'From Statistics to Data Mining' Computer Lab Session n° 9: Clustering (1/2)

Master 1 COSI / CPS[2]
Saint-Étienne, France

Fabrice Muhlenbach

Laboratoire Hubert Curien, UMR CNRS 5516
Université Jean Monnet de Saint-Étienne
18 rue du Professeur Benoît Lauras
42000 SAINT-ÉTIENNE, FRANCE
https://perso.univ-st-etienne.fr/muhlfabr/

Outcome

The objective of this lab is to become familiar with a simple clustering algorithm (k-means) and its translation in a ® program.

1 Programming k-means

1.1 k-means Algorithm

k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

This algorithm works as follows:

- Given a set of individuals $(x_1, x_2, \ldots, x_n)$, where each individual is a vector of dimension d, k-means algorithm aims to partition the n individuals into k sets $S = S_1, S_2, \ldots, S_k$ ($k \leq n$) while minimizing the distance between each point into each partition:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of the points in S_i.

- Choose k points representing the mean positions of the initial partitions $m_1^{(1)}, \ldots, m_k^{(1)}$ (e.g., randomly)

- Repeat until convergence:

– **Assignment step:** Assign each observation to the cluster whose mean yields the least within-cluster sum of squares. Since the sum of squares is the squared Euclidean distance, this is intuitively the 'nearest' mean (e.g., according to the Voronoi diagram generated by the means):

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - \mathbf{m}_i^{(t)} \right\| \leq \left\| \mathbf{x}_j - \mathbf{m}_{i^*}^{(t)} \right\| \forall\, i^* = 1, \ldots, k \right\}$$

– **Update step:** Calculate the new means to be the centroids of the observations in the new clusters:

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

• Convergence is reached when there is no more change.

1.2 Choice of an illustrative dataset

1.2.1 Iris Dataset

We will program the k-means algorithm so that we will see its changes occurring during the partitioning of the data.

For this we will use this program with a classical dataset: iris plant. This dataset is a set of measurements made on 150 flowers of iris plants with three classes (*iris setosa*, *iris versicolor*, and *iris virginica*). In the dataset, there are 50 representatives of each species. The 4 variables measured (in centimeters) are the widths and lengths of the sepals and petals.

Iris dataset is present in the packages launched by default when loading ®, so we can use the dataset simply by loading into memory with the data function. We will only copy the variables measures (the first 4) in a data frame called mydata, the fifth variable will be copied in a matrix called class and can be used for further processing (e.g., to measure the quality of final partition obtained).

```
data ( " i r i s " )
mydata <− i r i s [ 1 : 4 ]
class <− as . matrix ( i r i s [ 5 ] )
```

To maintain still some genericity of our algorithm, we get the number of columns (i.e, the number of variables, here: 4) and the number of rows (i.e., the number of individuals, here: 150).

```
n_c <− ncol ( mydata )
n_r <− nrow ( mydata )
dim <− n_r
```

1.2.2 Data Preprocessing

In a clustering process, we do not know in advance if some variables are more relevant than the others. This relevance difference exists in the iris dataset: the length and width of the petals (the third and fourth variables) are better than the sepal information (as shown by the representation obtained with the pairs graph obtained by pairs(mydata)).

Therefore, without any other information on the data, it is wiser to standardize the data for having a z-scores (i.e., having zero for the mean and one for the standard deviation by subtracting each value by the

mean and dividing all by the standard deviation), assuming that the data follow a normal distribution. In the case of iris dataset, this assumption is not quite correct: the data follow a normal distribution only for a particular species, not for all three species. We copy the dataset in mydata_zs (centered and reduced data after z-scores transformation).

