Data Mining en Ciencia y Técnica - TP1

Ariel Aguirre, Miguel Barros, José Badillo, Diego Dell'Era

TP1 - parte 2

Tarea 1

Aplicamos los pasos de la parte 1.

```
glx_0 <- read.csv("dataset.procesado.parte.1.csv", header = T, stringsAsFactors = F)
head(glx_0, 1)
```

```
##   nr   rmag            apdrmag   mcz  ujmag  bjmag  vjmag  usmag gsmag
## 1  6 24.995 0.9350000000000001 0.832 -17.67 -17.54 -17.76 -17.83 -17.6
##    rsmag  ubmag  bbmag  vbmag s280mag
## 1 -17.97 -17.76 -17.53 -17.76  -18.22
```

Analizamos agrupamientos usando k-medias. Pasos:

- Quitamos variables correlacionadas

Las variables que tienen índice de correlación 1, según los resultados de la parte 1, son:

```
# (ujmag, usmag, ubmag)
# (bjmag, bbmag)
# (vjmag, rsmag, vnmag)
```

Se trata, obviamente, de los grupos de magnitudes medidas en una misma banda. Nos quedamos con 1 variable que represente a cada grupo:

```
glx_1 <- glx_0[,c(1:7)]
```

- Estandarizamos variables

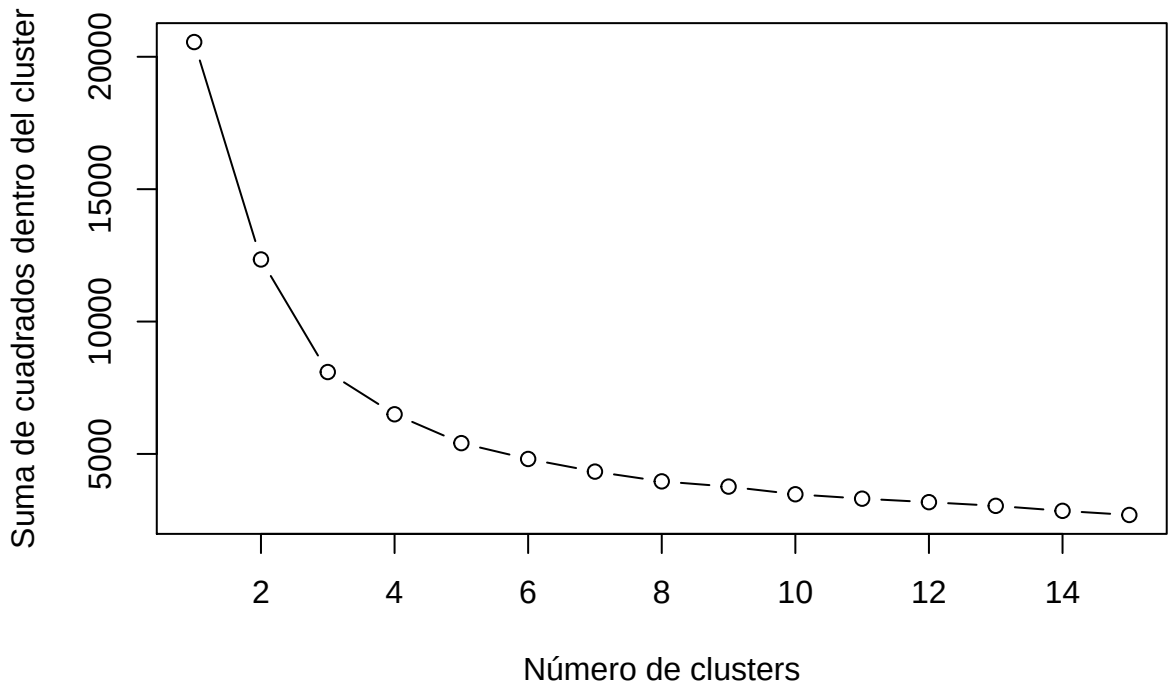
```
# sólo estandarizamos variables con mediciones (sin la primera columna, porque es el número de galaxia)
glx_2 <- scale(glx_1[,c(2:7)])
head(glx_2, 1)
```

```
##                    rmag           apdrmag                mcz
## [1,] 0.7481409363139495 2.530149066971338 0.3218825827357346
##                   ujmag              bjmag              vjmag
## [1,] 0.1184136486227584 0.1237063466556295 0.1841887849416674
```

