Seminar 9

Clustering

Overview

In this tutorial document, we will learn to make cluster analyses with R. To conduct k-means clustering and hierarchical clustering, the *stats* and *cluster* packages are required respectively. Let's install and activate the packages to complete the exercises in this tutorial.

> install.packages("stats"); install.packages("cluster")

> library(stats); library(cluster)

Note: the two packages are usually pre-installed in R. See if the packages were already installed on your R using ".packages(all.available = TRUE)".

Data

We will use the wine dataset from the *rattle* package. This dataset contains 13 chemical measurements on 178 Italian wine samples. The data originally come from the UCI machine Learning Repository (http://www.ics.uci.edu/~mlearn/MLRepository.html). This exercise attempts to cluster the wines based on the 13 chemicals using k-means clustering and hierarchical clustering. You don't need to import any data from you project folder for this exercise. Let's install the *rattle* package and load the wine data.

> install.packages('rattle')

> data(wine, package='rattle')

You can find the descriptions of the variables from the *rattle* package.

> ??wine

The data has a variable "*Type*" to separate the training set and testing set. We do not need the *Type* variable for this exercise. Delete the *Type* variable from the data and rename the data as 'WineData'

```
## ! Delete one column ##
WineData <- wine[, !(colnames(wine) == 'Type') ] #or WineData <- wine[-1]
```

Since the variables vary in range, they are standardized prior to clustering. To standardize the variables, we use the scale function.

> WineData_Stand <- scale(WineData) #standardize the variables

Please check out the variables and their values before/after the standardization!

> head(WineData); head(WineData_Stand)

- Before standardizing the variables

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
2	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
3	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
4	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
5	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
6	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450

- After standardizing the variables

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
[1,]	1.5143408	-0.56066822	0.2313998	-1.1663032	1.90852151	0.8067217	1.0319081	-0.6577078	1.2214385	0.2510088	0.3611585	1.8427215	1.01015939
[2,]	0.2455968	-0.49800856	-0.8256672	-2.4838405	0.01809398	0.5670481	0.7315653	-0.8184106	-0.5431887	-0.2924962	0.4049085	1.1103172	0.96252635
[3,]	0.1963252	0.02117152	1.1062139	-0.2679823	0.08810981	0.8067217	1.2121137	-0.4970050	2.1299594	0.2682629	0.3174085	0.7863692	1.39122370
[4,]	1.6867914	-0.34583508	0.4865539	-0.8069748	0.92829983	2.4844372	1.4623994	-0.9791134	1.0292513	1.1827317	-0.4263410	1.1807407	2.32800680
[5,]	0.2948684	0.22705328	1.8352256	0.4506745	1.27837900	0.8067217	0.6614853	0.2261576	0.4002753	-0.3183774	0.3611585	0.4483365	-0.03776747
[6,]	1.4773871	-0.51591132	0.3043010	-1.2860793	0.85828399	1.5576991	1.3622851	-0.1755994	0.6623487	0.7298108	0.4049085	0.3356589	2.23274072

k-means Clustering

The k-means cluster analysis is the most common partitioning method. Below is the k-means algorithm:

1. Determine the number of k (i.e., number of clusters)
2. Randomly select K centroids (i.e. centers of clusters)
3. Assign each data point to its closest centroid
4. Recalculate the centroids as the average of all data points in a cluster
5. Assigns data points to their closest centroids
6. Repeat Step 4 and 5 until the observations are not reassigned or the maximum number of iterations (R uses 10 as a default) is reached

In our dataset, we observe the composition of different wines. Given a set of observations (x_1, x_2, x_3, …,x_n), k-means clustering aims to partition the *n* observations into *k* groups, so as to minimize the within-groups sum of squares (WSS). The below is the objective function of WSS that we want to minimize:

$$\text{WSS}(k) = \sum_{i=1}^{n} \sum_{j=0}^{p} (x_{ij} - mean(x_{kj}))^2$$

where, *k* is the cluster, x_{ij} is the value of the j^{th} variable for the i^{th} observation, and $mean(x_{kj})$ is the mean of the j^{th} variable for the k^{th} cluster.

The clustering optimization problem is solved with the *kmeans* function in R. The syntax of the *kmeans* function is *kmeans*(*x*, *#centers*) where *x* is a numeric dataset (i.e., matrix or data frame) and *#centers* is the number of clusters to extract. The function returns the cluster memberships, centroids, sum of squares (within, between, and total), and cluster sizes.

