Implementation of Clustering Analysis

U97019176 Jen-Yin Chao

U52101803 Meng-Tse Li

Graduate Students,

BU MET Collage Computer Science Department,

CS566 spring 2018 – Alexander Belyaev

Table of Contents

Abstract

Cluster is an essential method of data analysis, which can be used in customer analysis, market basket analysis, and most cases of segmentation without clear boundary of classification. Accordingly, there have been several available packages in R and Python, but they can only be used one of the method at a time, while it's more useful and accurate to combine hierarchical and non-hierarchical cluster analysis, that is to say, the setting of group number of K-means method(non-hierarchical) can be fetched from the outcome of Ward's method(hierarchical). Furthermore, when the scale of data set becomes larger, most of packages will be quite time-consuming. As the result, our goal is to implement these two cluster methods with algorithm theories to improve efficiency and compute more accurate result. However, after the implementation, we cannot compete the execution time, although we have similar clustering outcome.

Introduction

Classification and Clustering are two types of learning methods which characterize objects into groups by one or more features. These processes appear to be similar, but there is a difference between them in context of data mining [1]. The prior difference between classification and clustering is that classification is used in supervised learning technique where number of groups is known and discriminant analysis derives a rule for allocating new observations whereas clustering is used in unsupervised learning where number of groups is unknown and cluster analysis forms groups on the basis of similarities or distances.

In this project, we are focusing on Clustering. Cluster analysis is used to solve some difficult tasks such as marketing which help marketers discover distinct group in their customer bases, and then use this knowledge to develop targeted marketing program, advertisement, or insurance which help identifying groups of motor insurance policy holders with a high average claim cost [2].

In additionally, there are two types of clustering:

Hierarchical clustering

By using hierarchical clustering, we first find the points and calculate distances between points among all points. After having all distances between points, we are resulted in two points with the shortest distances. Then we combine these two points as a group and update the graph. We run this algorithm until there is only two points(groups) left in the graph.

There are three ways to calculate the distances:

1. **Euclidean distance**: $\mathrm{D}(P_1, P_2) = \sqrt{(x_{P_1} - x_{P_2})^2 + (y_{P_1} - y_{P_2})^2}$
2. **Square Euclidean distance:** $\mathrm{D}(P_1, P_2) = (x_{P_1} - x_{P_2})^2 + (y_{P_1} - y_{P_2})^2$
3. **Manhattan or city-block distance:** $\mathrm{D}(P_1, P_2) = |x_{P_1} - x_{P_2}| + |y_{P_1} - y_{P_2}|$

In addition, there are four methods to choose the points to represent the group:

1. **Single linkage method:**
   In single linkage hierarchical clustering, also called nearest neighbor, the distance between two clusters is defined as the *shortest* distance between two points in each cluster [3].
2. **Complete linkage method:**
   In complete linkage hierarchical clustering, also called farthest neighbor, the distance between two clusters is defined as the *longest* distance between two points in each cluster [3].
3. **Average linkage method:**
   In average linkage hierarchical clustering, the distance between two clusters is defined as the *average* distance between each point in one cluster to every point in the other cluster [3].
4. **Ward's method:**
   It pretends to merge two clusters and then estimate a centroid for the resulting cluster. After estimating the centroid, calculate the sum of the squared deviations of all the points from the new centroid. For different merging pairs will have different deviations and then pick the merge that results in the smallest deviation from the new central.

Non-Hierarchical clustering

The most popular method is k-means. First, we separate the points into several groups and calculate each group's gravity central point. Then we compute the distances between every points with every gravities. After having all distances, we find the points which are the farthest from their own group's gravities and randomly choose a point to reassign to the closer group. Then we keep iterating the algorithm which reassigns the point until the graph is stable; i.e., every point has the shortest distance with its group gravity central point.

Objective

The objectives are the following:

1. Implement non-hierarchical clustering with average of Manhattan distance by using R.
2. Implement hierarchical clustering with K-Means by using R.

3. Code optimization for both algorithms to reduce run-time.
4. Compare run-time, table, and plot results between our algorithms and R cluster package.