You should know that with ℝ any function (already present or created by the user) can be applied to all elements of a matrix (or only certain rows or columns thereof) of a vector or a list using the functions apply, sapply or lapply (see help for more information). Standardization will apply to each variable (loop for).

```
mydata_zs <- mydata
for (i in 1:n_c) # Use of sapply function for computing directly
  { # the mean or the standard deviation on an attribute of a dataset
  mydata_zs[i] <- ((mydata_zs[i] - sapply(mydata_zs[i],mean))
                    / sapply(mydata_zs[i],sd))
  }
```

Nevertheless, using a for loop is in ℝ not it recommended because it is not very efficient. With apply function, we can program a "normalize" function for applying the normalization on each row.

```
normalize <- function(row) { (row - mean(row)) / sd(row) }
mydata_zs <- apply(mydata, 2, normalize)
```

1.2.3 Reading the Number of Clusters

The most important parameter of the k-means algorithm is the number of clusters (or partitions) that are desired at the end. This is the number 'k' of the k-means.

```
k <- readline("Number of clusters? ")
print(paste("Clustering in",k,"clusters"))
k <- as.integer(k)
```

Warning: the readline function will read a string and we must therefore transform k in an integer value. (Reminder: the paste command allows to concatenate strings.)

1.2.4 Random Choice of the First Centroids (the Means)

According to the algorithm, we must choose (randomly) k points that represent the mean position (the centroids) of the initial partitions $m_1^{(1)}, \ldots, m_k^{(1)}$. In ℝ, there are several functions to generate random values. The sample function is able to return a sample from a larger population, –i.e., a defined number of random integers from a set, with or without replacement.

```
random_points <- sample(1:dim, k, replace=F)
means <- matrix(nrow=k, ncol=n_c)
for (i in 1:k)
  {
  for (j in 1:n_c) { means[i,j] <- mydata_zs[random_points[i],j] }
  }
```

1.2.5 Variable Declaration and Definition

We will need different variables. We'll have to keep the partition performed by the algorithm in different clusters (variable clusters) but we will also take account of previous partition (prev_clusters) to verify that the algorithm is able to converge. These two variables are vectors with integer values (the cluster number of each individual). It will also be necessary to use two points to calculate their distances, i.e., one of the k points (a centroid) and all points in the dataset. These two points will have as much coordinates as variables. Moreover, it will be interesting to count the number of times the algorithm will loop before reaching the convergence (variable nb_loops) and a Boolean variable for indicating whether the convergence is still ongoing (and therefore it is still needed to run the algorithm) or if it is able to converge (and here we must stop the algorithm).

```r
clusters <- matrix(nrow=n_r, ncol=1, 0)
prev_clusters <- matrix(nrow=n_r, ncol=1, 0)
A <- matrix(nrow=n_c, ncol=1, 0)
B <- matrix(nrow=n_c, ncol=1, 0)
nb_loops <- 0
convergence_in_progress <- TRUE
```

1.2.6 Programming a Distance Function

We will define a distance function to find out what is the nearest mean point (centroid) of any point in the dataset. The choice of the distance function is crucial in the final partitioning obtained by the clustering method. The distance that we program here is simply the Euclidean distance.

```r
d <- function(X, Y) {
  distance <- 0
  for (z in 1:n_c) { distance <- distance + (X[z,1] - Y[z,1])^2 }
  return(sqrt(distance)) }
```

1.2.7 Graphical Representation

Within the loop, after reaching a partitioning, we represent the different individuals of the dataset with a different color depending on the cluster. The most interesting thing will be to represent the data by using the length and the width of the petals for x- and y-axes. On the same graph, we also represent the mean point (as blue square). It will be necessary to use the instruction par(new = TRUE) for not deleting the previous graphic, and not adding caption for the x- or y-axes (xlab and ylab will by assigned by an empty string). Moreover, we will not change the graph scale by keeping the scale of the dataset previously represented.

```r
x <- 1:dim
plot(mydata_zs[x,3], mydata_zs[x,4], col=clusters[x,1])
par(new = TRUE) # On the same graph,
y <- 1:k        # the "k" means (in blue squares)
plot(means[y,3], means[y,4], col="blue",
     xlab="", ylab="",
     xlim=range(mydata_zs[x,3]),
     ylim=range(mydata_zs[x,4]), pch=15)
```

1.2.8   End of the Program

You still have to complete the program with a loop while the convergence is still in progress.

