribuyen con el software IBM SPSS. El objetivo de dicho conjunto de datos es una tarea de clasificación, en la que se pide predecir si un conjunto de potenciales clientes incurrirá o no en impago. El conjunto está compuesto por 700 registros de entrenamiento, para los cuales tenemos registrado si incurrieron o no en impago, y 150 registros de test, para los cuales tenemos que predecir dicha etiqueta. En este trabajo, descartaremos los últimos 150 registros a priori, ya que nos interesa saber el valor de impago para los individuos a la hora de calcular algunas de las medidas que nos indicarán la calidad del modelo que hemos calculado.

En cuanto a las variables que se registran, se tienen datos de la edad del individuo, su nivel educativo, los años de empleo que lleva en su actual empresa, información de deudas e ingresos, y tres columnas que indican la probabilidad de que el individuo haya incurrido en impago (no se nos indica cómo se han calculado dichas probabilidades). En total, el conjunto está compuesto por 11 atributos más la clase. De estos atributos, dos de ellos son nominales (la clase, que toma dos valores distintos, y el nivel educativo, que toma cinco valores) y el resto son variables numéricas. Tomaremos la columna que indica el nivel educativo del individuo como una característica nominal porque no conocemos si los valores que toma esta variable corresponden a una escala ordenada, o son simplemente un código. El considerarlos una escala ordinal cuando realmente no lo son puede introducir un sesgo en la distancia que se calcula entre los elementos en esta variable, comportamiento que queremos evitar.

Como hemos dicho anteriormente, aunque el conjunto de datos está pensado originalmente para ser utilizado en un problema de clasificación, nosotros lo utilizaremos para un problema de agrupamiento. Utilizaremos las etiquetas para comprobar si los algoritmos de agrupamiento son capaces de separar correctamente los datos en las dos clases.

Comenzamos estudiando el algoritmo de *clustering* k-medias

2 K-medias

El algoritmo de clustering k-medias (también conocido como *k-means*) es un algoritmo que trata de particionar el espacio de características en k conjuntos distintos, tratando de minimizar la distancia media intra-cluster (dicha cantidad es la media de las distancia de cada punto del cluster a su centro). Formalmente, el objetivo es agrupar n observaciones en k conjuntos $S = \{S_1, ..., S_n\}$ tal que se minimice la siguiente suma:

$$\sum_{i=1}^{k} \sum_{x \in S_i} d(x, \mu_i)^2$$

Donde μ_i es el centro del clúster, y $d(v, w)^2$ es la distancia al cuadrado del vector v al vector w. Debido a que nos encontramos ante un problema de optimización de complejidad *NP-duro*, no suele darse la solución óptima al problema. En su lugar, se utilizan heurísticas de cálculo que dan soluciones aproximadas. La más conocida es el algoritmo de Lloyd. Este algoritmo calcula la solución al problema de las k-medias de forma iterativa, hasta que se llega a un punto estable. El algoritmo se divide en dos fases, una de asignación de puntos a los clusters y otra de actualización de los centros. El pseudocódigo del algoritmo es el siguiente:

1. Se generan aleatoriamente los k centros en el espacio de características. Existen varias técnicas para inicializar los puntos.
2. Mientras no se haya llegado a un punto estable:

- Asignación: Cada punto del conjunto de datos es asignado al cluster cuyo centro le es más cercano
- Actualización: Para cada cluster, se recalcula el centro del mismo a partir de la media de los puntos que pertenecen a él

3. El algoritmo termina cuando en una iteración todos los puntos son asignados al cluster al que pertenecían en la iteración anterior.

2.1 Aplicación del algoritmo

Una vez hemos dado una descripción teórica del algoritmo, pasamos a estudiar cómo aplicarlo. En primer lugar, aplicaremos el algoritmo utilizando sólo las variables numéricas de las que disponemos. Comenzamos cargando el conjunto de datos y seleccionando las variables numéricas que nos interesan.

```
dataset.path <- "dataset/bankloan-spss.csv"
dataset <- read.csv(dataset.path, sep = ";", dec=",")
dataset <- dataset[1:700,]

numeric.vars <- c(
    'ingresos', 'deudaingr', 'deudacred',
    'deudaotro', 'empleo',  'direccion', 'edad'
)

numeric.data <- dataset %>% select(numeric.vars)

kable(head(numeric.data))
```

ingresos	deudaingr	deudacred	deudaotro	empleo	direccion	edad
176	9.3	11.359392	5.008608	17	12	41
31	17.3	1.362202	4.000798	10	6	27
55	5.5	0.856075	2.168925	15	14	40
120	2.9	2.658720	0.821280	15	14	41
28	17.3	1.787436	3.056564	2	0	24
25	10.2	0.392700	2.157300	5	5	41

Un problema que se presenta cuando se trabaja con algoritmos basados en cálculo de distancias viene producido por el rango de las variables con las que trabajamos. Si observamos los valores máximos y mínimos del conjunto de datos de trabajo:

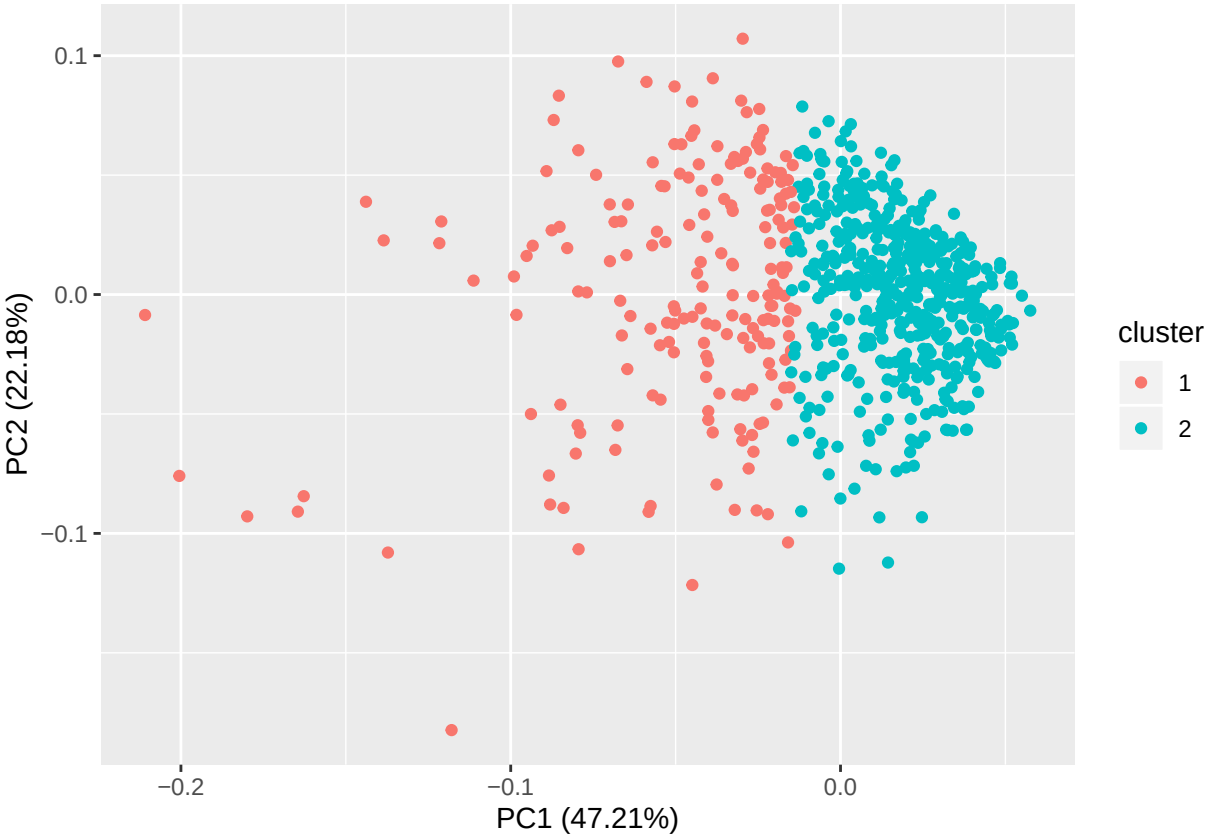
	ingresos	deudaingr	deudacred	deudaotro	empleo	direccion	edad
Min	14	0.4	0.011696	0.045584	0	0	20
Max	446	41.3	20.561310	27.033600	31	34	56

Mientras que la variable `ingresos` tiene un rango de más de 400, otras variables, como `deudacred` apenas tiene un rango de 20 unidades. A la hora de calcular una distancia, la primera variable tendrá una influencia mucho mayor en el resultados que la segunda. Para evitar esta problemática, normalizaremos las variables con las que estamos trabajando, restando a cada elemento su media y dividiendo por la desviación típica de los datos. De esta manera, transformamos la distribución de los datos a otra con media nula y desviación típica unitaria, de forma que todas las variables serán igualmente relevantes a la hora de calcular la distancia entre dos elementos de la base de datos:

```
scaled.data <- scale(numeric.data) # La función scale hace la
↪  transformación
                                   # que hemos especificado
```

Una vez aplicada la transformación, aplicamos el algoritmo de k-medias. Este algoritmo requiere que especifiquemos a priori el número de *clusters* en los que se agruparán los datos. Vamos a tratar de crear dos grupos, para comprobar si con este método se pueden separar los elementos en las dos clases que se nos proporcionan:

```
nclust <- 2

kmeans.res <- kmeans(scaled.data, nclust)

autoplot(kmeans.res, data = scaled.data)
```

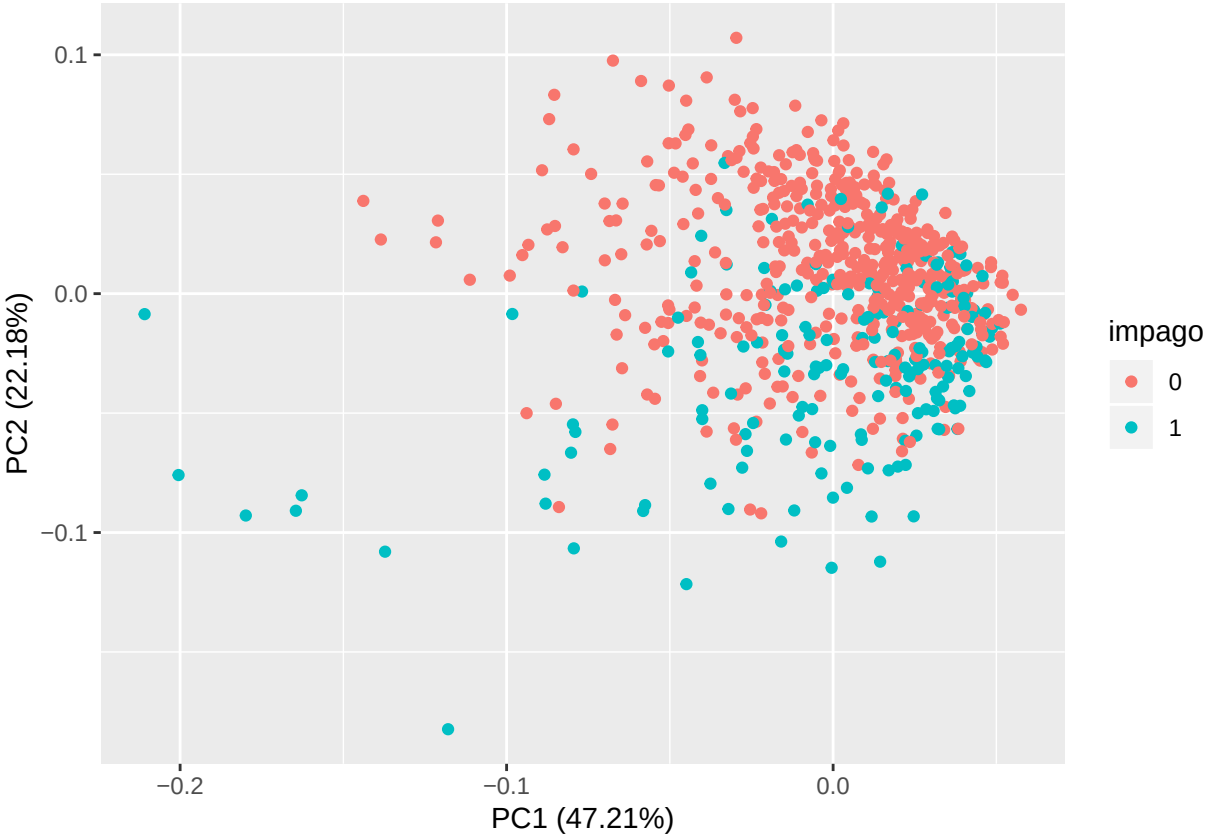


Podemos comprobar como los dos clusters que se han formado están significativamente bien separados. Comprobamos si esta separación se corresponde con los valores de la columna a predecir:

	0	1
0	368	137
1	149	46

Podemos observar que los resultados obtenidos no son especialmente a la hora de clasificar los elementos de nuestro conjunto de datos en pagadores y no pagadores. En efecto, si mostramos el mismo gráfico que anteriormente, pero utilizando la columna impago como etiqueta, obtenemos el

siguiente resultado:



Donde podemos observar que este algoritmo difícilmente será capaz de dividir correctamente los datos del conjunto en las dos clases, debido a que las mismas no forman grupos convexos de elementos. Es posible que, dado que las dos componentes principales de los datos explican sólamente el 70 % de la variabilidad de los mismos, al añadir más componentes las clases sean más fácilmente separables.

Usualmente, cuando se utilizan algoritmos de aprendizaje no supervisado, no se dispone de una clase a predecir, por lo que no se puede construir una matriz de confusión como la que hemos mostrado anteriormente. En ese caso, existen medidas que permiten medir la calidad de los agrupamientos de forma no supervisada. Algunas de estas medidas son:

- Medidas de cohesión: Miden cómo de cercanos están los datos dentro un mismo cluster. Un cluster cuya cohesión es buena suele indicar un buen agrupamiento, porque esta medida indica que sus elementos son similares entre si.
- Medidas de separación: Miden la distancia que existe entre dos clusters distintos. Valores altos en esta medida indican que los clusters generados están distanciados unos de otros, por lo que los elementos que los componen están bien diferenciados.
- Medidas de validez: Usualmente, la métrica que se usa para medir la calidad de un cluster es una combinación de las medidas de separación y cohesión de sus puntos, y la validez total del

agrupamiento es una combinación de las medidas de validez de los grupos que lo componen.