Approach

To approach the problem, R is the software used in this project, and we combine all the code in a main script as shown in fig 1, includes our self-designed hierarchical algorithm, nonhierarchical algorithm, clustering package, and k-means package.

```
library(datasets) import pk for data import
library(ggplot2) import pk for outcome plotting
source("Hierarchical.R") import self-designed function "hier"
source("Nonhierarchical.R") import self-designed function "non_hier"
#Clustering with self-designed algorithm
df_dier <- hier(data, normalize, groupNum=3)
df_nondier <- non_hier(data, normalize, groupNum=3)
#Clustering with package
library(cluster) import pk for hierarchical clustering
d <- dist(data, method="manhattan") compute the distances between datasets
fit <- hclust(d, method="average") hierarchical clustering
groups <- cutree(fit, groupNum=3) get the group outcome of clustering
iris.kmeans <- kmeans(data, groupNum=3) k-means clustering
```

Figure 1. Pseudo Code of Main.R

Hierarchical Algorithm

1. In the outer loop, we merge each closet point until they are all in the same group.
2. In the for loop, we compute the distances between points, store in a distance matrix, and then find the minima value in it.
3. Depending on the minima value, we can find the index of those points, and use the self-defined function, newP1, to merge them.
4. During the loop, by setting the parameter groupNum or not, user can get either the clustering outcome with specific group number or the automatic clustering output.
5. After the main outer for loop finished, the last for loop will update the clustering result to the data frame, and then return it to main script.

```
hier <- function(df, normalize=TRUE, groupNum=0){  #define function
    newP1 <- function(p1,p2){ #define local function
        compute the center of mass btw p1 & p2
        concatenate the index of p1 & p2
        sum up the weight of p1 & p2
        return (all the above information)
    }
    create df_temp to iteratively manipulate combination
    for (j from 1 to row # -1 of df) {  #keep merging until all dataset are in the same group
        for (i from 1 to row # of df_temp) {
            calculate the distance btw groups and store in a distance matrix
        }
        find the min value in the distance matrix
        if (groupNum is not set) {  #If groupNum is not set, get the outcome automatically
            if (min distance is larger than the stored min distance) {
                update stored min distance by current min distance
                update stored df_temp by current df_temp
            }
        } else if (row # of df_temp = groupNum) {
            update stored df_temp by current df_temp
        }
        find the index of the closest two points
        update point1 with newP1()
        delete point2
    }
    for (k from 1 to row # of stored df_temp) {
        assign the group # from stored df_temp to df  #from the concatenated index }
    return (df) }
```

Figure 2. Pseudo Code of Hierarchical.R

Non-hierarchical Algorithm

1. At the beginning, users can choose whether they want to normalize the data or not.
2. By using self-defined function, group_init, it can randomly assign all the points to several groups.
3. In the while loop, it will keep checking until all the points are stable.
4. Inside the while loop, it will compute the distance between points and the centers of groups, by using self-defined function distance_Ele2Grp.
5. Depending on the matrix of distance from previous step, we can check which point is not closet to the center of its own group and store their index in the unstable list.
6. So, if the unstable list is not empty, at the beginning of the while loop, the if function will randomly choose one of the unstable point to reassign it to its closet group.

```
library(dplyr) import pk for easier coding formula
source("normalize.R") import self-designed function "normalize"
source("group_init.R") import self-designed function "group_init"
source("distance_Ele2Grp.R") import self-designed function "distance_Ele2Grp"
non_hier <- function(df, normalize=TRUE, groupNum){  #define function
    if (normalize) { df <- normalize(df) }
    randomly assign point to different groups by group_init()
    initial unstable_list = 0
    while (length of unstable_list != 0) {  #keep reassign until all points are stable
        if (unstable_list != 0){
            choose one of unstable point and reassign to the most appropriate group
        }
        compute distances btw points & groups by distance_Ele2Grp()
        update unstable_list by checking the matrix from the above line
    }
}
```

Figure 3. Pseudo Code of Non-hierarchical.R

distance_Ele2Grp.R

1. First, it computes the mean, the center, of each group.
2. In the for loop, by using matrix manipulation, it computes the distance between each points and groups with complexity of the number of group.

```
distance_Ele2Grp <- function(df, groupNum) {  #define function
    compute the means of variables of each group
    for (i from 1 to groupNum) {
        expand the array of means by repeating as a matrix
        use matrix manipulation to compute the distance to ith group of each dataset }
    return (distances matrix btw datasets to the center of each group) }
```

Figure 4. Pseudo Code of distance_Ele2Grp.R

Comparison

We have coded two algorithms which one is *hierarchical* clustering with *Manhattan distance* and *average linkage method* and another is *non-hierarchical* clustering with *k-means*. Then we use the same data to run both our algorithms and R's packages to compare. The result is similar yet not quite the same in three different categories:

Efficiency:

```
> #Self designed Algo
> source("Hierarchical.R")
> source("Nonhierarchical.R")
> #Hier
> ptm <- proc.time()
> df_hier <- hier(iris, normalize = FALSE, groupNum)
> proc.time() - ptm
   user  system elapsed
  8.170   0.055   8.240
> #Non-hier
> ptm <- proc.time()
> df_nonhier <- non_hier(iris, groupNum, normalize = FALSE)
> proc.time() - ptm
   user  system elapsed
  0.922   0.007   0.963
>
> #Package
> library(cluster)
> #Hier
> ptm <- proc.time()
> d <- dist(iris[,1:4], method = "manhattan") # distance matrix
> fit <- hclust(d, method="average")
> groups <- as.factor(cutree(fit, k=3)) # cut tree into 3 clusters
> proc.time() - ptm
   user  system elapsed
  0.003   0.001   0.003
> plot(fit)
> rect.hclust(fit, k=3, border="red")
> #Non-hier
> ptm <- proc.time()
> iris.kmeans <- kmeans(iris[1:4],3)
> proc.time() - ptm
   user  system elapsed
  0.001   0.000   0.001
```

Figure 5. Runtime of each algorithm

As we can see from the screenshot of the computing result, the first two are our self-implement hierarchical and non-hierarchical algorithms with 8.240 seconds and 0.963 seconds. The last two are R packages with 0.003 seconds and 0.001 seconds. In our opinion, our algorithms are good but can be improved. For example, the hierarchical algorithm can use only half the matrix or only update the distances of the specific group to reduce the run time.

Table of Clustering:

```
> #table
> table(df_hier$Species, df_hier$Group)   #self hier

             1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0  6 44
> table(df_nonhier$Species,df_nonhier$group)    #self non-hier

             1  2  3
  setosa      0  0 50
  versicolor  3 47  0
  virginica  36 14  0
>
> table(iris$Species,groups)    #cluster package
            groups
             1  2  3
  setosa     50  0  0
  versicolor  0 49  1
  virginica   0 14 36
> table(iris$Species,iris.kmeans$cluster)    #kmeans package

             1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0 14 36
```

Figure 6. Clustering outcome table of each algorithm

As we can see from the screenshot, the first two are results of self-implement tables with our hierarchical and non-hierarchical algorithms. The last two are results of R's cluster package and R's k-means package. There are only minor differences between our tables and the tables generated by R's package, therefore we can conclude that our algorithms are accurate.

Plot:

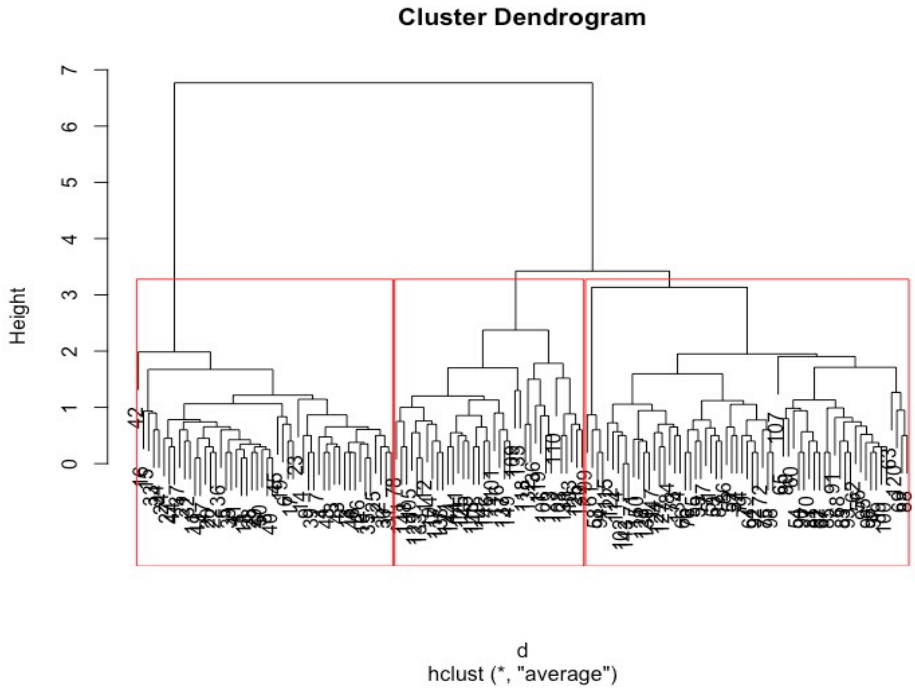

Figure 7. Cluster dendrogram plot from cluster package

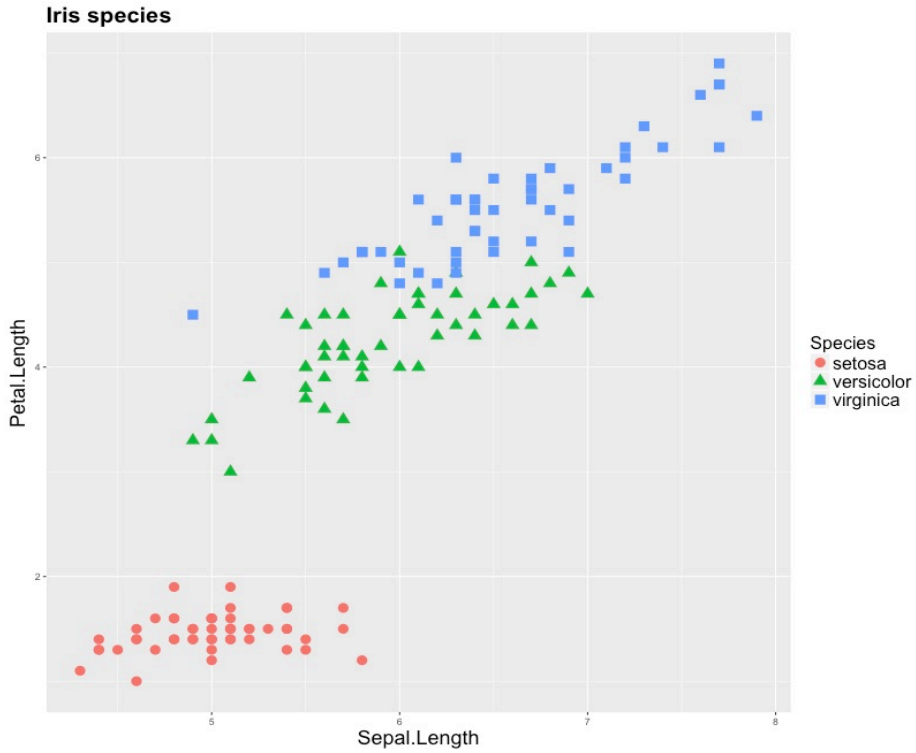

Figure 8. Iris species dot plot

In figure 7, it shows the process of merging by cluster package, and we can easily find out that it's hard to get the outcome with 3 groups because of the little difference while merging three groups into two.

