k-means clustering

STAT30270 – Statistical Machine Learning

Contents

1 k-means clustering

The main function for k-means clustering in R is `kmeans`. The function takes the data in input and using the argument `centers` you need to specify the number of clusters. The default is to use random starting centroid points to initialize the algorithm, but specific starting centers can also be given using the same argument `centers`. Take some time to familiarize with the documentation `?kmeans`.

We run the k-means algorithm on the dataset `penguins`, available in the package `palmerpenguins`. The data are a sample of 3 different penguin species and include four numerical variables concerning measurements of flipper length, body mass, and bill dimensions. More info available in the help page `?penguins` and at this link. The data contain some missing data, and we remove them using function `na.omit`. The data is also in `tibble` format (see `?dyplr::tibble`), and it is converted into a standard `data.frame` object. We then run k-means on the data with $K = 3$ clusters, using the following code.

```r
# load and prepare the data
library(palmerpenguins)
penguins <- as.data.frame( na.omit(penguins) )

# we only implement k-means on the numerical features
x <- penguins[,3:6]

# plot the data - do you see clusters?
pairs(x, gap = 0, pch = 19, col = adjustcolor(1, 0.4))
```

```r
# run k-means (with K = 3)
fitkm <- kmeans(x, centers = 3)
```

As we noted in class, because the algorithm is initialized from random starting points, it could reach a local minimum. Good practice to avoid this problem is to run k-means starting from multiple initial points and then keep the best solution in terms of total within sum of squares. This process is implemented automatically in the `kmeans` function. Using the `nstart` argument is possible to specify the number of random starts for the algorithm.

```r
# run k-means with 20 random starts
fitkm <- kmeans(x, centers = 3, nstart = 20)

# examine the results
fitkm
```

We can visualize the clustering solution obtained and compare it to the iris classification into the three species.

```r
col <- c("darkorange2", "deepskyblue3", "magenta3")  # some colors

# plot with color corresponding to the species
pairs(x, gap = 0, main = "Species", pch = 19,
      col = adjustcolor(col[penguins$species], 0.4))

# plot with color corresponding to the clusters
pairs(x, gap = 0, main = "Clustering result - K = 3", pch = 19,
      col = adjustcolor(col[fitkm$cluster], 0.4))
```

1.1 Task

- The clustering obtained from k-means does not visually resemble the classification into species of the penguins. Some high density areas of the data seem to have split into multiple clusters. Can you tell why? …**Because of the Euclidean distance in the objective function of k-means, the scale and range of variation of the variables can have an important impact on the clustering results!** Use function `scale` to standardize the data and rerun k-means with $K = 3$ and produce the same pairs plot above, what do you notice now?
- Try a few different values of K and see how the algorithm works and what clustering results you obtain.
- Inspect the output of the `kmeans` function and see what each part of the output tells us. How many observations each cluster has? Can you find the cluster centroids in the output? What are the slots `withinss`, `totss`, and `betweenss`?

2 Internal validation and selection of the number of clusters

Selection of the number of clusters and assessment of the results are important and difficult tasks in clustering. We can use different tools to the purpose.

In this section we will consider an example analysis on data concerning chemical characteristics of Italian wines from three different cultivars, Barolo Grignolino and Barbera. The data are available in the `gclus` package and contain 13 measurements on 178 samples. The numerical variables appear to be on different ranges. Sometimes it is useful to standardize the data in this situation, ensuring that the variables have all mean 0 and standard deviation 1, and hence they are on the same range of variation. This is a common operation, as several statistical machine learning methods are sensitive to the scale and range of the data. This is particularly the case for k-means, as the objective function of the algorithm employs the Euclidean distance (see previous tasks). Function `scale` allows to easily standardize the data.

```r
# load the data
data(wine, package = "gclus")

# rename type of wine in a readable way
type <- factor( wine$Class, labels = c("Barolo", "Grig", "Barbera") )

# we can easily standardized the data using the function 'scale'
x <- scale(wine[,-1])     # remove first column with type wine
```