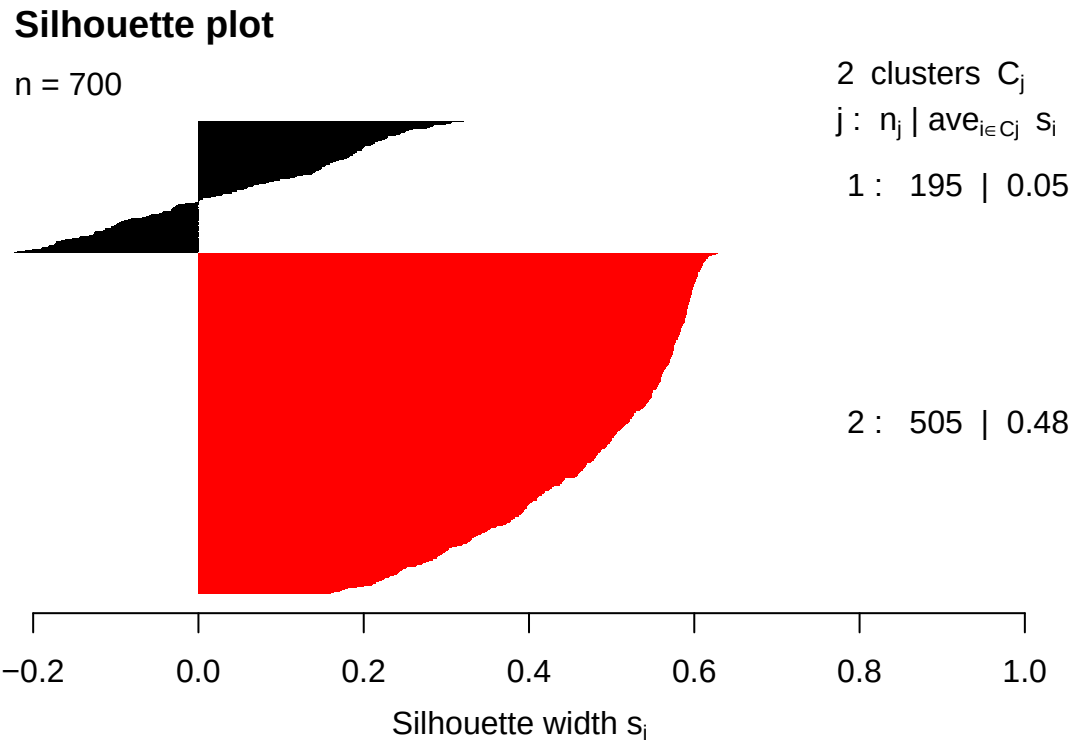
Una medida que suele ser utilizada para determinar la calidad de un agrupamiento es lo que se conoce como el coeficiente de silueta. Este coeficiente se calcula para todos los elementos del conjunto de datos. Para cada punto:

1. Se calcula la distancia media de cada punto a los elementos de su grupo (a_i)
2. Se calcula la distancia media de cada punto a los elementos que no son de su grupo (b_i)
3. El coeficiente de silueta del elemento i se define como

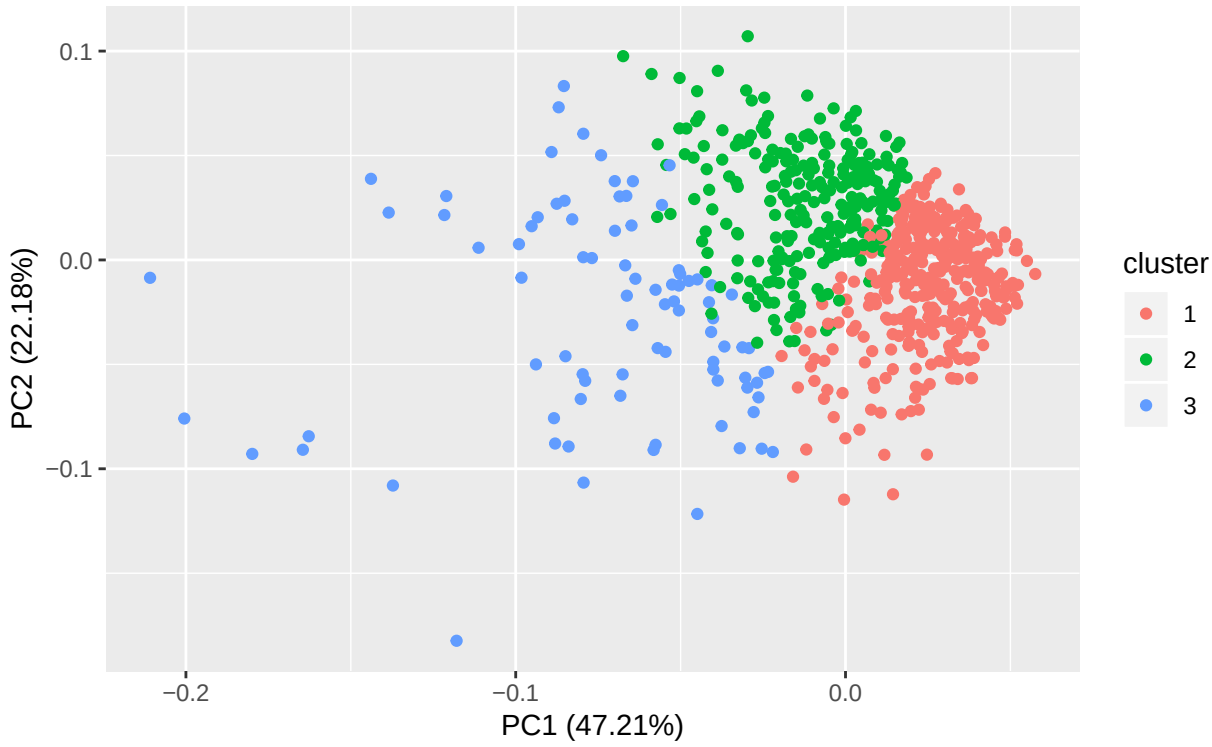
$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Este valor oscila entre -1 y 1. Un valor cercano a 1 es deseable, ya que indica que el punto está bien representado dentro de su cluster (cercano a los elementos de su cluster y alejado de los otros grupos). Un valor negativo, por el contrario, indica que el elemento puede estar mal emplazado, ya que la distancia media del elemento a los elementos de su cluster es mayor que la distancia a los puntos de otros clusters. Una vez calculado el coeficiente de silueta para todos los puntos de un cluster, el coeficiente de silueta del cluster completo se calcula como la media de los coeficientes que componen el grupo, y el coeficiente de silueta del agrupamiento es la media de los coeficientes de todos los grupos. Podemos calcular los coeficientes de silueta del agrupamiento que hemos generado anteriormente de la siguiente forma:

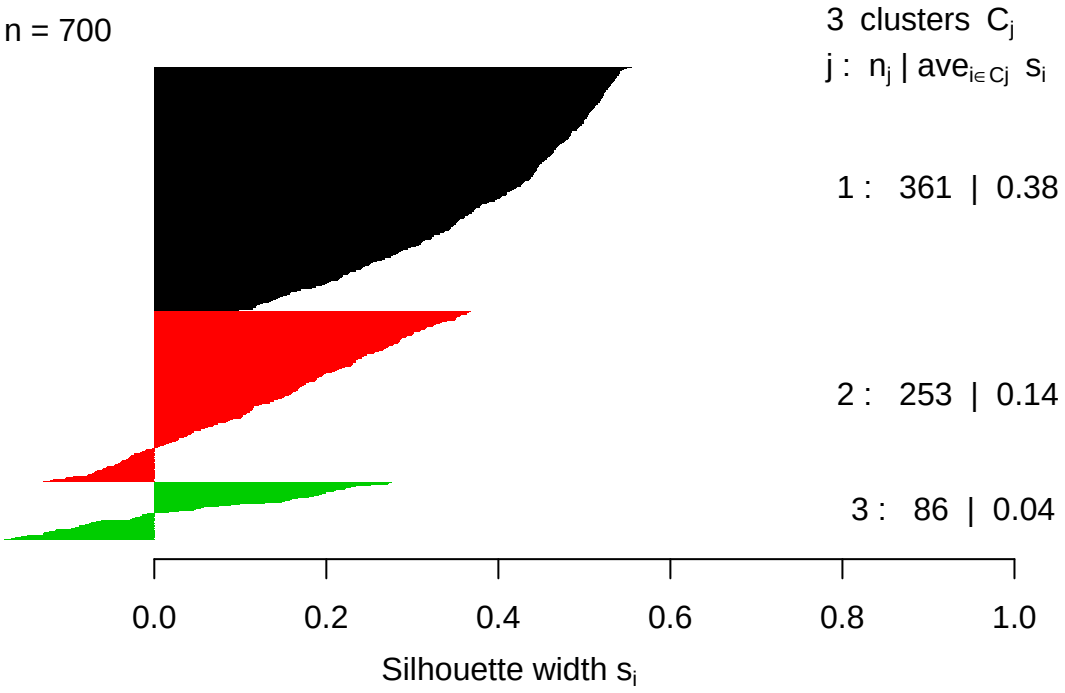
```
scaled.distances <- dist(scaled.data) # Tenemos que calcular
                                      # manualmente las distancias
sil <- silhouette(kmeans.res$cluster, scaled.distances)
plot(sil, col=1:nclust, main = "Silhouette plot")
```



Podemos observar en el gráfico anterior que los resultados no son especialmente buenos. El cluster etiquetado con el valor 2 tiene un coeficiente de silueta relativamente alto, pero el cluster marcado en negro tiene un coeficiente muy cercano a 0, teniendo además una cantidad importante de puntos con un valor negativo. El coeficiente de silueta total del agrupamiento podemos observarlo abajo, y tiene un valor de 0.36. No es un valor especialmente bueno, lo que podría estar indicando que el número de clusters que se ha seleccionado no es el idóneo. Escogimos el valor 2 en un primer momento tratando de obtener una correlación entre los clusters formados y la clase a predecir en este conjunto, pero podría no ser la forma idónea de agrupar los mismos. Repitiendo el proceso anterior cambiando a 3 el número de clusters obtenemos el siguiente resultado:



Silhouette plot



Average silhouette width : 0.25

Observamos que los resultados obtenidos no son especialmente buenos, siendo de peor calidad que

los conseguidos al trabajar con dos clases. Aumentando el número de clusters, el coeficiente de silueta empeora:

	2	3	4	5	6	7	8	9	10
Sil. coef	0.355	0.254	0.208	0.231	0.249	0.219	0.178	0.173	0.202

Como podemos observar en la tabla anterior, el mejor resultado, medido utilizando el coeficiente de silueta, se obtiene con dos clusters, aunque no se produce un resultado especialmente bueno. Esto indica que el método utilizado, posiblemente, no es muy apropiado en este conjunto de datos.

Pasamos a estudiar un algoritmo similar al anterior, pero que en lugar de calcular centroides en el espacio de características, utiliza puntos del propio conjunto como centros de los clusters. Este algoritmo se conoce como el algoritmo de los k-medioides.

3 K-medioides

Pasamos a ver otro algoritmo de agrupamiento, conocido como k-medioides. Este algoritmo tiene un funcionamiento similar al de las k-medias, pero en lugar de trabajar con centroides, los cuales no tienen por qué coincidir con elementos del conjunto de datos, utiliza mediodes o prototipos, que son elementos del propio conjunto de datos. La ventaja de utilizar este algoritmo frente a las k-medias es que no se necesita conocer la posición de los puntos en el espacio de parámetros, si no que es suficiente con conocer la matriz de distancias entre los mismos, ya que no se necesita construir instancias artificiales como son los centroides.

Existen diversos algoritmos que computan soluciones por el método de los k-mediodes. El más típico es una búsqueda voraz, que no encuentra necesariamente el óptimo, pero aporta buenas soluciones. Su funcionamiento es parecido al que describimos para el método de las k-medias:

1. Se inicializan aleatoriamente los k medioides
2. Se asocia cada punto al cluster definido por el mediode más cercano
3. Mientras el coste final de la configuración decreza:

- Para cada medioide m_i y cada punto no medioide o

1. Se intercambian los papeles de dichos puntos y se recalcula el coste
2. Si el coste es mejor que el resultado actual, se guarda el intercambio

- Se realiza el mejor intercambio encontrado. Si no se ha encontrado un intercambio mejor que el resultado actual, el algoritmo termina

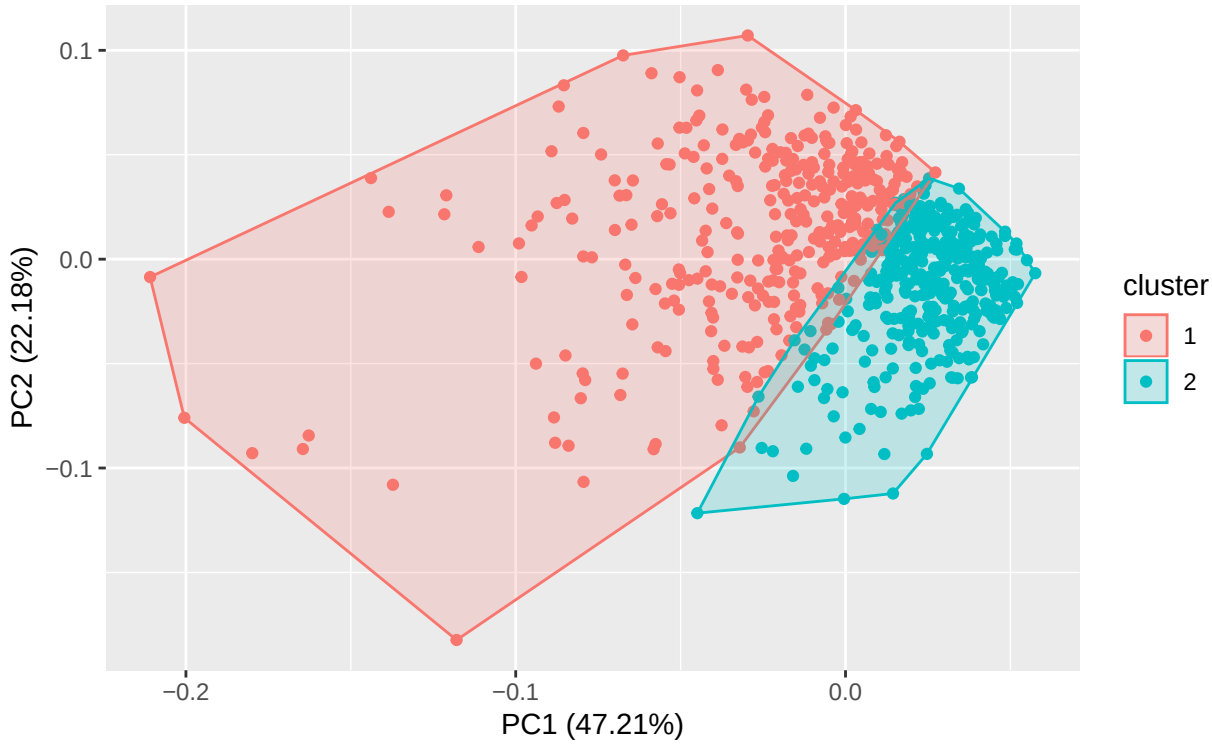
El coste de la configuración se define como la suma de las distancias de cada punto al medioide de su *cluster*, de la misma manera que ocurre en el algoritmo de las k-medias.

A continuación vamos a ejecutar el algoritmo sobre nuestro conjunto de datos.