- For each individual,

  - for each variable,

    * build point A...

  - For each centroid,

    * for each variable,

      · build point B...

    * Compute the distance between A and B

  - Initialization: the minimal distance will be infinite and the cluster number will be zero

  - For each centroid,

    * if the current distance is smaller, then change the number of cluster and the minimal distance

  - The number of cluster found will be added to the variable clusters

- The number of loops is incremented

- To verify that the convergence is not achieved, we have to test whether the current clusters and those obtained before are identical (i.e., for each individual, if they don't differ from one element, they are identical)

- The convergence must still be done if the identity is not found

- The graphic representation described above (subsection 1.2.7) has to be performed

- We print that the current result was obtained after nb_loops repetitions.

- If the convergence is still in progress, we compute the new centroids (mean points) for each cluster.

2 Exercises

1. Copy the program and modify it for taking into account only the 3rd and 4th variable of iris dataset. Run the program to see the differences.

2.
   - Copy the program and modify it for running the program with ruspini.txt dataset. The file can be downloaded from *Claroline* platform.

   - For reading a textfile (e.g., a file stored on "R/data" repository), we have to write:

     ```
     mydata <- read.table("~/R/data/ruspini.txt", quote="\"")
     ```

   - According to you, just by seeing a simple graphical representation of Ruspini dataset, how many clusters can be found in the dataset?

   - Run the program with this number for k, and smaller and greater values for k to see the stability of the results.

3 Recommended Readings

The main literature for this section is:

- Adler (2010), "R in a Nutshell – a Desktop Quick Reference,"
  Chapter 22 "Machine Learning."

- Baayen (2008), "Analyzing Linguistic Data: A Practical Introduction to Statistics using R,"
  Chapter 5 "Clustering and classification."

- Cook and Swayne (2007), "Interactive and Dynamic Graphics for Data Analysis: With R and GGobi,"
  Chapter 5 "Cluster Analysis."

- Everitt and Hothorn (2010), "A Handbook of Statistical Analyses Using R,"
  Chapter 9 "Recursive Partitioning" and Chapter 18 "Cluster Analysis."

- Everitt and Hothorn (2011), "An Introduction to Applied Multivariate Analysis with R,"
  Chapter 6 "Cluster Analysis."

- Husson et al. (2010), "Exploratory Multivariate Analysis by Example Using R,"
  Chapter 4 "Clustering."

- James et al. (2013), "An Introduction to Statistical Learning: with Applications in R,"
  Chapter 10 "Unsupervised Learning."

- Ledolter (2013), "Data Mining and Business Analytics with R,"
  Chapter 15 "Clustering."

- Wehrens (2011), "Chemometrics with R: Multivariate Data Analysis in the Natural Sciences,"
  Chapter 5 "Self-Organizing Maps" and Chapter 6 "Clustering."

References

Adler, J. (2010). *R in a Nutshell – a Desktop Quick Reference*. O'Reilly.

Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics using R*. Cambridge University Press.

Cook, D. and D. F. Swayne (2007). *Interactive and Dynamic Graphics for Data Analysis: With R and GGobi*. Use R! Springer.

Everitt, B. and T. Hothorn (2011). *An Introduction to Applied Multivariate Analysis with R*. Use R! Springer.

Everitt, B. S. and T. Hothorn (2010). *A Handbook of Statistical Analyses Using R* (2nd ed.). Chapman & Hall / CRC.

Husson, F., S. Lê, and J. Pagès (2010). *Exploratory Multivariate Analysis by Example Using R*. Computer Science & Data Analysis. Chapman & Hall / CRC.

James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

Ledolter, J. (2013). *Data Mining and Business Analytics with R*. John Wiley & Sons, Ltd.

Wehrens, R. (2011). *Chemometrics with R: Multivariate Data Analysis in the Natural Sciences*. Use R! Springer.