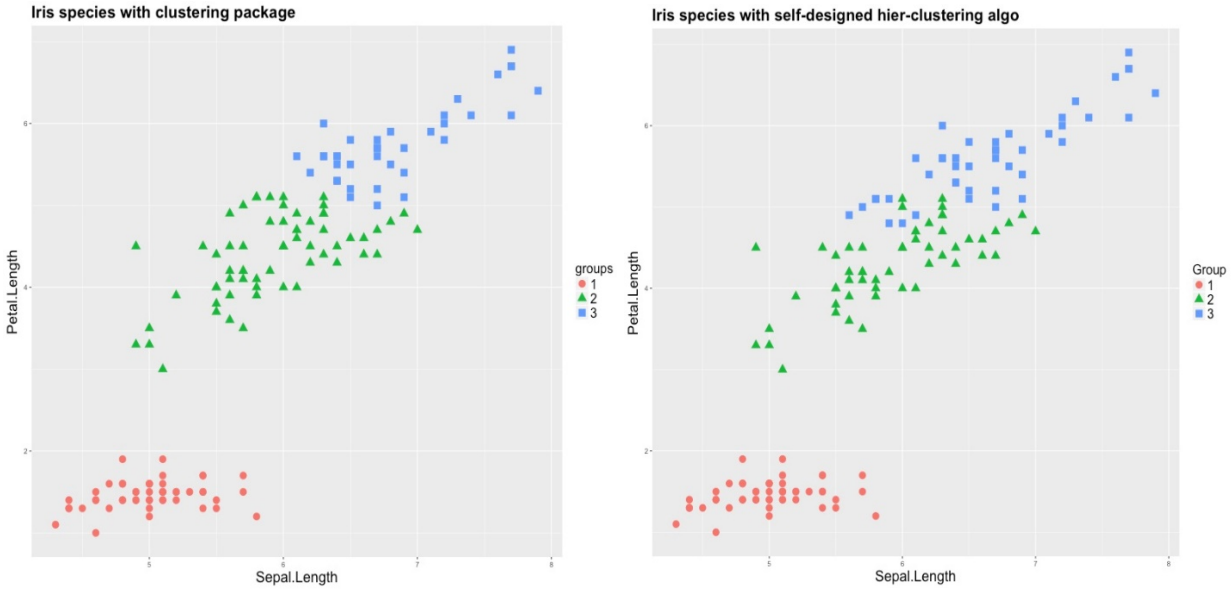

Figure 9. Hierarchical clustering dot plot

Fig 8 is the dot plot of iris species, and fig 9 is the hierarchical cluster outcome from the package and our self-designed algorithm, which are quite similar to the original one.

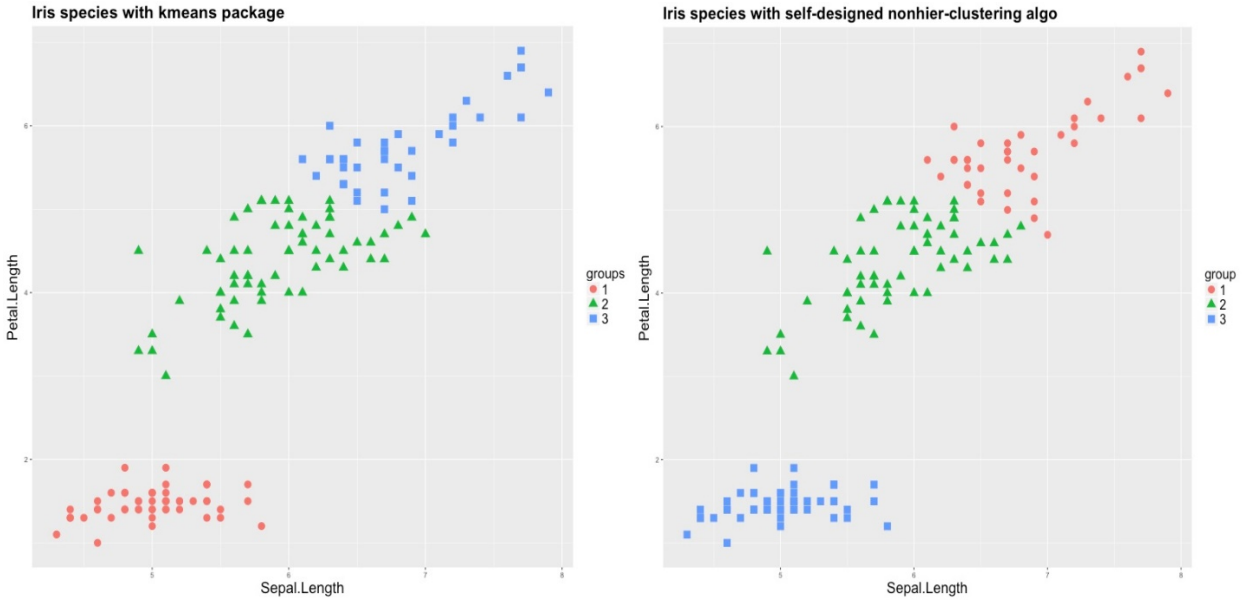

Figure 10. Non-hierarchical clustering dot plot

Fig 10 is the comparison of outcome between k-means package and the self-designed non-hierarchical algorithms, which are both quite similar too, representing our algorithm comes up with right clustering.

Conclusion

Clustering analysis is indeed a powerful method of data analysis. It can be used in marketing analysis, customer analysis, and most cases of segmentation without clear boundary of classification. However, the current clustering analysis function in R is limited. If we can make a comprehensive cluster analysis with all hierarchical and non-hierarchical methods, I believe this clustering function will be more powerful and broadly used at this modern world.

References

[1] "Difference Between Classification and Clustering (with Comparison Chart)." *Tech Differences*, 2 Jan. 2018, https://techdifferences.com/difference-between-classification-and-clustering.html

[2] "What is Cluster Analysis?", http://www.stat.columbia.edu/~madigan/W2025/notes/clustering.pdf

[3] Sayad, Saed. "Hierarchical Clustering." *Hierarchical*, www.saedsayad.com/clustering_hierarchical.htm.