3.1 Aplicación del algoritmo

Al igual que hemos hecho anteriormente, comenzaremos aplicando el algoritmo sobre los datos numéricos de los que dispone el problema:

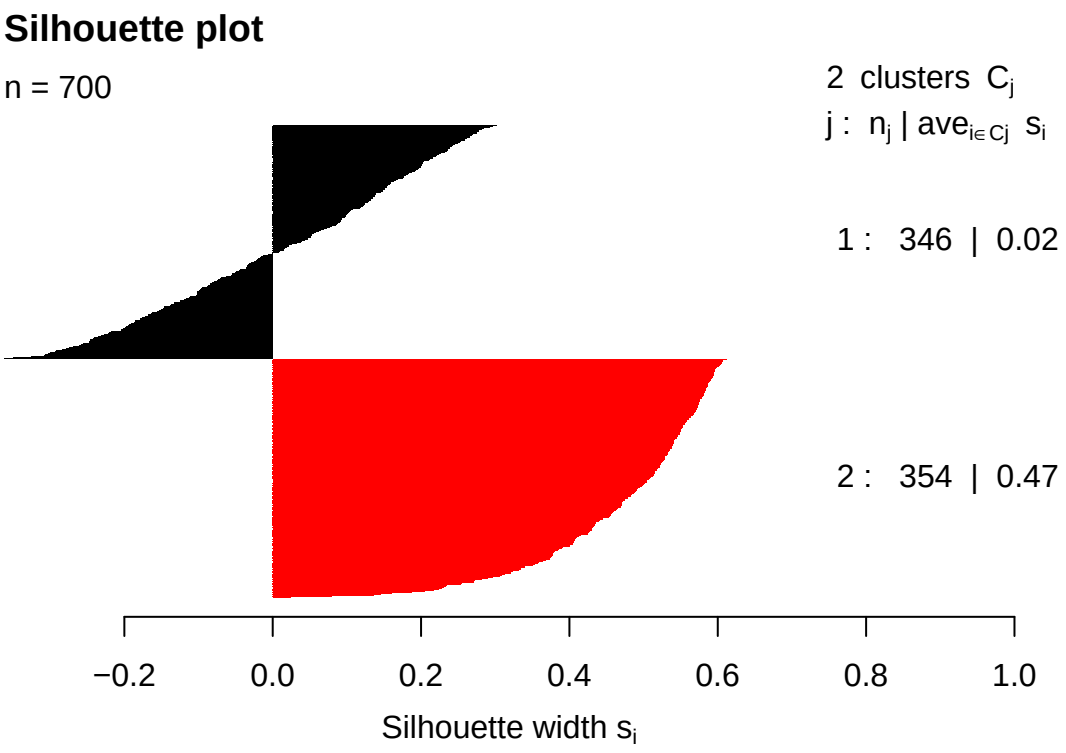
```
nclust <- 2
dataset <- read.csv(dataset.path, sep=";",  dec=",")
dataset <- dataset[1:700,]

numeric.data <- dataset %>% select(numeric.vars)
scaled.data <- scale(numeric.data)

set.seed(0)
pam.result <- pam(scaled.data, nclust)

autoplot(pam.result, frame=T)
```



Podemos ver como, a pesar de ser un algoritmo con un funcionamiento similar al algoritmo de las k-medias, el hecho de forzar que los centros de los clusters sean puntos del conjunto de datos produce unos resultados significativamente distintos. Probablemente, la existencia de puntos extremos (se pueden ver en el gráfico en la parte izquierda, más o menos aislados), hacen que el centroide del cluster 1 en las k-medias se desplace de la zona donde se concentran la mayoría de los puntos. Este comportamiento en el algoritmo de los k-medioides no puede darse.

Pasamos a ver si los agrupamientos encontrados por el algoritmo son de buena calidad:

```
sil <- silhouette(pam.result$cluster, scaled.distances)
plot(sil, col=1:nclust, main = "Silhouette plot")
```



Podemos observar ciertas diferencias entre el resultado de este modelo y el que tuvimos a dos clusters con las k-medias. En primer lugar, obtenemos clusters más compensados, con un número de puntos similar. Con el algoritmo anterior, el número de puntos en el primer cluster era significativamente menor que en el segundo. Aquí la diferencia es mucho menor. Al igual que en el caso anterior, no obstante, tenemos muchos puntos del primer cluster que tienen un coeficiente de silueta negativo. Estos puntos son los puntos cercanos a la frontera entre un grupo y el otro. La distancia de dichos puntos al otro cluster es relativamente pequeña, mientras que la distancia a los puntos extremos de su propio cluster es significativamente alta. Probablemente, el problema no resida en el cluster en

el que están colocados los puntos cercanos a la frontera, si no más bien el hecho de que los puntos alejados, en realidad, no sean agrupables en un cluster, si no más bien puntos ruidosos. El problema de los dos algoritmos que hemos visto hasta el momento es que no son capaces de marcar elementos del conjunto como ruido. Veremos más adelante otros algoritmos que nos permitirán marcar puntos como ruido, y no incluirlos dentro de ningún grupo.

Pasamos a ver cómo podemos incorporar información de las variables nominales a este modelo, cosa que no podíamos hacer en el caso de las k-medias.

3.2 Combinación de distancias

En el algoritmo de las k-medias no pudimos aprovechar la información contenida en las variables nominales debido a que necesitábamos poder generar instancias artificiales en el espacio de características. Esto hacía que tuviésemos que proporcionar el conjunto de datos completo al algoritmo. En el caso de los k-medioides, será suficiente con calcular a priori la matriz de distancias, y esto nos permite combinar la distancia de las variables numéricas y las nominales.

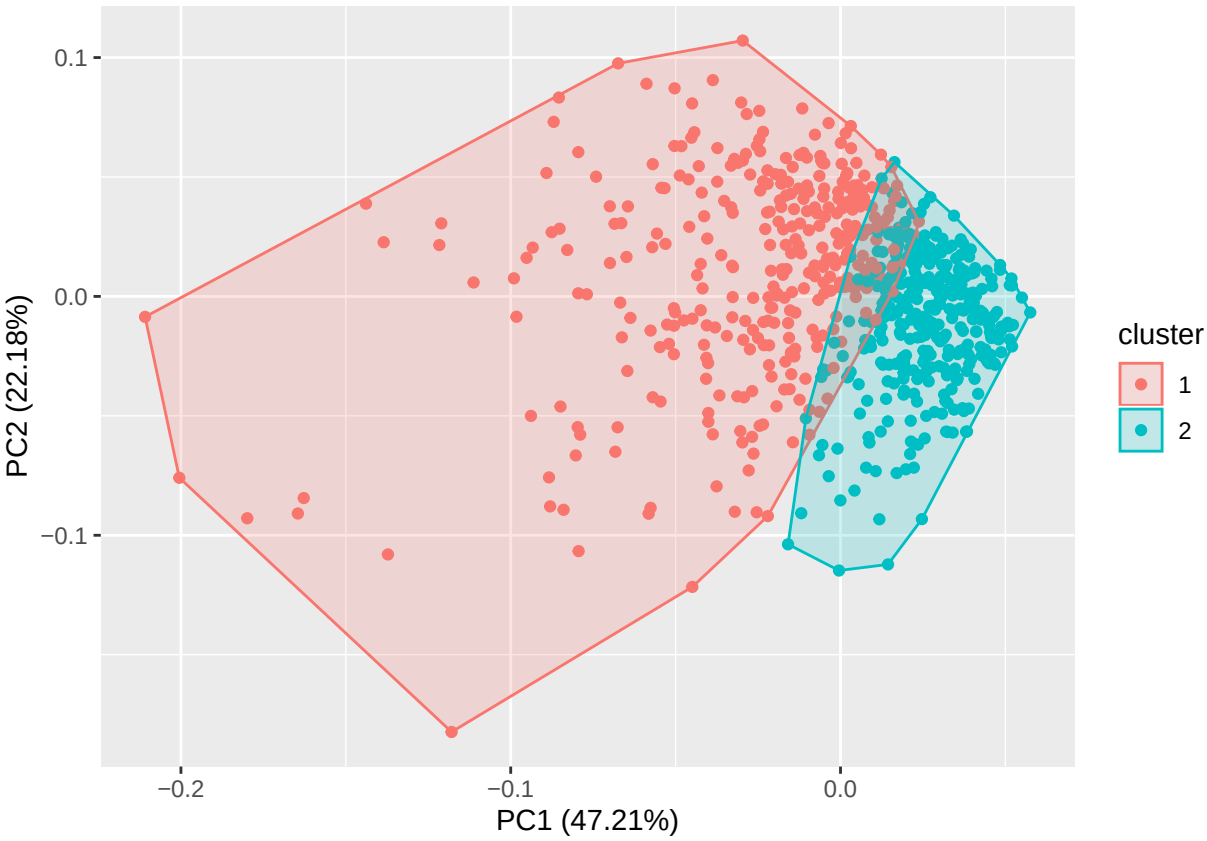
Se puede argumentar que el algoritmo de las k-medias se puede modificar para que utilice una función de distancia propia, que calcule adecuadamente la distancia entre dos vectores que contienen simultáneamente variables numéricas y nominales. No obstante, aparece otro problema con este enfoque. Para las variables numéricas, el algoritmo calcula los centroides haciendo la media de los valores de cada variable de los puntos que pertenecen a cada cluster. En el caso de las variables nominales, esta media carecería de sentido, dado que los valores que tomaría esta variable pasarían a ser continuos, perdiéndose el sentido categórico de las mismas.

Existen extensiones del algoritmo de las k-medias, que utilizan trucos similares al *kernel trick* utilizado en las máquinas de vectores de soporte (SVMs), que permiten obtener soluciones aproximadas al problema de las k-medias sin necesidad de calcular explícitamente los centroides. No obstante, estos métodos quedan fuera del ámbito de conocimiento de la asignatura y no los estudiaremos.

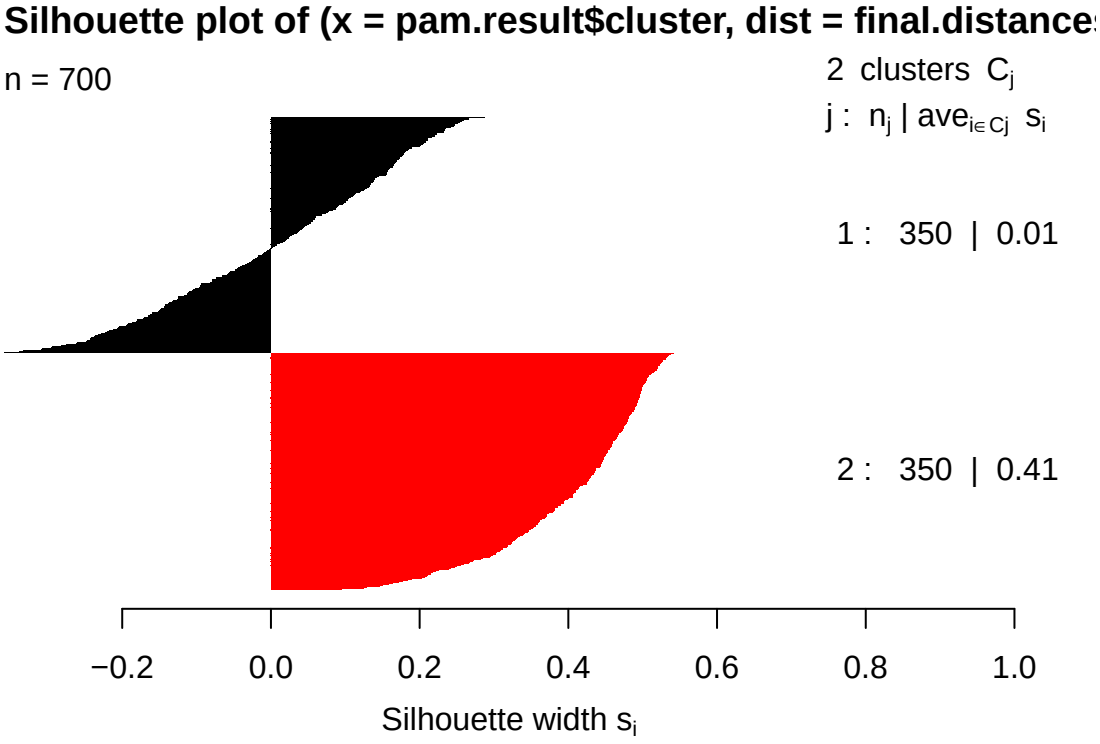
Para el problema que nos ocupa, la combinación de distancias que vamos a llevar a cabo es la siguiente. Para las variables numéricas, seguiremos calculando la distancia euclídea sobre las variables normalizadas. Sobre la variable nominal que tenemos en el problema, la cual mide el nivel educativo del individuo, crearemos tantas variables binarias como posibles valores toma dicha variable. Sobre esa binarización de las variables, se calculará una distancia binaria, la cual se mide como la proporción de columnas que difieren entre ambos ejemplos.

Una vez tengamos calculadas las dos matrices de distancia, la distancia final que calcularemos será la media entre ambas matrices de distancia. Se muestra el funcionamiento del algoritmo:

```
dataset <- read.csv(dataset.path, sep=";", dec = ",")
dataset <- dataset[1:700,]

numeric.vars <- c(
    'ingresos', 'deudaingr', 'deudacred',
    'deudaotro', 'empleo',  'direccion', 'edad'
)
nominal.vars <- c('educ')

numeric.data <- dataset %>% select(numeric.vars)
nominal.data <- dataset %>% select(nominal.vars)

dummy.data <- dummy_cols(nominal.data)[2:6]

scaled.data <- scale(numeric.data)

scaled.distances <- dist(scaled.data)
dummy.distances <- dist(dummy.data, method="binary")
final.distances <- (scaled.distances + dummy.distances) / 2

pam.result <- pam(final.distances, nclust)

pam.result$data <- as.data.frame(scaled.data) # Necesitaremos los datos
                                              # para la representación gráfica
autoplot(pam.result, frame=T)
```



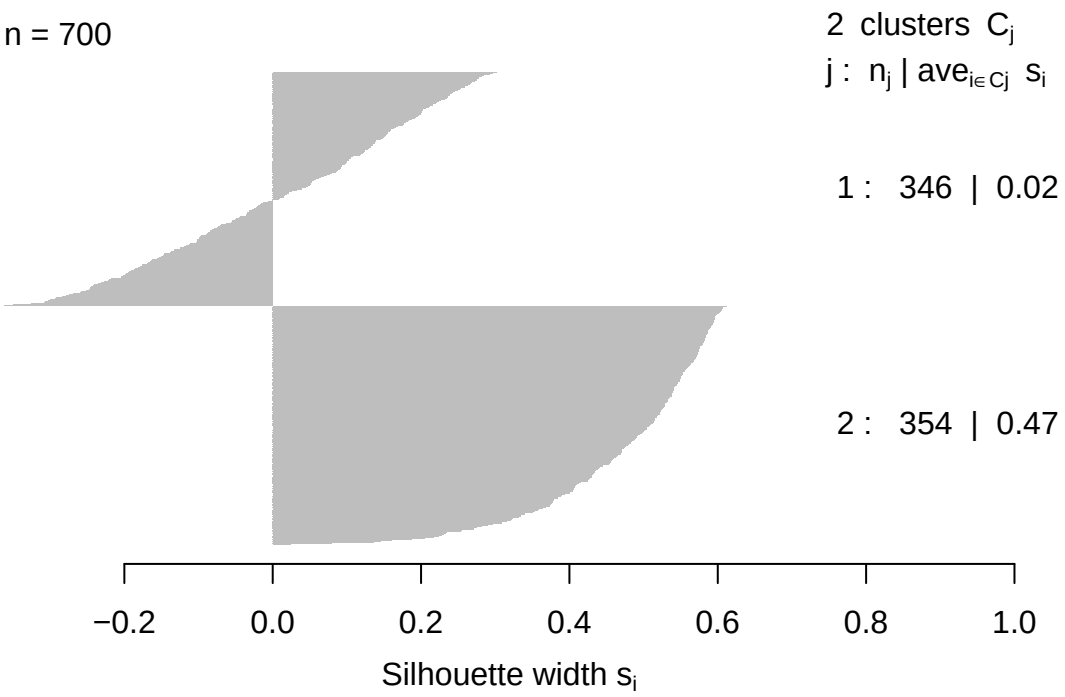

```
sil <- silhouette(pam.result$cluster, final.distances)
plot(sil, col=1:nclust)
```



En primer lugar, podemos apreciar que el resultado del algoritmo es significativamente distinto,