2.1 Elbow plot and Calinski-Harabasz index

We run k-means on the `wine` data for different values of K. Storing within sum of squares and the between sum of squares, with the output we can calculate and plot the Calinski-Harabasz index, while the within sum of squares can be readily used to produce an elbow plot.

```r
K <- 10                # set K max
wss <- bss <- rep(NA, K)      # initialize empty vector

for ( k in 1:K ) {
  # run kmeans for each value of k
  fit <- kmeans(x, centers = k, nstart = 50)
  wss[k] <- fit$tot.withinss     # store total within sum of squares
  bss[k] <- fit$betweenss
}

# compute calinski-harabasz index
N <- nrow(x)
ch <- ( bss/(1:K - 1) ) / ( wss/(N - 1:K) )
ch[1] <- NA        # the value of CH index for K = 1 is not defined!

# plot the two and compare
par( mfrow = c(1,2) )   # split the plot window in 2 screens
plot(1:K, wss, type = "b", ylab = "WSS", xlab = "K", main = "WSS")
plot(1:K, ch, type = "b", ylab = "CH", xlab = "K", main = "CH")
par( mfrow = c(1,1) ) # reset
```

There is really no way to tell where the elbow is from the within sum of squares plot. Instead, looking at the CH index, a solution with 2 or 3 clusters seems plausible, with preference for $K = 3$. Let's compare them visually.

```r
fit2 <- kmeans(x, centers = 2, nstart = 50)
fit3 <- kmeans(x, centers = 3, nstart = 50)

col <- c("darkorange2", "deepskyblue3", "magenta3")

par( mfrow = c(1,2) )
pairs(x[,c(1,5,7,0,13)], gap = 0, pch = 19,
      col = adjustcolor(col[fit2$cluster], 0.4), main = "Clustering result - K = 2")
```

```r
pairs(x[,c(1,5,7,0,13)], gap = 0, pch = 19,
      col = adjustcolor(col[fit3$cluster], 0.4), main = "Clustering result - K = 3")
```

2.2 Silhouette

The silhouette plot is used to assess cluster cohesion. Observations with high silhouette value are much closer to their own cluster members than the members of other clusters. Observations with low (or even negative) silhouette are close to other cluster members and thus may have ambiguous cluster membership.

The `cluster` package in has a function called `silhouette` which computes the silhouette for each observation. Take some time to read the documentation `?silhouette`. The function takes two arguments: a clustering of the data and a distance matrix. It is natural to use the same distance when clustering the data and for computing the silhouette, but this is not required. For k-means, we use the squared euclidean distance. Let's compute the average silhouette for different values of K.

```r
library(cluster)

# construct a distance matrix using squared Euclidean distance
d <- dist(x, method = "euclidean")^2

# run k-means and compute silhouette for multiple values of K
```

```r
ave_sil <- rep(NA, K)
sil <- vector("list", K)
for ( k in 2:K ) {
  km <- kmeans(x, k, nstart = 20)
  sil[[k]] <- silhouette(km$cluster, d)
  ave_sil[k] <- mean(sil[[k]][,"sil_width"])
}
ave_sil[1] <- 0

# plot
plot(1:K, ave_sil, type = "b", ylab = "CH", xlab = "K", main = "Avg silhouette")
ave_sil
```

Also according to average silhouette, a solution with 3 clusters seem more appropriate, followed this time by solutions with 2 and 4 clusters. We can produce a silhouette plot for $K = 2$ and $K = 3$.

```r
# produce the two silhouette plots
col <- c("darkorange2", "deepskyblue3", "magenta3")
#
# par( mfrow = c(1,2) )
plot(sil[[2]], col = adjustcolor(col[1:2], 0.9),
     main = "Wine data -  K = 2")
plot(sil[[3]], col = adjustcolor(col, 0.9),
     main = "Wine data -  K = 3")
```

2.3 Gap statistic

The gap statistic is used to compare the within sum of squares computed on the data for a given value of K against that one obtained by clustering the data with the same K but under the assumption that they were uniformly distributed at random in the data space. If the gap is "large", there is indication that the within sum of squares obtained with a given value of K is dissimilar from that one that would be obtained if the data to be clustered had no clusters at all. The estimated number of clusters is then the smallest value of K for which the value of the gap statistic is greater than the first local maximum minus its estimated standard error.