- Determinamos el `k` óptimo para K-means:
- Gráficamente

Podemos ver el `k` óptimo como el punto de corte en que deja de disminuir marcadamente la suma de cuadrados dentro del cluster. En otras palabras, el punto en que la cohesión de los clusters deja de aumentar.

```r
wssplot <- function(data, nc=15, seed=1234){wss <- (nrow(data)-1)*sum(apply(data,2,var))
               for (i in 2:nc){
                    set.seed(seed)
                    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
                plot(1:nc, wss, type="b", xlab="Número de clusters",
                     ylab="Suma de cuadrados dentro del cluster")}

wssplot(glx_2)
```



El gráfico sugiere usar k = 2 ó 3 (es discutible).

- Analíticamente

Calculamos una matrix de distancias:

```r
library(cluster)
glx_2_dist <- dist(glx_2)
```

Usamos la matriz de distancias con kmeans:

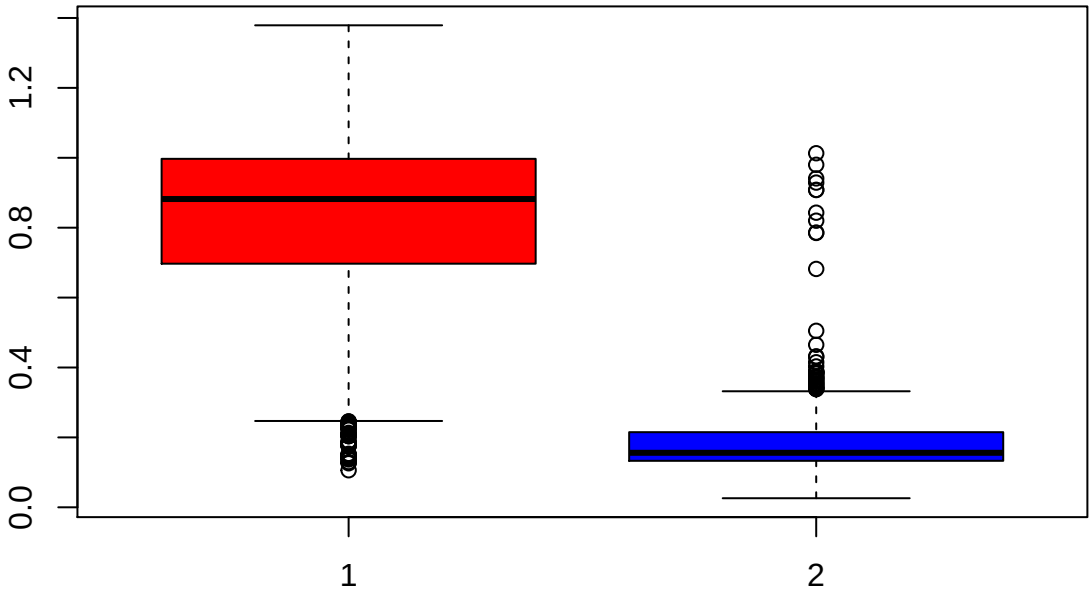
```r
# una función que, dado un k, clusteriza por kmeans -> calcula silhouette -> devuelve ancho promedio
silhoutte_kmeans <- function(k) {
  c <- kmeans(glx_2, centers = k)$cluster
  s <- silhouette(c, glx_2_dist)
  return(summary(s)$avg.width)
}

# aplicamos la función para valores de k entre 2 y 10
indice_por_k_kmeans <- sapply(c(2:10), silhoutte_kmeans)
which.max(indice_por_k_kmeans)
```

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

El k que maximiza el ancho promedio de silhouette para todos los clusters creados via K-means es el primero => k = 2.