3.3 Búsqueda del valor óptimo de k

```
dataset <- read.csv(dataset.path, sep=";", dec = ",")
dataset <- dataset[1:700,]

numeric.data <- dataset %>% select(numeric.vars)
nominal.data <- dataset %>% select(nominal.vars)

scaled.data <- scale(numeric.data)

scaled.distances <- dist(scaled.data)
nominal.distances <- dist(nominal.data, method="binary")

final.distances <- (scaled.distances + nominal.distances) / 2

pamk.result <- pamk(scaled.distances)

plot(pamk.result$pamobject)
```




```
cluster.stats(final.distances, pamk.result$pamobject$clustering)
```

```
## $n
## [1] 700
##
## $cluster.number
## [1] 2
##
## $cluster.size
## [1] 346 354
##
## $min.cluster.size
## [1] 346
##
## $noisen
## [1] 0
```

```
##
## $diameter
## [1] 7.306150 3.409958
##
## $average.distance
## [1] 1.7947219 0.9933108
##
## $median.distance
## [1] 1.6047014 0.9153235
##
## $separation
## [1] 0.1729315 0.1729315
##
## $average.toother
## [1] 1.898308 1.898308
##
## $separation.matrix
##           [,1]      [,2]
## [1,] 0.0000000 0.1729315
## [2,] 0.1729315 0.0000000
##
## $ave.between.matrix
##          [,1]     [,2]
## [1,] 0.000000 1.898308
## [2,] 1.898308 0.000000
##
## $average.between
## [1] 1.898308
##
## $average.within
## [1] 1.389437
##
## $n.between
## [1] 122484
##
## $n.within
## [1] 122166
##
```

```
## $max.diameter
## [1] 7.30615
##
## $min.separation
## [1] 0.1729315
##
## $within.cluster.ss
## [1] 917.2731
##
## $clus.avg.silwidths
##          1          2
## 0.01758306 0.47407793
##
## $avg.silwidth
## [1] 0.248439
##
## $g2
## NULL
##
## $g3
## NULL
##
## $pearsongamma
## [1] 0.2862956
##
## $dunn
## [1] 0.0236693
##
## $dunn2
## [1] 1.057717
##
## $entropy
## [1] 0.6930819
##
## $wb.ratio
## [1] 0.7319345
##
## $ch
```

```
## [1] 232.8335
##
## $cwidegap
## [1] 3.314062 1.515009
##
## $widestgap
## [1] 3.314062
##
## $sindex
## [1] 0.3086691
##
## $corrected.rand
## NULL
##
## $vi
## NULL
```

4 DBSCAN

El algoritmo de agrupamiento DBSCAN es un algoritmo de agrupamiento por distancias que trata de formar grupos de puntos por densidad. El funcionamiento del algoritmo es el siguiente. Mientras no se hayan agrupado todos los puntos:

1. Se selecciona un punto que aún no ha sido agrupado x
2. Se seleccionan todos los elementos del conjunto que estén a una distancia menor que ε de x y se añaden al cluster de x, siempre que no hayan sido seleccionados previamente
3. Para todos los puntos seleccionados, se repite el proceso
4. Cuando no queden más puntos por seleccionar dentro del cluster
5. Si se han agrupado suficientes puntos juntos (un parámetro del algoritmo dicta cuántos puntos son necesarios), se marcan todos los puntos agrupados como un cluster.
6. Si no se han agrupado suficientes puntos, se marcan como aislados
7. Si quedan puntos sin marcar como pertenecientes a un cluster o como puntos aislados, se vuelve al inicio

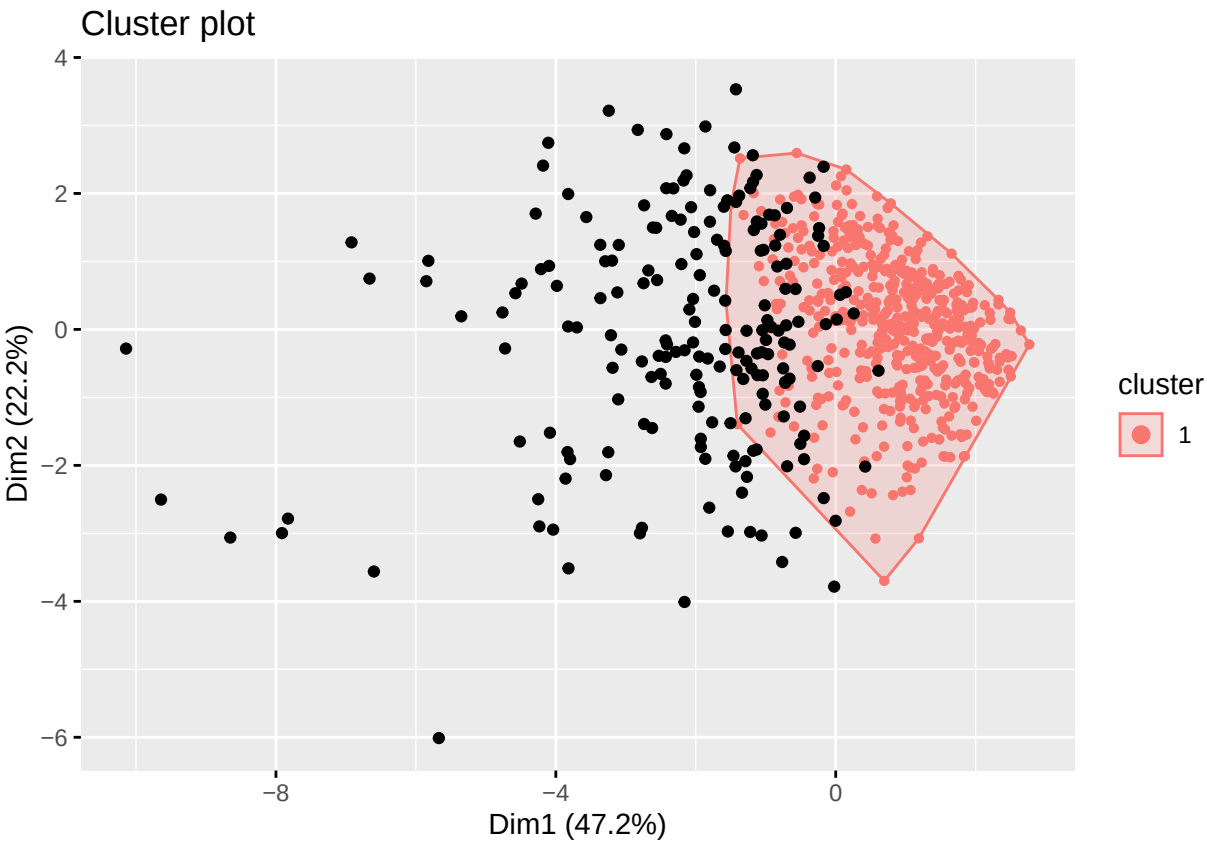
Una ventaja que tiene este algoritmo respecto a las k-medias que comentábamos anteriormente es su capacidad para inferir el número de grupos, que no ha de especificarse a priori. Además, es capaz de identificar datos ruidosos, que se encuentran alejados de los clusters.

Por otro lado, como podemos observar en la descripción del algoritmo, hay que ajustar dos parámetros. Por un lado, tenemos el valor ε, que nos marca la distancia máxima a la que pueden encontrarse los puntos para ser agrupados, y por otro el número mínimo de puntos que deben ser relacionados para que un agrupamiento pase a considerarse un cluster, en lugar de un grupo pequeño de outliers.

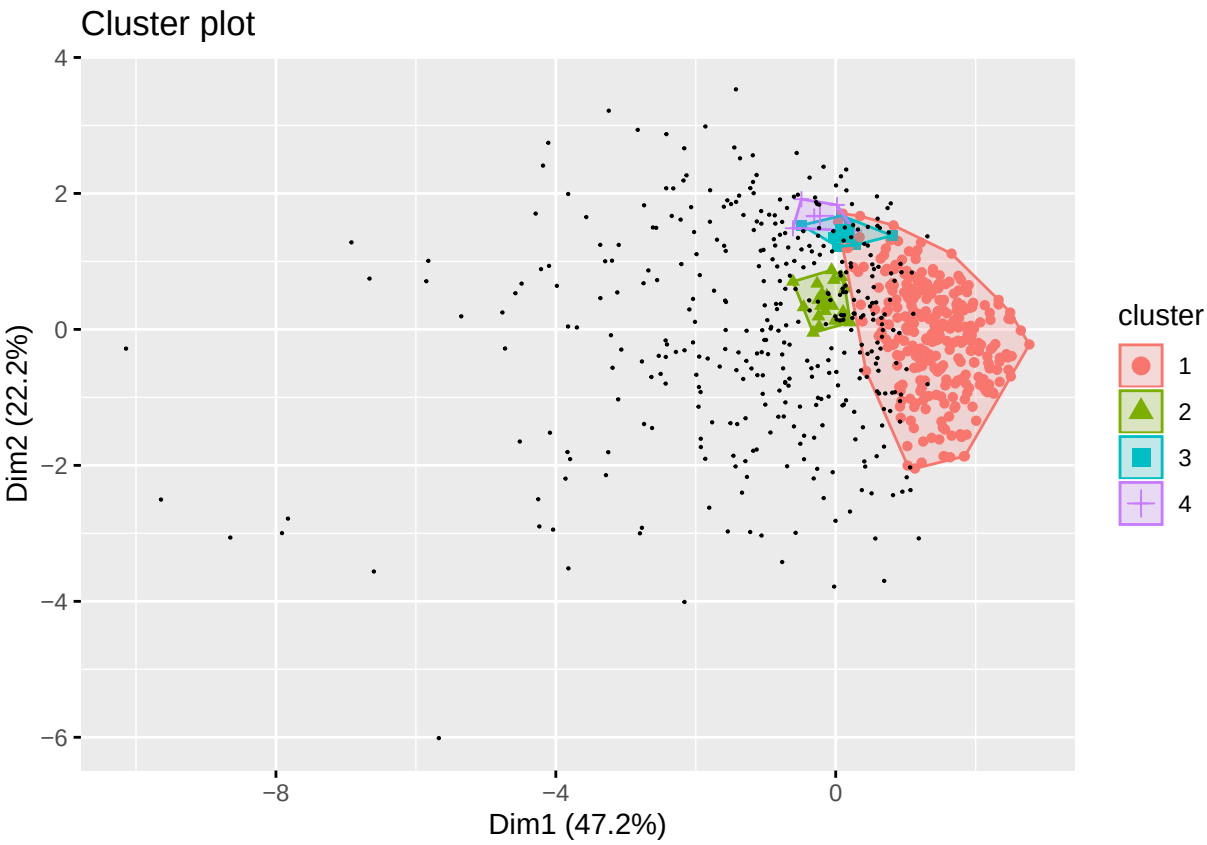
4.1 Aplicación del algoritmo

Pasamos a hacer pruebas con este algoritmo con el conjunto de datos con el que estamos trabajando. Empezaremos utilizando sólamente las variables numéricas y más adelante incorporaremos las variables nominales. El parámetro del mínimo de vecinos lo dejaremos en su valor por defecto, $MinPts = 5$. El valor ε, como podremos observar, tiene una gran influencia en el resultado final. Como no conocemos una política específica para la elección de este valor, comenzaremos tomando $\varepsilon = 1$:

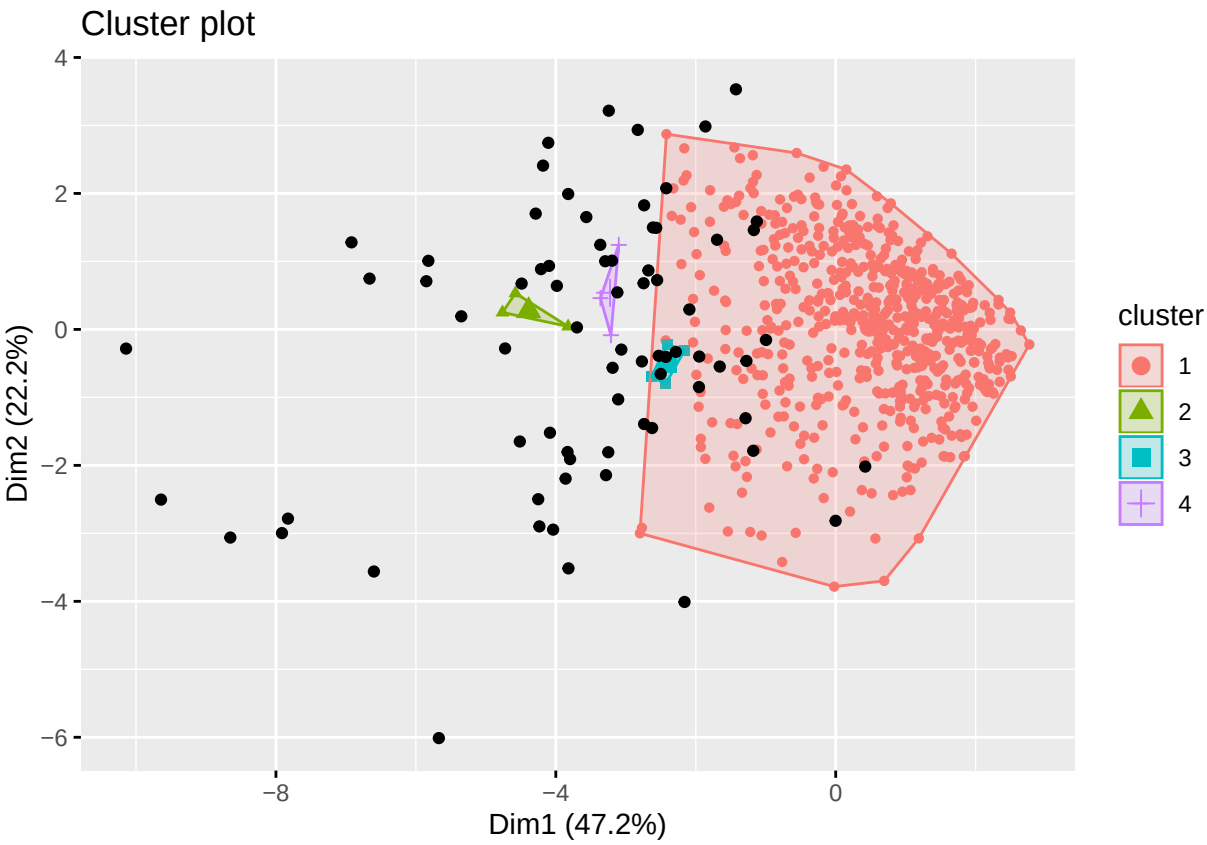
```
numeric.data <- dataset %>% select(numeric.vars)
scaled.data <- scale(numeric.data)

## Aplicamos DBSCAN (con el parámetro dist este método acepta una matriz
## de distancias en lugar de los datos)
dbscan.res <- dbscan(scaled.data, eps=1)

fviz_cluster(dbscan.res, data = scaled.data, geom = "point")
```



Podemos observar que sólo se han etiquetado los puntos en dos conjuntos, el cluster 1, y los puntos marcados como outliers (se les asigna el cluster 0 a todos ellos). Esto nos hace pensar que los valores que hemos tomado en los parámetros no son los adecuados. En particular, el parámetro que tiene una mayor afectación en el resultado del algoritmo es el ε escogido. Trataremos de bajar dicho valor ligeramente, para forzar a que los puntos que han sido introducidos en el *cluster* 1 se dividan en más grupos. Si tomamos un $\varepsilon = 0.7$, obtenemos:



Se ha reducido el tamaño de los outliers para permitir la visualización de los nuevos *clusters* que se han formado. En el gráfico podemos observar que el número de grupos ha pasado de 1 a 4, pero también ha aumentado significativamente el número de outliers. Además, tenemos un grupo de gran tamaño, y tres relativamente pequeños. Esto nos indica que en nuestro conjunto tenemos una zona en la que existe una gran densidad de ejemplos, mientras que en el resto del espacio los puntos se encuentran mucho más repartidos. Esta disposición de los datos es especialmente perjudicial para los resultados que ofrece el algoritmo. Dado que el valor ε que se utiliza es global, una densidad de puntos distinta en dos zonas del espacio de características puede producir que ciertos agrupamientos sean etiquetados como datos outliers, si la densidad de una de las zonas es demasiado baja. Podemos controlar en parte esta problemática reduciendo el número de puntos necesarios para formar un cluster y ampliando la distancia. Por ejemplo, si tomamos $\varepsilon = 1.3$, y reducimos a 3 el número de vecinos, el resultado que obtenemos es el siguiente:

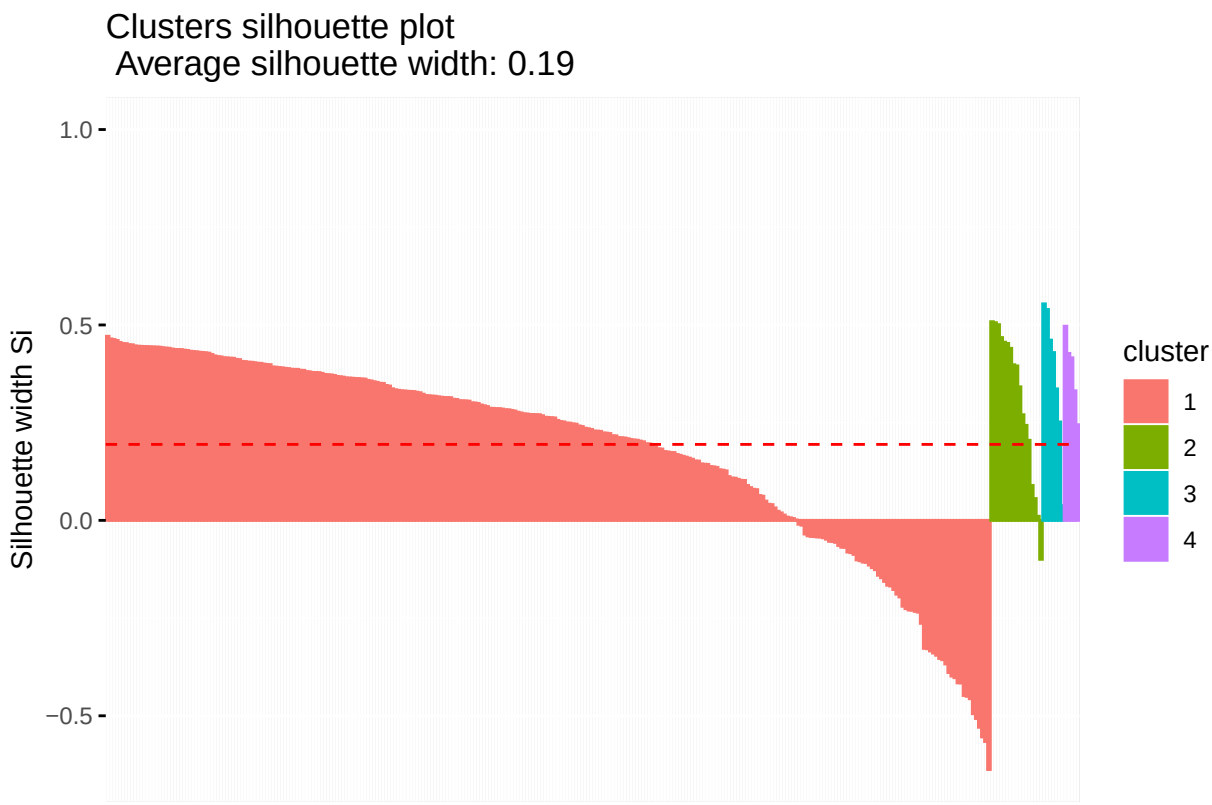


De nuevo, tenemos los datos distribuidos en cuatro clusters, y el número de outliers se ha reducido significativamente, pero a cambio el cluster 1 ha crecido mucho en tamaño, mientras que los otros tres clusters están formados por un número muy pequeño de puntos. Vamos a estudiar ahora la calidad de los agrupamientos que se han formado. No tiene sentido que estudiemos la calidad de los grupos del primer agrupamiento porque sólo se ha generado un grupo, el resto de puntos no están agrupados entre sí. Estudiaremos por tanto los otros dos agrupamientos.

```
dbscan.res.1 <- dbscan(scaled.data, eps=.7)
dbscan.res.2 <- dbscan(scaled.data, eps=1.3, MinPts=3)

idx.1 = (dbscan.res.1$cluster != 0)
idx.2 = (dbscan.res.2$cluster != 0)

## Eliminamos del conjunto aquellas filas que han sido etiquetadas
## como outliers antes de calcular el coeficiente de silueta
clustered.data.1 <- scaled.data[idx.1,]
clustered.data.2 <- scaled.data[idx.2,]

dist1 <- dist(clustered.data.1)
```

```
dist2 <- dist(clustered.data.2)

sil1 <- silhouette(dbscan.res.1$cluster[idx.1], dist1)
sil2 <- silhouette(dbscan.res.2$cluster[idx.2], dist2)

fviz_silhouette(sil1)
```

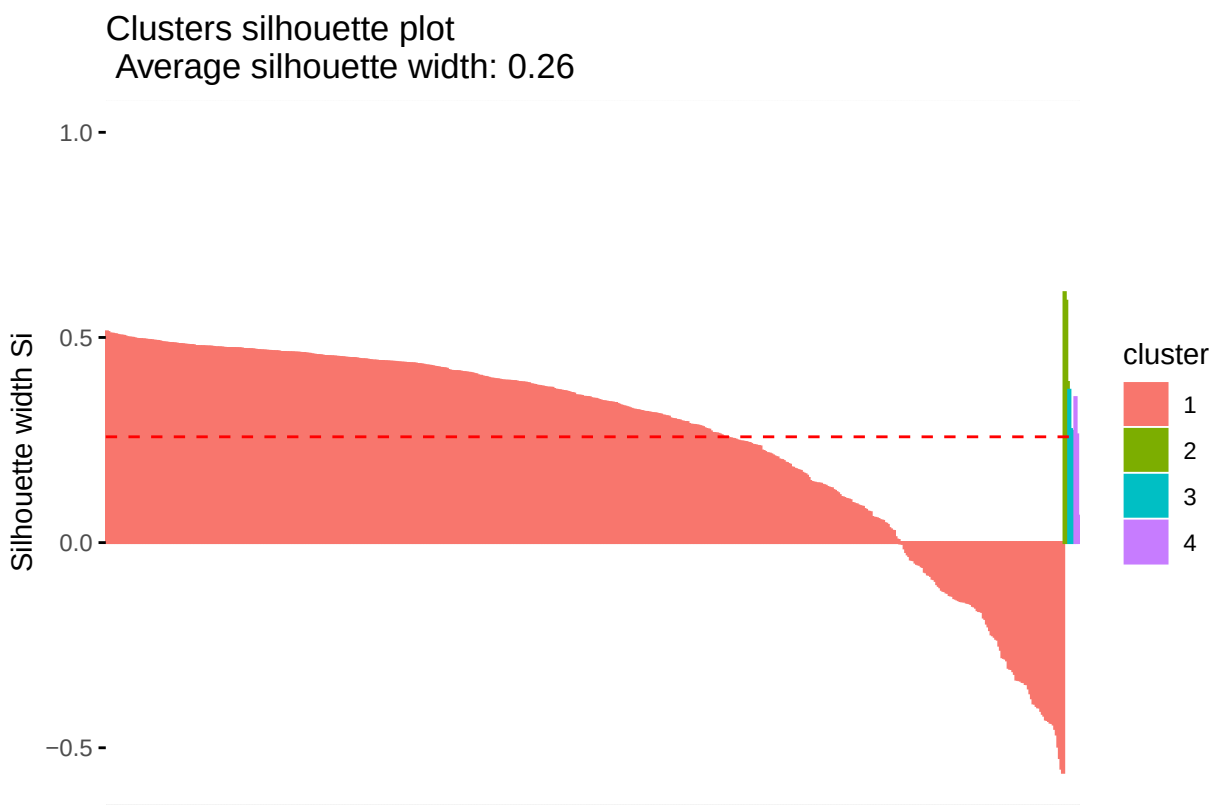
```
##   cluster size ave.sil.width
## 1       1  289          0.18
## 2       2   17          0.31
## 3       3    7          0.37
## 4       4    5          0.38
```




```
fviz_silhouette(sil2)
```

```
##   cluster size ave.sil.width
## 1       1  617          0.26
## 2       2    3          0.53
```

```
## 3       3     4          0.26
## 4       4     3          0.23
```



Antes de cada gráfico se muestra el tamaño de cada uno de los grupos y el valor de su coeficiente de silueta. Como hemos comentado anteriormente, tenemos un desbalanceo importante en los grupos que se forman. Un grupo contiene a casi todos los elementos, mientras que los otros tres son de reducido tamaño. Esto es especialmente notorio en el segundo caso, donde el primero *cluster* tiene más de 600 elementos, y los otros grupos menos de 5 en todos los casos.

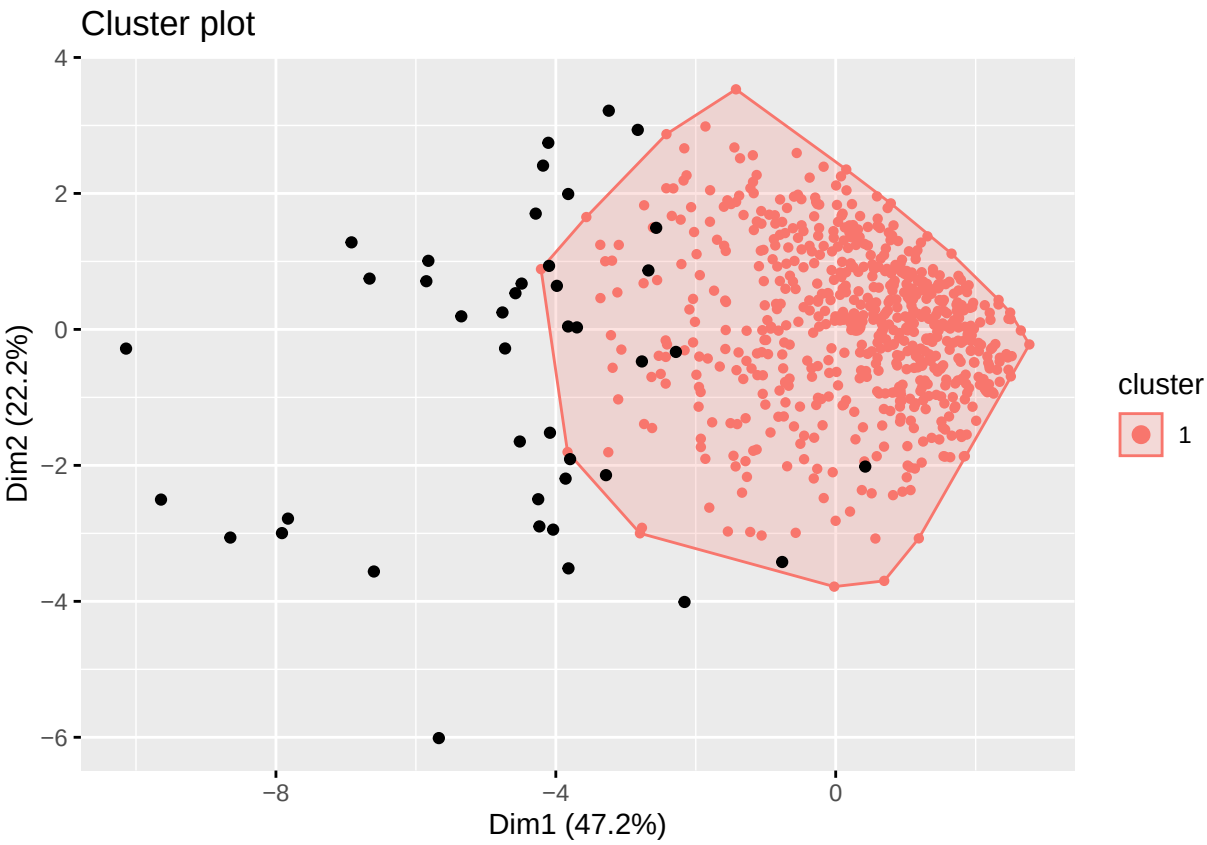
Otra conclusión que se puede extraer de las gráficas es que muchos de los puntos que están incluidos en el primer grupo no pertenecen a este grupo de forma natural, teniendo un coeficiente de silueta negativo. Esta problemática viene dada por el hecho de tener un número de puntos muy grande en la zona densa. Con una probabilidad muy alta, cuando se seleccione el primer punto para crear el primer cluster, este punto va a ser uno de los que se encuentra en la zona densa. Cuando empiece a construirse este grupo, una gran cantidad de puntos irán añadiéndose por proximidad. De esta forma, cuando termina de formarse el primer grupo, los grupos siguientes dispondrán de pocos vecinos (debido a que ya habrían sido agrupados), y los puntos que no hayan sido agrupados hasta ese momento estarán muy distantes. Es por esto por lo que se forma un primer grupo masivo, en el que muchos de sus puntos no están bien representados, y después pequeños grupos de pocos puntos.

Pasamos a ver cómo podemos incorporar la información de las variables nominales al resultado de

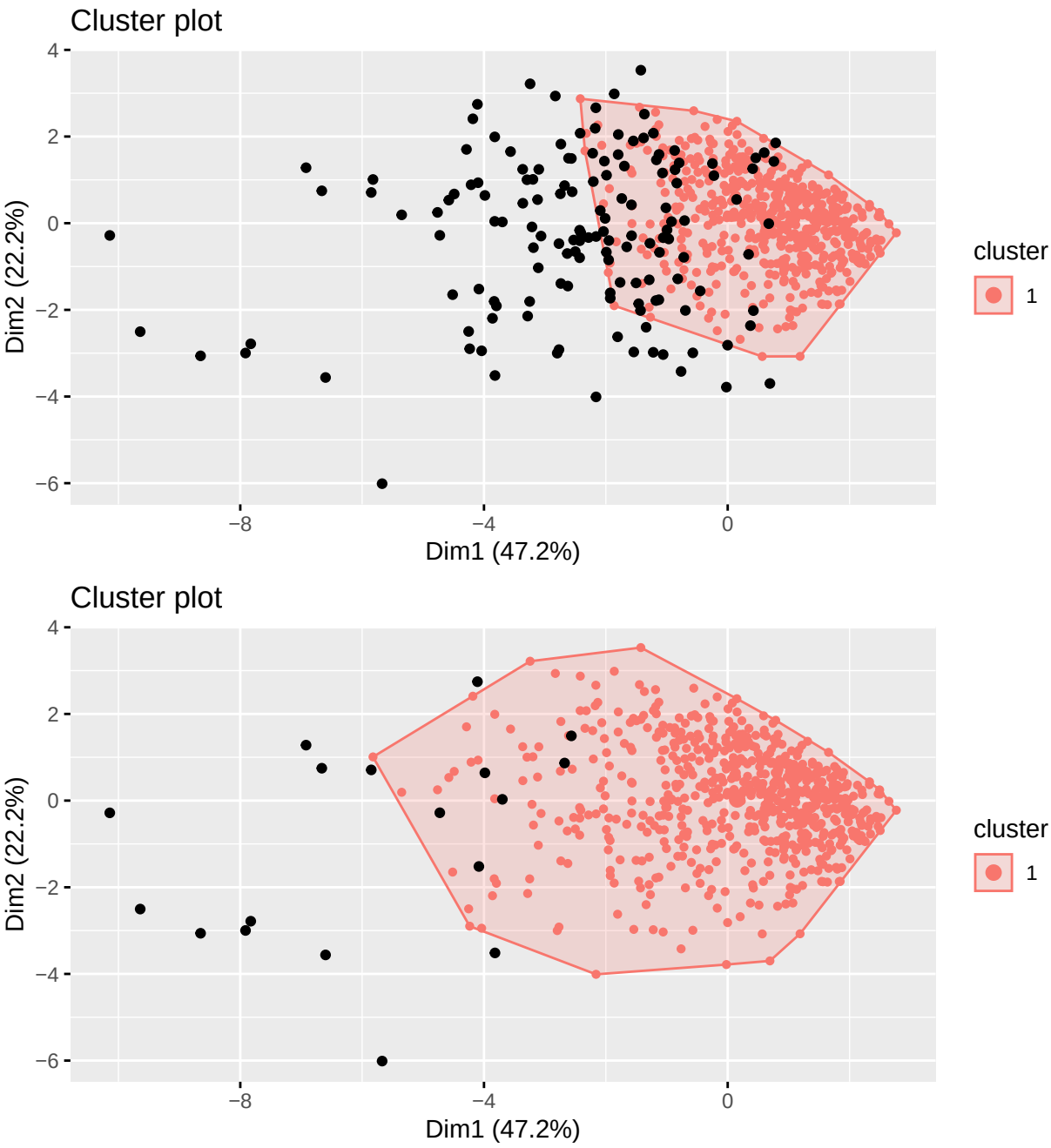
este algoritmo:

4.2 Combinación de distancias

```
numeric.vars <- c(
    'ingresos', 'deudaingr', 'deudacred',
    'deudaotro', 'empleo',  'direccion', 'edad'
)

dataset <- read.csv(dataset.path, sep = ";", dec=",")[1:700,]

numeric.data <- dataset %>% select(numeric.vars)
nominal.data <- dataset %>% select(nominal.vars)

scaled.data <- scale(numeric.data)
dummy.data <- dummy_cols(nominal.data)[2:6]

scaled.dist <- dist(scaled.data)
dummy.dist <- dist(dummy.data, method = "binary")

## Tomamos la media aritmética de las distancias como la distancia final
final.dist <- (scaled.dist + dummy.dist) / 2

dbscan.res <- dbscan(final.dist, eps=1, method = "dist")

fviz_cluster(dbscan.res, data = scaled.data, geom = "point")
```



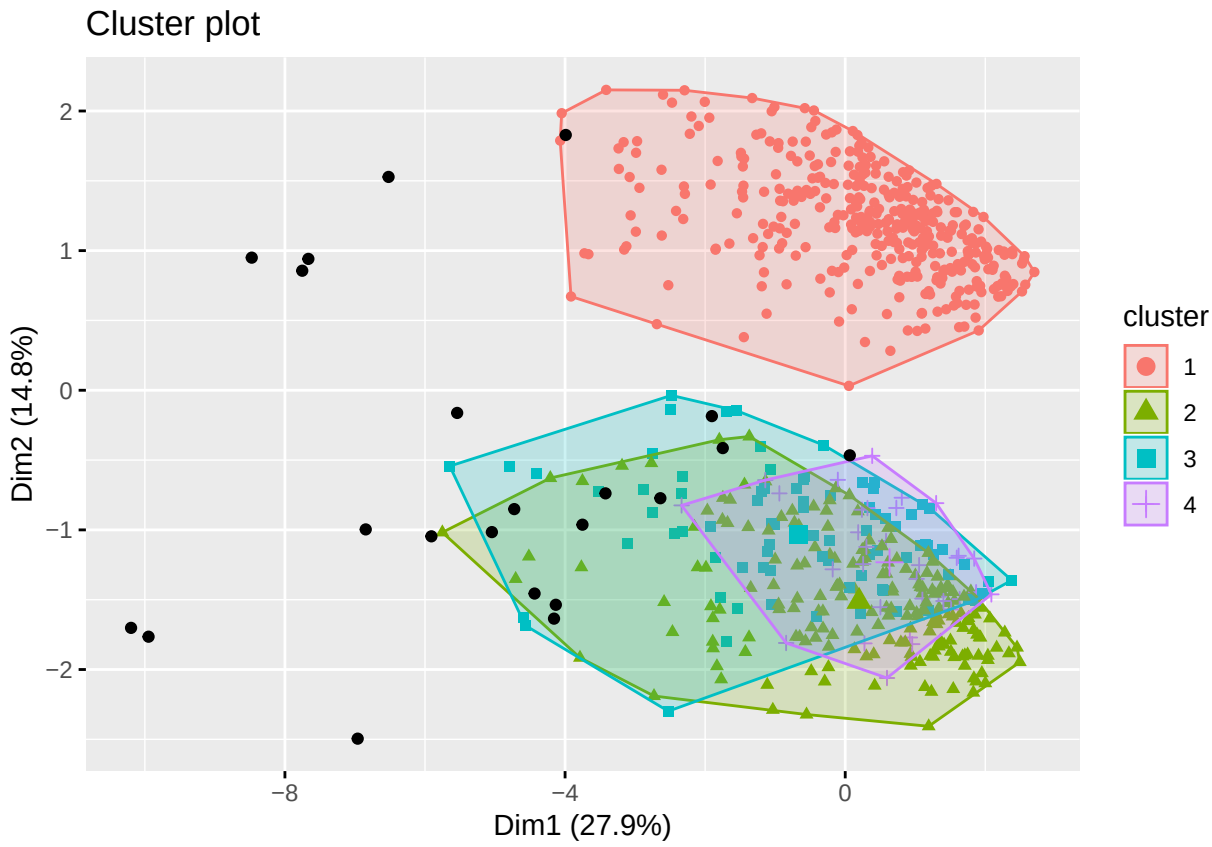
Hemos utilizado la misma configuración del algoritmo que utilizamos en primer lugar. Podemos observar que, en este caso, muchos más puntos han pasado a formar parte del primer cluster. Esto indica que las distancias entre los puntos se han reducido. Repetimos también los cambios de configuración, para ver cómo ha cambiado el resultado:



Observamos cómo la modificación en las distancias nos ha producido que las configuraciones anteriores no nos sean útiles ahora para generar los agrupamientos. Esto puede haber sido provocado por una mala combinación de las distancias. Hemos escogido la media aritmética como la forma de combinar las distancias, pero es posible que no sea la combinación óptima. En particular, si tenemos distinto número de variables continuas y nominales, el dar el mismo peso a ambas matrices puede introducir sesgos no deseados. Otra posibilidad es que por conocimiento previo de nuestro problema, nos interese dar más relevancia a unas variables o a otras. En nuestro caso, si damos más importancia

a las variables nominales que a las numéricas, obtenemos el siguiente resultado:

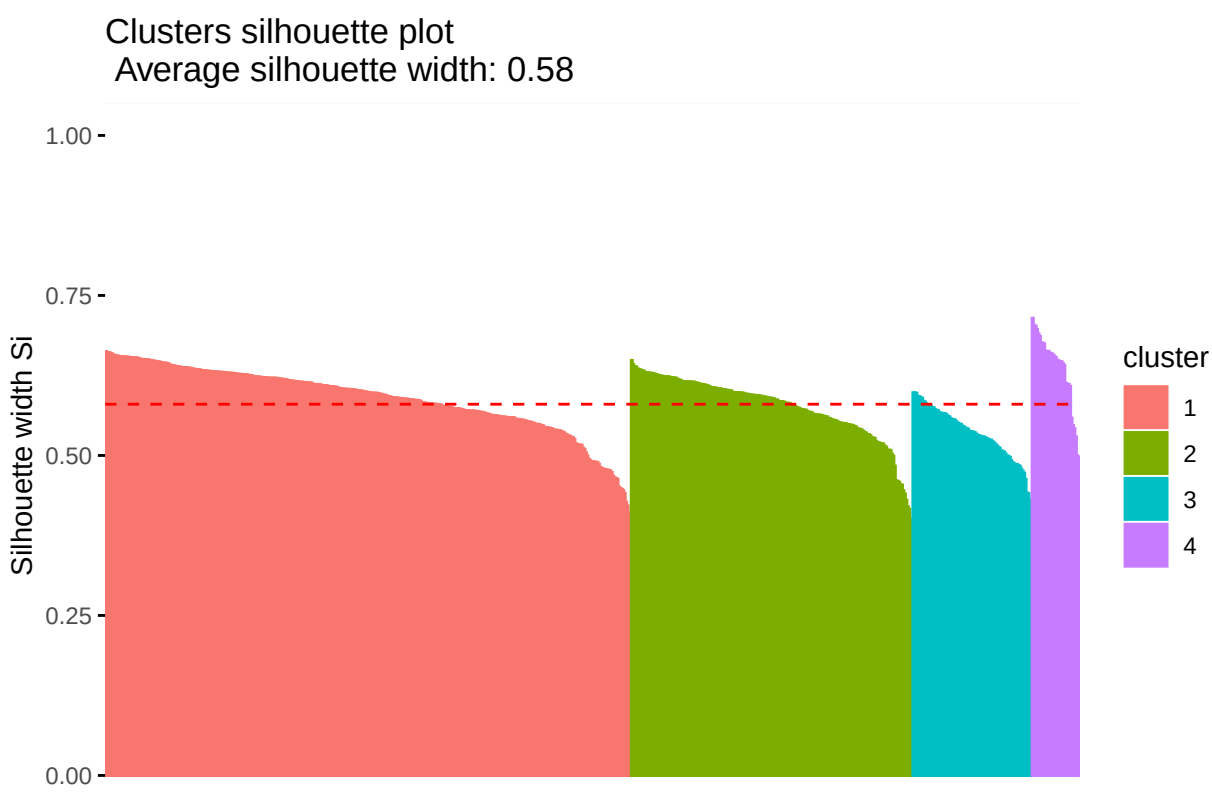
```
## Hacemos una media ponderada entre ambas matrices de distancia
## otorgando el triple de importancia a las distancias provenientes de
## las variables nominales
final.dist <- 0.25*scaled.dist + 0.75*dummy.dist

dbscan.res <- dbscan(final.dist, eps=.7, MinPts = 3, method="dist")
repr.data <- cbind(scaled.data, dummy.data)
fviz_cluster(dbscan.res, data = repr.data, geom = "point")
```




```
## Nos quedamos con aquellos elementos que han sido asignados a
## algún cluster
idx <- dbscan.res$cluster != 0

num.slice <- scaled.data[idx,]
dummy.slice <- dummy.data[idx,]

num.dist <- dist(num.slice)
```

```
dummy.dist <- dist(dummy.slice)

fin.dist <- 0.25*num.dist + 0.75*dummy.dist

sil <- silhouette(dbscan.res$cluster[idx], fin.dist)

fviz_silhouette(sil)
```

```
##   cluster size ave.sil.width
## 1       1  366          0.59
## 2       2  196          0.57
## 3       3   83          0.53
## 4       4   33          0.64
```



Podemos observar que esta modificación ha supuesto una mejora importante en los resultados obtenidos, si medimos la calidad del agrupamiento con el coeficiente de silueta. En la representación gráfica del resultado, tres *clusters* se superponen, al representar todo el conjunto utilizando sólo dos componentes principales (las cuales sólo explican un 40 % de la variabilidad de los datos), pero claramente el agrupamiento es de buena calidad, en función de los coeficientes de silueta obtenidos.

En particular, no hay ningún punto con un coeficiente de silueta negativo, lo que indica que todos los puntos se encuentran bien representados en sus grupos.

Una vez hemos visto el algoritmo de agrupamiento DBSCAN, pasamos a estudiar el clústering jerárquico

5 Clustering jerárquico

Los modelos de *clustering* jerárquico buscan establecer una jerarquía en el agrupamiento. Existen dos aproximaciones a la solución de este problema:

- Clustering aglomerativo: Los algoritmos que utilizan este enfoque parten colocando a cada elemento en un cluster distinto, y van combinando clusters al avanzar hacia arriba en la jerarquía
- Clustering divisivo: En este caso, todas las observaciones comienzan dentro del mismo cluster, y se van dividiendo los grupos en sucesivas etapas.

Usualmente se utiliza un algoritmo de cluster aglomerativo, y es el que viene por defecto implementado en el paquete `stats` de R. Este algoritmo comienza con todos los elementos separados y en cada etapa agrupa los dos *clusters* más similares. El criterio de similaridad que utilizaremos es el criterio de Ward, el cual minimiza la variabilidad intracluster, es decir, en cada etapa une los dos clusters cuyo resultado vaya a aumentar lo menos posible la distancia de los puntos agrupados entre sí. Seguiremos trabajando con la distancia euclídea en primer lugar.

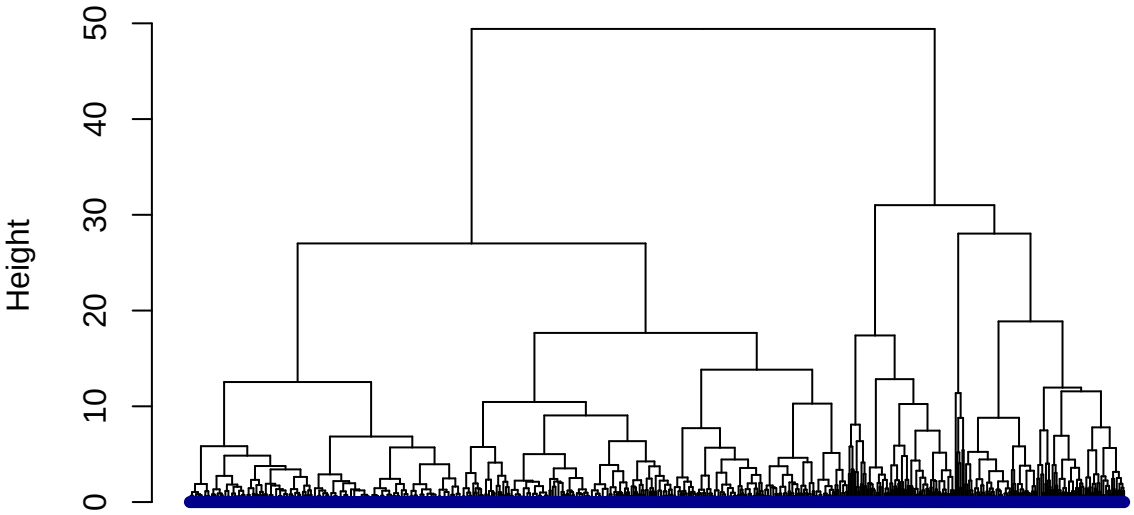
A continación vamos a aplicar el algoritmo de clústering jerárquico a nuestro conjunto de datos

5.1 Aplicación del algoritmo

En primer lugar, como hemos hecho hasta ahora, aplicamos el algoritmo a los datos numéricos, sin tener en cuenta las variables nominales:

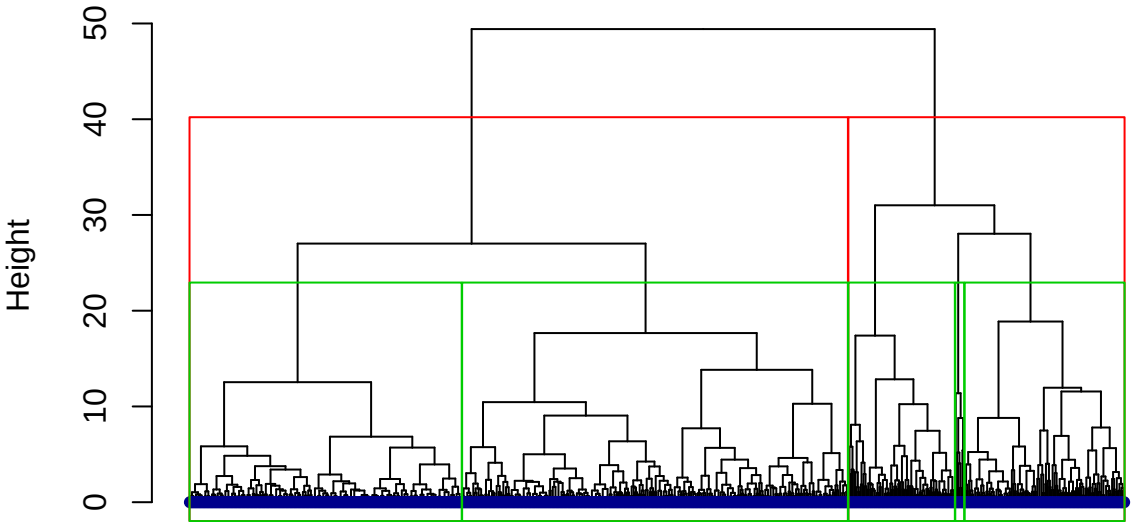
```
dataset <- read.csv(dataset.path, sep=";",  dec = ",")
dataset <- dataset[1:700,]

numeric.data <- dataset %>% select(numeric.vars)
scaled.data <- scale(numeric.data)

numeric.distances <- dist(scaled.data)

hierarch.clust <- hclust(numeric.distances, method="ward.D2")
```

```
nodePar <- list(lab.cex = 0.6, pch = c(NA, 19), cex = 0.7, col =
↪  "darkblue")
plot(as.dendrogram(hierarch.clust), ylab = "Height",
     nodePar = nodePar, leaflab = 'none')
```



