Lab 2: Cluster Analysis Part I

1 Objectives

- Dissimilarity Measures
- K-Means
- PAM
- Hierarchical Clustering

2 Dissimilarity Measures

In the lectures we introduced a few distances to measure the dissimilarity between two samples (or variables). In R those distances can actually be calculated by a simple function. Here is an example below.

```
# Import the dataset
data.new <- read.table("~/Box Sync/ST592-Winter-2019/Labs/Week 2/data_new.txt",
                       header = FALSE)

# Calculate the Manhattan distance between the first two variables using the dist function
dist(t(data.new[, c(1, 2)]), method = "manhattan")

# Calculate the Manhattan distance between the first two variables using formula
sum(abs(data.new[, 1] - data.new[, 2]))
```

The **dist** function is a common function to calculate multiple types of distance between two or more variables. There are few things you might need to notice for this function.

- The first input in the **dist** function is usually a matrix or a data frame. It will automatically calculate the distance between each pair of rows in the matrix/data frame. Thus, if the distances between columns are needed, you need to add **t()** function to transpose the matrix.
- There are several types of distance that the **dist** function can calculate. You can try the **help** function or question mark to see the detail.

```
# The two functions below will give you the same result
help(dist)
?dist

# Calculate the Euclidean distance using the dist function
dist(t(data.new[, c(1, 2)]), method = "euclidean")

# Calculate the Euclidean distance using formula
sqrt(sum((data.new[, 1] - data.new[, 2])^2))
```

The **dist** function can also calculate the distance for asymmetric binary variables.

```
# Generate a data frame with two rows of binary data
x <- matrix(0, nrow = 2, ncol = 20)
for (i in 1 : 2) {
  for (j in 1 : 20) {
    x[i, j] <- rbinom(1, 1, 0.5)
  }
}
```

```r
# Calculate the distance by the dist function
dist(x, method = "binary")

# Calculate the distance by formula
x.table <- table(x[1, ], x[2, ])
(x.table[1, 2] + x.table[2, 1])/(x.table[1, 2] + x.table[2, 1] + x.table[2, 2])
```

The **dist** function can deal with multiple variables instead of just two. When we have a data frame with more than two rows, it will automatically return a distance matrix.

```r
dist.eu <- dist(t(data.new), method = "euclidean")
```

The **dist** function will store the output distance matrix as class "dist". For more information see the **Value** section in *help(dist)*.

3 Cluster Analysis

3.1 K-means

The function of k-means clustering is **kmeans**. Here is an example.

```r
# Create two groups of points from two different populations
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
colnames(x) <- c("x", "y")

# Visualize the data
plot(x)

# Use K-means clustering to divide the whole group into two clusters
cl <- kmeans(x, 2)

# Visualize the clusters
plot(x, col = cl$cluster)
points(cl$centers, col = 1 : 2, pch = 8, cex = 2)
```

In the **kmeans** function, there are two input options that you can modify.

- The first input is your data set, the rows of which are treated as data objects to be clustered;
- The second input is **centers**, which will specify how many clusters you want to be divided from the whole group. The **centers** input can be either a number or a set of initial cluster centers. If it is a number, then it represents the number of clusters you need; if it is a set of initial cluster centers, then the number of clusters will be the number of initial centers you specify, and the start points in k-means algorithm will be those centers;

The output of **kmeans** function contains lots of information. Here I list a few important ones.

- cluster: this output contains a vector of integers, which indicates which cluster each point is allocated;
- centers: this output contains all the cluster centers for the final clustering;
- size: this output shows the number of points in each cluster.

For more imformation of **kmeans** function, you can use help function to see those details.

```r
?kmeans
```

Next, we will apply k-means to a real genetic dataset. The dataset we use next is a gene expression data that contains 47 samples, and 100 genes per sample. The 47 samples belong to two groups (different types of leukemia), which are included in the file "data_type.txt". These two groups will serve as the truth for us to evaluate the cluster analysis results.

```r
# Import the data set
data.new <- read.table("~/Box Sync/ST592-Winter-2019/Labs/Week 2/data_new.txt",
                       header = FALSE)
data.type <- read.table("~/Box Sync/ST592-Winter-2019/Labs/Week 2/data_type.txt",
                        header = FALSE)
data.type
# Transform the type into "1" and "2" (to compare with clustering result)
data.type <- data.type + 1

# K-means with 2 clusters
fit.kmeans <- kmeans(data.new, centers = 2)

# "str" and "summary" are powerful functions to check results
str(fit.kmeans)
cluster.kmeans <- fit.kmeans$cluster

# Clustering result versus truth
cbind(cluster.kmeans, data.type)

# Number of correctly and incorrectly clustered objects
sum(cluster.kmeans == data.type)
sum(cluster.kmeans != data.type)
```

One thing you need to notice for the **kmeans** function is that the labeling from its output might be different from the true clustering labeling. Since there are only two groups in the population, you can switch the labeling from k-means clustering and choose the one that better agrees with the true labeling.

3.2 PAM

In order to use PAM to conduct cluster analysis, we need to install package "cluster" first. Actually many clustering algorithms are included in this package. Let's take the same gene expression data as an example.

```r
# Install the package (only need to do it once)
install.packages("cluster")

# Require package "cluster"
library(cluster)

# Clustering by PAM
fit.pam <- pam(data.new, k = 2, metric = "euclidean")

# Clustering result
str(fit.pam)
cluster.pam <- fit.pam$clustering

# Number of correctly and incorrectly clustered objects
sum(cluster.pam == data.type)
sum(cluster.pam != data.type)
```

According to this example, we can see that the function **pam** has similar input options as the function **kmean**. However, there is one difference. In the **pam** function there is an option called *metric*, which has two possible values, “euclidean” and “manhattan”. This will specify the metric (distance) to be used for calculating dissimilarities between objects. Try the previous example with the manhattan distance.

3.3 Hierarchical clustering

In the lecture we introduced two methods of hierachical clustering. During the lab we will introduce the function **hclust**, which will use agglomerative method to conduct the hierachical clustering. Take the same data again as an example.

```r
# Calculate the euclidean distance matrix of this data set
dist.new <- dist(data.new, method = "euclidean")

# Apply hierachical clustering to this distance matrix
fit.hclust <- hclust(dist.new, method = "complete")

# Based on the clustering result, find two clusters
cluster.hclust <- cutree(fit.hclust, k = 2)

# Number of correctly and incorrectly clustered objects
sum(cluster.hclust == data.type)
sum(cluster.hclust != data.type)
```

The structure for the function **hclust** is very different from **kmean** or **pam**.

- The first input of **hclust** is no longer the original data set, it should be a distance matrix based on the original data set. Thus, we need to calculate a distance matrix based on any metric we want before using **hclust** function.
- The output of **hclust** will not give a clustering directly. Instead, it will give a hierachical clustering structure (also called dendrogram, which will be covered in the next week). Thus, if the number of clusters is known, we need to use the **cutree** function to “cut” the dendrogram into that number of clusters.
- As we learned from the lectures, there are several inter-cluster distance types. We cam modify that by the option *method* in **hclust** function. For further information, go to the help file to see the details.