Hay 2 grupos. Leyendo un poco la descripción del dataset, una interpretación posible es que los grupos representen galaxias 'azules' y 'rojas'; en otras palabras, que los grupos dependen del corrimiento al rojo. Si lo ploteamos, las medias difieren bastante:

```
k2 <- kmeans(glx_2, centers = 2)
boxplot(glx_1$mcz ~ k2$cluster, col=c("red", "blue"))
```



Éstos son los centros de cada cluster (revirtiendo a las medidas originales). Se ve que el redshift mayor o menor que 0 divide a los grupos:

```
aggregate(glx_2, by = list(k2$cluster), mean)
```

```
##   Group.1                  rmag                apdrmag                   mcz
## 1       1  0.005773528295152889 -0.009751884029828599  0.317479333145156
## 2       2 -0.030731788065529994  0.051908091191984349 -1.689903830486670
##                  ujmag                bjmag                vjmag
## 1 -0.3471593365751252 -0.3401438286077606 -0.3361592806310137
## 2  1.8478868745742363  1.8105441799509021  1.7893349162739356
```

Tarea 1 (optativa): Comparar métodos

Ahora usamos PAM en lugar de K-means:

```
# una función que, dado un k, clusteriza por PAM -> calcula silhouette -> devuelve ancho promedio
silhoutte_pam <- function(k) {
  c <- pam(glx_2_dist, k=k, diss=T)
  s <- silhouette(c)
  return(summary(s)$avg.width)
```

```
}

# aplicamos la función para valores de k entre 2 y 5
indice_por_k_pam <- sapply(c(2:5), silhoutte_pam)
which.max(indice_por_k_pam)
```

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

El k que maximiza el ancho promedio de silhouette para todos los clusters creados via PAM es el primero => k = 2.

Ventajas de PAM sobre K-means:

- es más robusto ante la presencia de outliers (porque la mediana es más robusta que la media);
- se puede usar cualquier medida de similitud;
- como usa medoides, encuentra objetos representativos del cluster => el resultado se puede interpretar mejor estudiando las propiedades de un objeto particular.

Principal desventaja de PAM ante K-means:

- es más lento, porque tiene una complejidad del orden de O(n^2 * k * i) (donde n = cantidad de objetos, k = cantidad de medoides, i = iteraciones). K-means, en cambio, tiene una complejidad de O(n * k * i) (evita el término cuadrático porque sólo tiene que calcular distancias de los objetos a los centroides).

También podemos probar otra biblioteca para clusterizar maximizando el índice de Silhouette:

```
library(NbClust)
set.seed(1234)
nc <- NbClust(glx_2, min.nc = 2, max.nc = 5, method = "kmeans", index = "silhouette")
nc$Best.nc
```

```
## Number_clusters     Value_Index
##          2.0000          0.4915
```

Este método también sugiere usar k = 2.

Tarea 2

Cargamos el dataset entero (partimos de un .csv que fue creado a partir del dataset original descargado del website y parseado según las instrucciones para cada campo):

```
glx <- read.csv("dmcyt_tp2.csv", header = T, stringsAsFactors = F, sep = '|')
glx$mc_class <- as.factor(glx$mc_class)

# evitemos que R trunque decimales
options(digits=16)
```

Quitamos outliers:

```
glx_sin_outliers <- subset(glx, apd_rmag > -3.2)
glx_sin_outliers <- subset(glx_sin_outliers, bjmag < -7.0)
glx_sin_outliers <- subset(glx_sin_outliers, ujmag < -10.0)
glx_sin_outliers <- subset(glx_sin_outliers, vjmag < -10.0)
glx_sin_outliers <- subset(glx_sin_outliers, usmag < -10.0)
glx_sin_outliers <- subset(glx_sin_outliers, gsmag < -9.0)
glx_sin_outliers <- subset(glx_sin_outliers, rsmag < -9.0)
# glx_sin_outliers <- subset(glx_sin_outliers, bbmag < -9.0) # todos los valores en 0?
# glx_sin_outliers <- subset(glx_sin_outliers, ubmag < -10.0) # todos los valores en 0?
# glx_sin_outliers <- subset(glx_sin_outliers, vbmag < -10.0) # todos los valores en 0? (antes era vnma

nrow(glx) - nrow(glx_sin_outliers)
```

```
## [1] 8181
```