To implement and compute the gap statistic we can use the `clusGap` function in the `cluster` package. The function takes in input the data, the method/function used for clustering, the maximum number of clusters to be considered, and other arguments. Take some time to read the documentation `?clusGap`. By default, the argument `spaceH0 = scaledPCA`, which means that the clustering and the calculations of the gap statistic are based on the scaled principal components of the data. The function takes in input the data and the function used to cluster them through the argument `FUNcluster`. We compute the gap statistic on the wine data for a maximum number of clusters of 10.

The function has an associated plotting method that can be used to visualize the gap values and associated 1 standard error intervals.

```r
gap_stat <- clusGap(x, FUNcluster = kmeans, K.max = 10, nstart = 20)
gap_stat
plot(gap_stat)
```

The figure suggests a solution with 3 clusters is more appropriate for these data.

We can also use the quantities in output to identify the optimal number of clusters, reaching the same conclusion.

```r
# identify first local maximum, i.e. first K for which gap(K+1) - gap(K) < 0
lmax <- which(diff(gap_stat$Tab[,3]) < 0)[1]
#
# compute gap - se for this local optimum
thres <- gap_stat$Tab[lmax,3] - gap_stat$Tab[lmax,4]
#
# identify smallest value ok K for which gap(K) > (gap - se), i.e.
# the smallest K for which gap(K) - (gap - se) > 0
```

```r
gap_stat$Tab[,3] - thres
which( (gap_stat$Tab[,3] - thres) > 0 )[1]
```

2.4 Task

- Take some time to read the silhouette plot figure. What are the different quantities shown? See also `?plot.silhouette`. Compute the average silhouette using the quantities in the figure.
- Check the output from the function `clusGap`. What are the different quantities `logW`, `E.logW`, `gap`, `SE.sim`? Using those quantities in the table, and following the steps above, identify the optimal number of clusters
- Change the argument `spaceH0 = "original"` and rerun the function `clusGap`. How many clusters are suggested now? Do you have an intuition of why such larger number of clusters is suggested by the statistic?
- Go back to the (standardized) `penguins` data and use different methods to select the appropriate number of clusters for these. Compare the number of clusters suggested by the different procedures. Do the different methods all agree on the selected number of clusters?

3 External validation with the adjusted Rand index

A cross tabulation can be used to compare different clusterings of the data and also a clustering solution versus an external reference partition. This comparison can also be performed using the adjusted Rand index.

The `e1071` package for can be used for computing the Rand and adjusted Rand indices. The command for computing the adjusted Rand index (and Rand index) is `classAgreement`. The slot containing the adjusted Rand index is `crand`. Let's use it to compare the results of k-means with $K = 3$ or $K = 2$and the wine types.

```r
# load package
library(e1071)

# cross tabulation between the two partitions -- K = 2
tab2 <- table(fit2$cluster, type)
tab2
classAgreement(tab2)

# cross tabulation between the two partitions -- K = 3
tab3 <- table(fit3$cluster, type)
tab3
classAgreement(tab3)
```

4 Spotify songs data

Dataset `data_spotify_songs.rda` contains data about audio features for a collection of songs extracted from the music streaming platform Spotify. The songs are classified into three genres: `acoustic`, `pop`, `rock`. For each song, measurements about some numerical audio features and popularity are recorded; more information about the numerical audio features is available here:

`https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/`

4.1 Task

Perform a cluster analysis of the numerical audio features data using k-means:

1. Consider a range of values of K from 1 to 10 and use different methods to select the appropriate number of clusters for this data.

2. Compare the clustering obtained to the classification into genres. Is there agreement between the clusters you found and the genres? What genres characterize the different clusters?
3. Standardize the data and implement again k-means, also selecting the appropriate number of clusters for the standardized data. Does your clustering results change in comparison to the k-means applied on the original data?