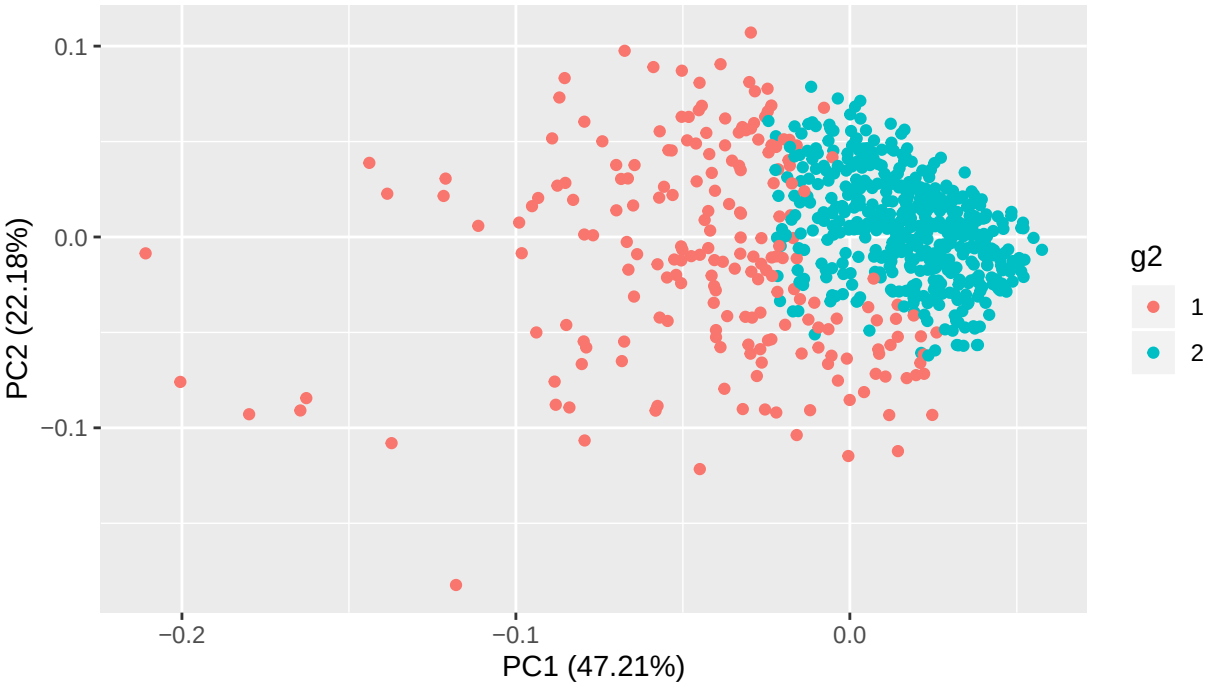
El gráfico anterior es lo que se conoce como dendrograma. En él se muestra el agrupamiento jerárquico de los puntos en los clusters que se han formado. En el eje vertical se coloca la distancia utilizada, y en el eje horizontal los puntos del conjunto. Para cada valor en el eje vertical, la línea horizontal que pasa por dicho punto marca los clústers que se forman utilizando esa distancia como cota máxima. Los puntos que pertenecen a cada cluster son aquellos que cuelgan de cada subárbol que se forma con el corte entre la línea horizontal y el resto del árbol.

Explorando el árbol visualmente, podemos establecer varios puntos en los que cortar el agrupamiento. Por un lado, podemos agrupar en dos grupos fácilmente. Después, el conjunto se separa en tres, y con un aumento muy pequeño de la distancia vuelve a separarse en cuatro e inmediatamente en cinco. Este comportamiento parece indicar que el particionado más natural de los datos es en dos o en cinco subconjuntos. Podemos visualizar fácilmente estos cortes colocando rectángulos sobre el gráfico:

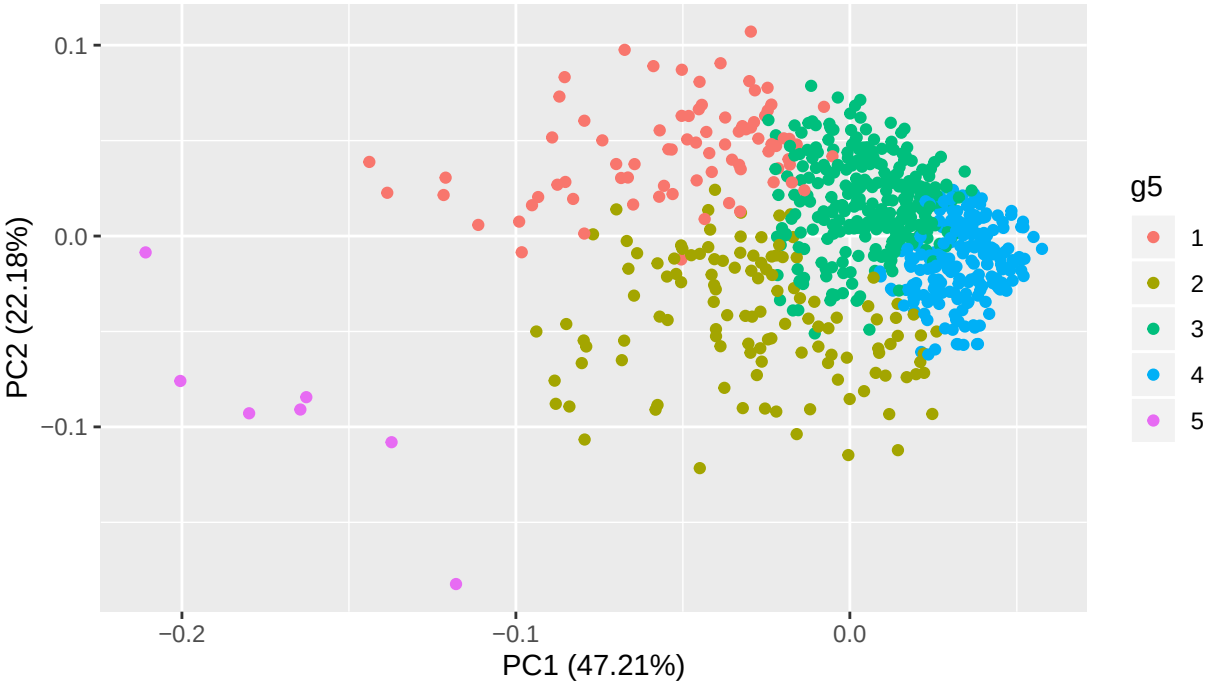


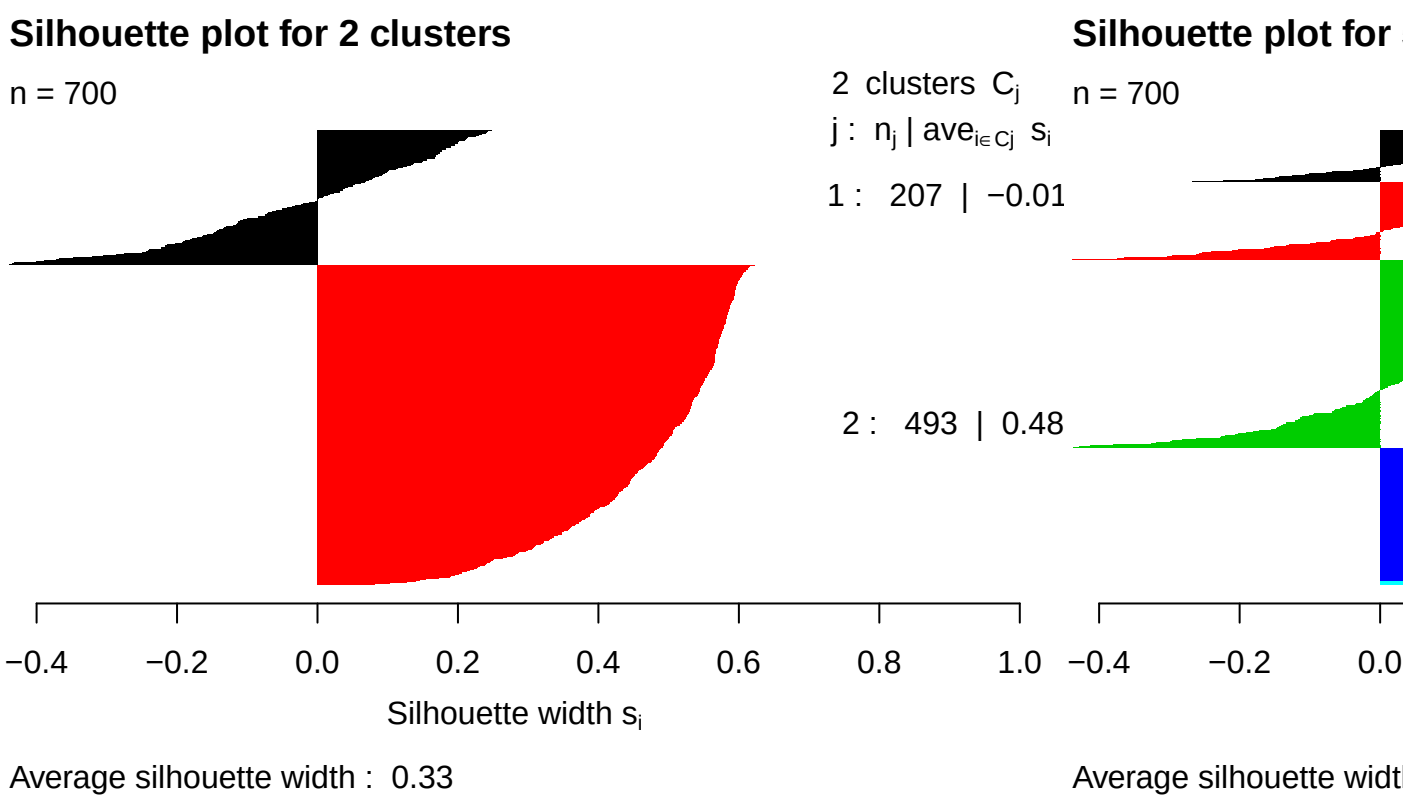
Podemos estudiar cuál de los dos agrupamientos nos proporciona mejores resultados, midiendo la calidad de los mismos utilizando el coeficiente de silueta:

```
groups.2 <- cutree(hierarch.clust, k=2)
groups.5 <- cutree(hierarch.clust, k=5)

repr.data <- data.frame(cbind(scaled.data, 'g2' = groups.2, 'g5' =
↪  groups.5))

repr.data[,'g2'] <- factor(repr.data[,'g2'])
repr.data[,'g5'] <- factor(repr.data[,'g5'])

autoplot(prcomp(scaled.data), data = repr.data, colour = 'g2')
```




```
autoplot(prcomp(scaled.data), data = repr.data, colour = 'g5')
```





Se puede observar que en ninguno de los casos el resultado es especialmente bueno. Especialmente en el caso de los cinco clusters, los coeficientes de silueta de una gran cantidad de puntos en los grupos 1, 2 y 3 son valores negativos. Lo mismo ocurre con aproximadamente la mitad de los puntos del primer grupo en el agrupamiento a dos. De hecho, en el primer caso, el primer cluster tiene un coeficiente de silueta negativo. En el segundo agrupamiento, aunque son de muy baja calidad, todos los agrupamientos tienen valores positivos.

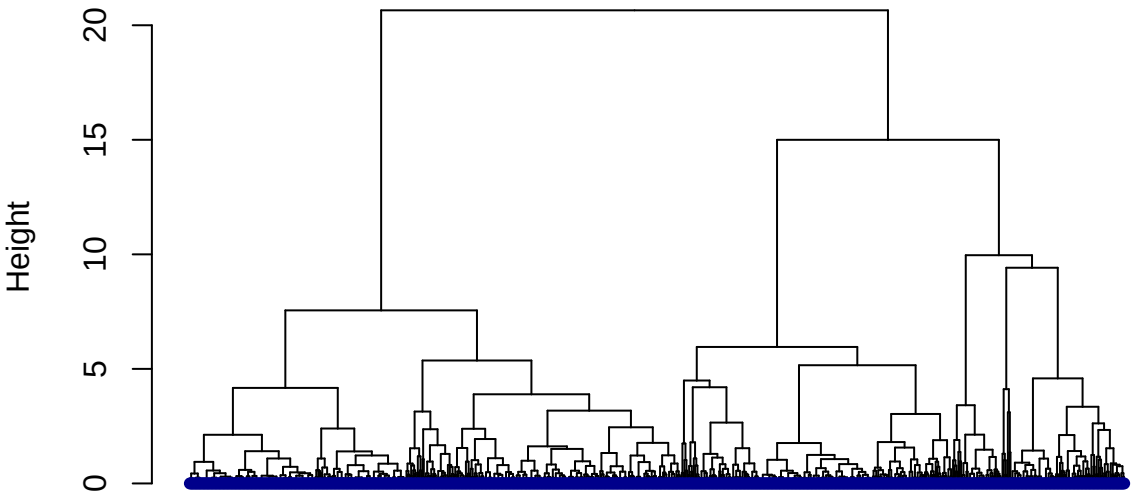
Pasamos a ver cómo podemos combinar las distancias numéricas y nominales para este algoritmo.

5.2 Inclusión de distancias nominales

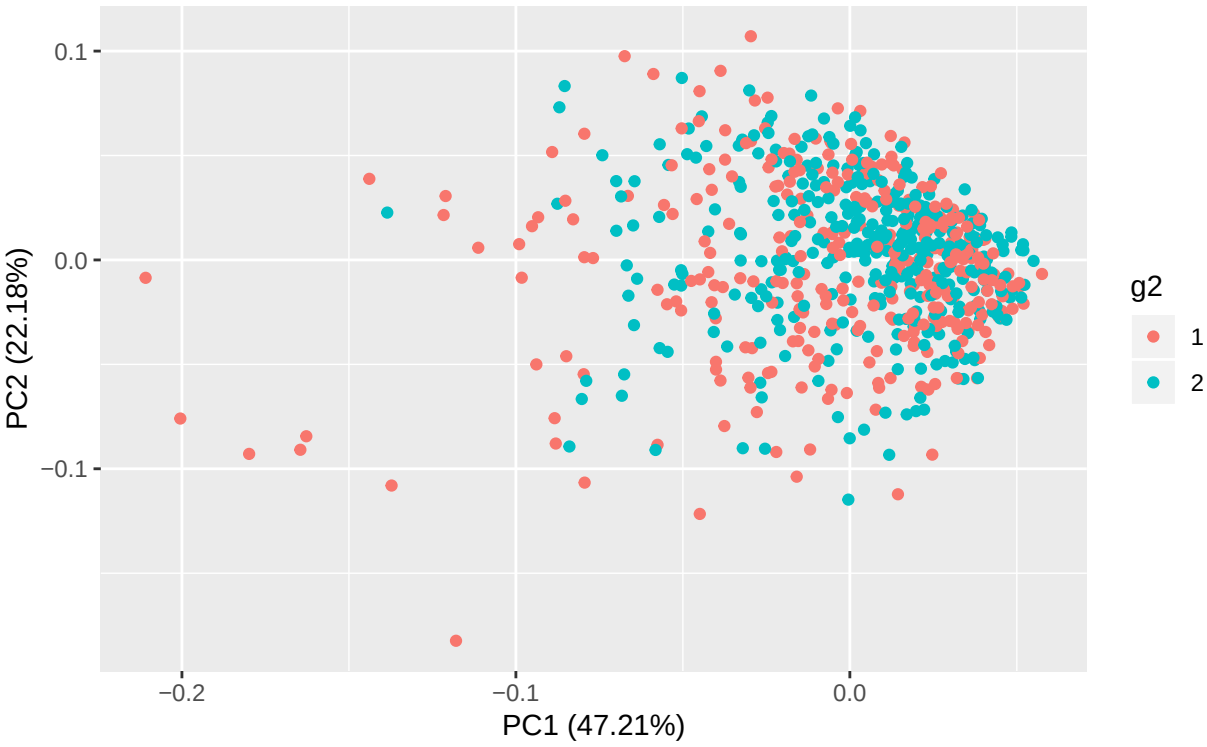
Dado que este algoritmo funciona con distancias entre nuestros puntos, podemos añadir las distancias de las variables nominales de la misma forma que hicimos anteriormente. Observemos cómo cambia el dendrograma cuando añadimos estas distancias. Al igual que hicimos para el algoritmo DBSCAN, vamos a dar una mayor importancia a las variables categóricas:

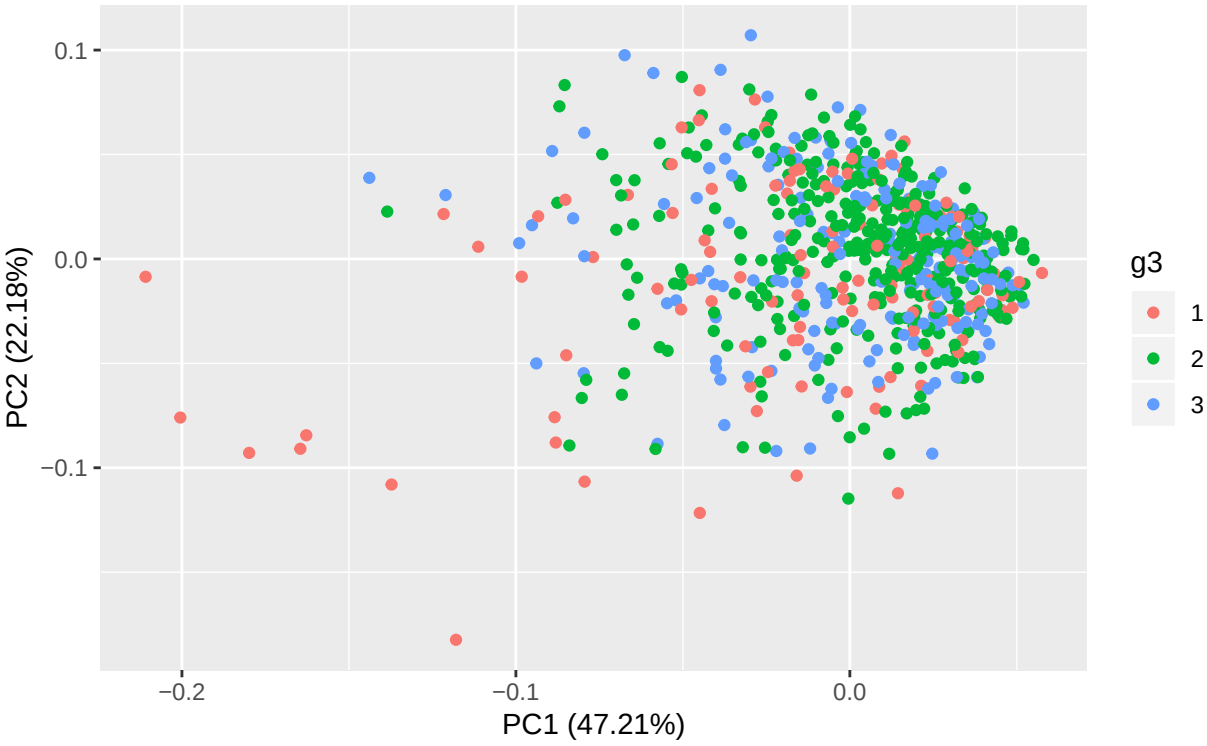
```
dataset <- read.csv(dataset.path, sep=";",  dec = ",")
dataset <- dataset[1:700,]

numeric.data <- dataset %>% select(numeric.vars)
```

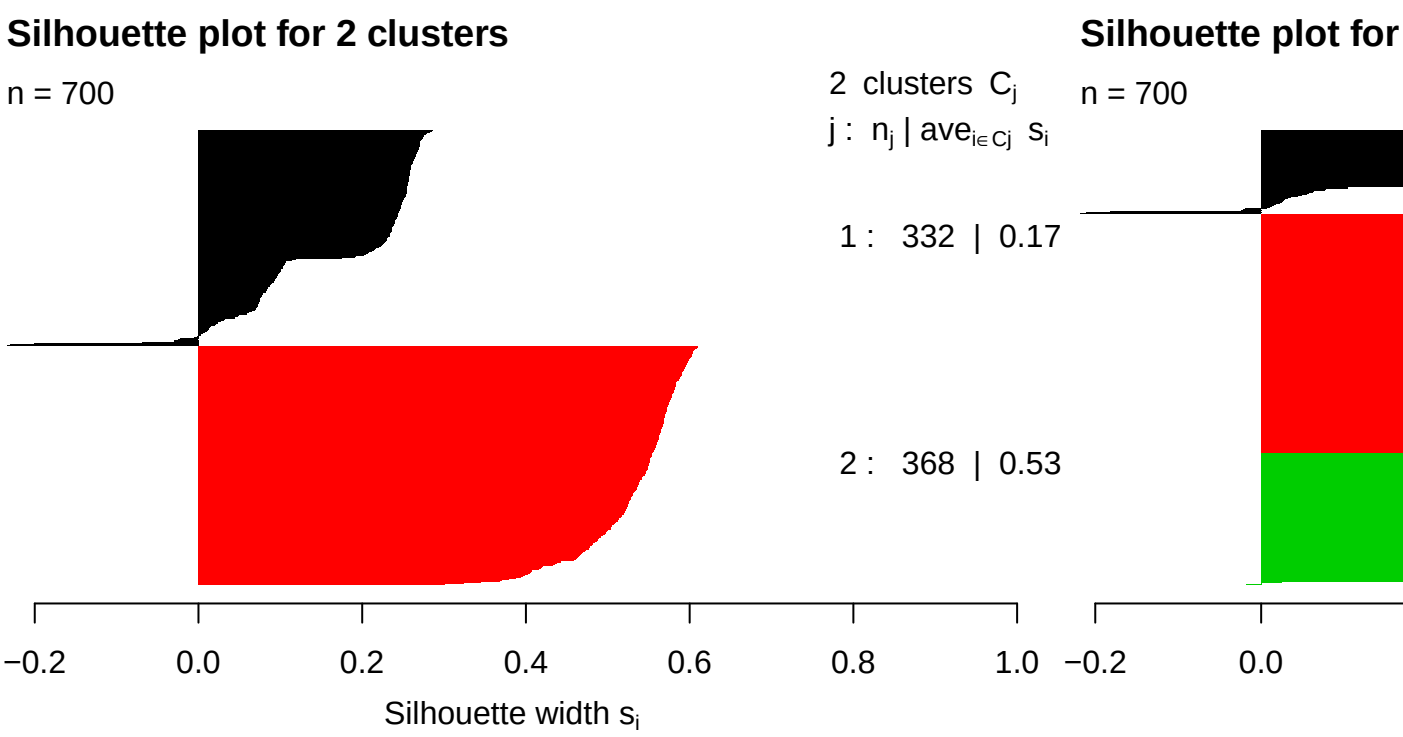
```
nominal.data <- dataset %>% select(nominal.vars)

dummy.data <- dummy_cols(nominal.data)[2:6]

scaled.data <- scale(numeric.data)

scaled.distances <- dist(scaled.data)
dummy.distances <- dist(dummy.data, method="binary")
final.distances <- 0.25*scaled.distances + 0.75*dummy.distances

hierarch.clust <- hclust(final.distances, method="ward.D2")

plot(as.dendrogram(hierarch.clust), ylab = "Height",
     nodePar = nodePar, leaflab = 'none')
```



Podemos ver que existen claras diferencias entre el dendrograma que se generó anteriormente y el actual. En este caso, el agrupamiento en dos conjuntos sigue pareciendo adecuado, pero en lugar del agrupamiento en cinco grupos, parece más lógico el agrupamiento en 3. Veamos los resultados obtenidos al generar esos dos grupos:


0.1
0.0
−0.1
PC2 (22.18%)
−0.2
−0.1
0.0
PC1 (47.21%)
g2
1
2


0.1
0.0
−0.1
PC2 (22.18%)
−0.2
−0.1
0.0
PC1 (47.21%)
g3
1
2
3

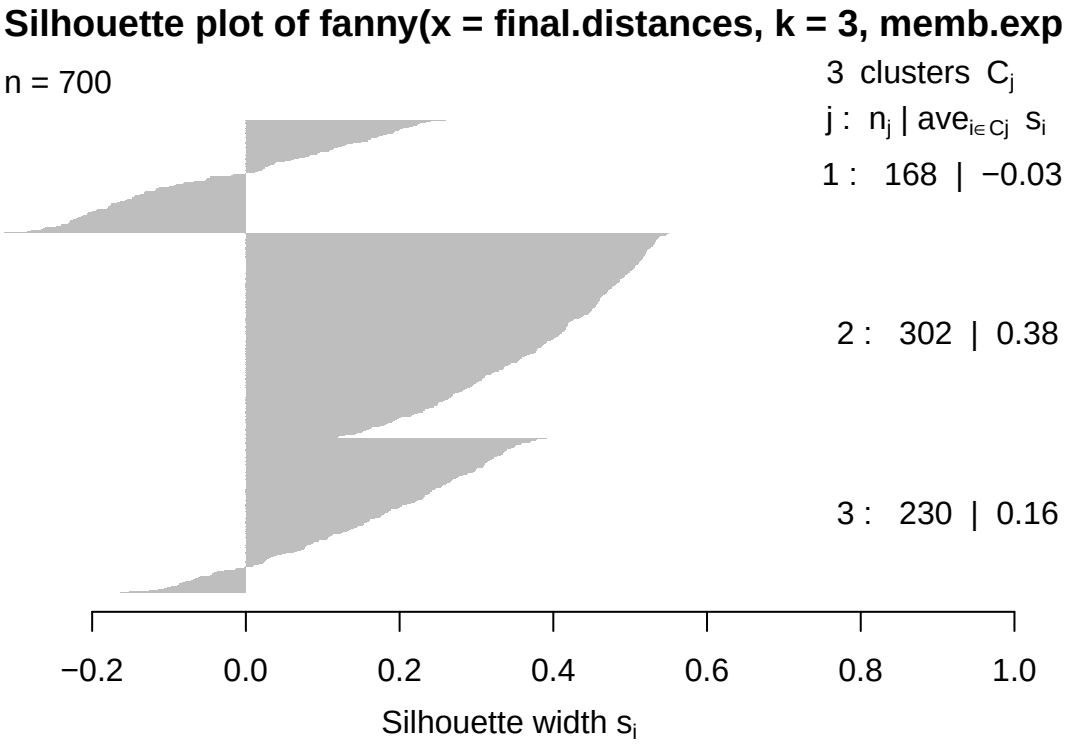


Podemos comprobar que, al igual que ocurrión con el algoritmo DBSCAN, la adición de distancias nominales a la hora de calcular los agrupamientos mejora significativamente la calidad de los resultados obtenidos. A excepción del *cluster* 1, que en ambos casos tiene un coeficiente de silueta relativamente bajo, y con algunos puntos con un valor negativo bastante alto (probablemente estos puntos fueran marcados como outliers por el algoritmo DBSCAN), el resto de clusters calculados son de bastante calidad.

6 Fuzzy k-means

```
dataset <- read.csv(dataset.path, sep=";", dec = ",")
dataset <- dataset[1:700,]

numeric.data <- dataset %>% select(numeric.vars)
nominal.data <- dataset %>% select(nominal.vars)

scaled.data <- scale(numeric.data)

scaled.distances <- dist(scaled.data)
```

```
nominal.distances <- dist(nominal.data, method="binary")

final.distances <- (scaled.distances + nominal.distances) / 2

fuzzy.result <- fanny(final.distances, 3, memb.exp=1.3)

plot(fuzzy.result)
```




```
str(fuzzy.result)
```

```
## List of 10
##  $ membership : num [1:700, 1:3] 0.6728 0.1927 0.1033 0.4123 0.0637 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:700] "1" "2" "3" "4" ...
##   .. ..$ : NULL
##  $ coeff      : Named num [1:2] 0.621 0.432
##   ..- attr(*, "names")= chr [1:2] "dunn_coeff" "normalized"
##  $ memb.exp   : num 1.3
```

```
##  $ clustering : Named int [1:700] 1 2 3 3 2 3 1 3 2 3 ...
##   ..- attr(*, "names")= chr [1:700] "1" "2" "3" "4" ...
##  $ k.crisp    : num 3
##  $ objective  : Named num [1:2] 3.92e+02 1.00e-15
##   ..- attr(*, "names")= chr [1:2] "objective" "tolerance"
##  $ convergence: Named int [1:3] 80 1 500
##   ..- attr(*, "names")= chr [1:3] "iterations" "converged" "maxit"
##  $ diss       : NULL
##  $ call       : language fanny(x = final.distances, k = 3, memb.exp = 1.3)
##  $ silinfo    :List of 3
##   ..$ widths         : num [1:700, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:700] "166" "550" "7" "584" ...
##   .. .. ..$ : chr [1:3] "cluster" "neighbor" "sil_width"
##   ..$ clus.avg.widths: num [1:3] -0.0267 0.3821 0.1589
##   ..$ avg.width      : num 0.211
##  - attr(*, "class")= chr [1:2] "fanny" "partition"
```

```
cluster.stats(final.distances, fuzzy.result$clustering)
```

```
## $n
## [1] 700
##
## $cluster.number
## [1] 3
##
## $cluster.size
## [1] 168 302 230
##
## $min.cluster.size
## [1] 168
##
## $noisen
## [1] 0
##
## $diameter
## [1] 7.135285 2.683818 2.874299
##
```

```
## $average.distance
## [1] 2.0634266 0.8699496 1.1648920
##
## $median.distance
## [1] 1.8579416 0.8299248 1.1351830
##
## $separation
## [1] 0.2165967 0.1524266 0.1524266
##
## $average.toother
## [1] 2.273969 1.838576 1.672825
##
## $separation.matrix
##           [,1]      [,2]      [,3]
## [1,] 0.0000000 0.3791498 0.2165967
## [2,] 0.3791498 0.0000000 0.1524266
## [3,] 0.2165967 0.1524266 0.0000000
##
## $ave.between.matrix
##          [,1]     [,2]     [,3]
## [1,] 0.000000 2.398645 2.110265
## [2,] 2.398645 0.000000 1.429482
## [3,] 2.110265 1.429482 0.000000
##
## $average.between
## [1] 1.904669
##
## $average.within
## [1] 1.253294
##
## $n.between
## [1] 158836
##
## $n.within
## [1] 85814
##
## $max.diameter
## [1] 7.135285
```

```
##
## $min.separation
## [1] 0.1524266
##
## $within.cluster.ss
## [1] 741.8496
##
## $clus.avg.silwidths
##           1           2           3
## -0.02674015  0.38213734  0.15888123
##
## $avg.silwidth
## [1] 0.2106512
##
## $g2
## NULL
##
## $g3
## NULL
##
## $pearsongamma
## [1] 0.3986484
##
## $dunn
## [1] 0.02136237
##
## $dunn2
## [1] 0.6927707
##
## $entropy
## [1] 1.07089
##
## $wb.ratio
## [1] 0.6580112
##
## $ch
## [1] 226.1483
##
```

```
## $cwidegap
## [1] 3.3140624 0.7365743 0.7504805
##
## $widestgap
## [1] 3.314062
##
## $sindex
## [1] 0.2836323
##
## $corrected.rand
## NULL
##
## $vi
## NULL
```