Buscamos datos faltantes:

```
glx_sin_faltantes <- glx_sin_outliers[complete.cases(glx_sin_outliers),]
nrow(glx_sin_outliers) - nrow(glx_sin_faltantes)
```

```
## [1] 720
```

Dimensiones del dataset final:

```
dim(glx_sin_faltantes)
```

```
## [1] 54600    26
```

Tarea 3

Extraemos las variables que tienen mediciones para poder encontrar clusters:

```
glx_tarea3.0 <- glx_sin_faltantes[,c(11,13,17,18,19)]
```

Ahora calculamos distancias para correr PAM y después Silhouette:

```
# glx_tarea3.1 <- dist(scale(glx_tarea3.0))
```

Pinchó. ¿Por qué?

Calcular la matriz de distancias implica que, por cada elemento de la matriz, hay que calcular la distancia a todos los demás elementos. Es decir, es un procedimiento cuadrático: si la matriz tiene `n` elementos, hay que almacenar `n x n` distancias.

Se puede deducir, para el caso de la implementación de R, dónde está el problema de alocación de memoria:

```
# tomamos las primeras 16000 filas
glx_muestra_16k <- glx_sin_faltantes[c(1:16000), c(11,13,17,18,19)]

# intentamos calcular la matriz, y la imprimimos para forzarlo a que nos devuelve un mensaje de error
# dist(scale(glx_muestra_16k))
```

Por el mensaje de error, vemos que no puede alocar espacio (0.975 Gb). Entonces podemos despejar la cantidad de bytes que usa para guardar distancias:

```
0.975 * (1024)^3 / (16000^2)
```

```
## [1] 4.0894464
```

Son 4 bytes. Usando apenas 4 bytes para almacenar distancias, el cálculo para una matriz de tamaño modesto se vuelve demasiado costoso, rápidamente.

Podemos probar de nuevo restringiendo el tamaño del dataset (y limpiando un poco con gc()).

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   407387  10.9     750400   20.1    750400   20.1
## Vcells 67767914 517.1  188229511 1436.1 195853105 1494.3
```

Tomamos una muestra de 10000 objetos (en la compu donde corre esto, si tomamos más de 15000 pincha). Agrupamos con K-means buscando el `k` óptimo:

```
filas_muestra <- sample(nrow(glx_sin_faltantes), size = 10000)
glx_muestra_10k <- glx_sin_faltantes[filas_muestra,]
glx_muestra_10k_mediciones <- glx_muestra_10k[,c(11,13,16,17,18,19)]
head(glx_muestra_10k_mediciones, 1)
```

```
##         rmag ap_rmag  mc_z  ujmag  bjmag  vjmag
## 31278 24.193  24.278 0.965 -19.02 -18.93 -19.11
```

```
glx_muestra_10k_dist <- dist(scale(glx_muestra_10k_mediciones))

silhoutte_kmeans <- function(k) {
  c <- kmeans(glx_muestra_10k_mediciones, centers = k)$cluster
  s <- silhouette(c, glx_muestra_10k_dist)
  return(summary(s)$avg.width)
}

# aplicamos la función para valores de k entre 2 y 5
indice_por_k_kmeans <- sapply(c(2:5), silhoutte_kmeans)
which.max(indice_por_k_kmeans)
```

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

Obtenemos el primer k, o sea $\texttt{k} = 2$.

(Sería interesante investigar si el "valle verde" que está entre las galaxias rojas y la nube de galaxias azules en el gráfico de color-magnitud aparecería ampliando la muestra. . .)