Let's try with 3 clusters!

```
## Run a k-means clusering with 3 clusters ##
set.seed(2406)
Wine_KM <- kmeans(WineData_Stand, 3)
```

As k-means cluster analysis starts with *k* randomly chosen centroids, a different solution can be obtained each time the function is involved. Use the *set.seed()* function to make the results reproducible.

In the outcome of *kmeans* function (i.e., Wine_KM), all the elements/attributes of the cluster output are contained:

```
> attributes(Wine_KM)
$names
[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"

$class
[1] "kmeans"
```

Let's return the cluster sizes (i.e., number of observations for each cluster)

```
> Wine_KM$size
[1] 65 51 62
```

Then, we can find the centers of clusters (i.e., centroids for the variables):

```
> Wine_KM$centers
    Alcohol      Malic       Ash Alcalinity  Magnesium    Phenols  Flavanoids Nonflavanoids Proanthocyanins      Color       Hue  Dilution    Proline
1 -0.9234669 -0.3929331 -0.4931257  0.1701220 -0.49032869 -0.07576891  0.02075402   -0.03343924      0.05810161 -0.8993770 0.4605046 0.2700025 -0.7517257
2  0.1644436  0.8690954  0.1863726  0.5228924 -0.07526047 -0.97657548 -1.21182921    0.72402116     -0.77751312  0.9388902 -1.1615122 -1.2887761 -0.4059428
3  0.8328826 -0.3029551  0.3636801 -0.6084749  0.57596208  0.88274724  0.97506900   -0.56050853      0.57865427  0.1705823 0.4726504 0.7770551  1.1220202
```

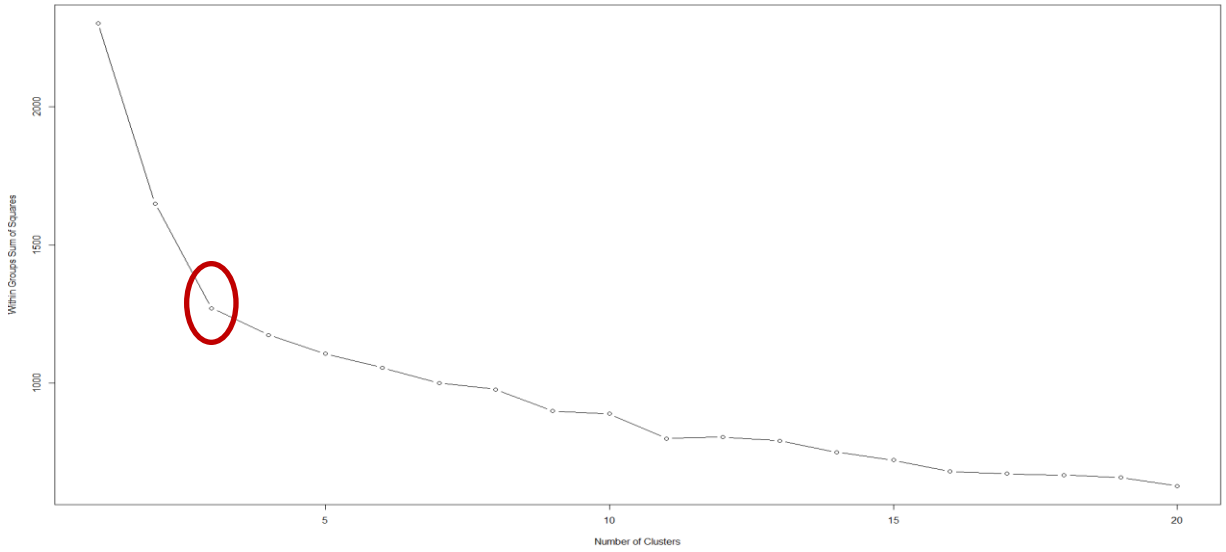
Finally, lets' return the clusters for observations.

```
> Wine_KM$cluster
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [44] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 2 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 2 1 1
 [87] 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 3 1 1 1 1 1 1 1
[130] 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[173] 2 2 2 2 2 2
```

A fundamental question is how to determine the value of *k*. Plot the within groups sums of squares (WSS) vs. the number of clusters extracted by varying the values of k from 1 to 20. WSS information is stored in the *withinss* attribute of kmeans output

- i.e., kmeans(WineData_Stand, #clusters)$withinss)

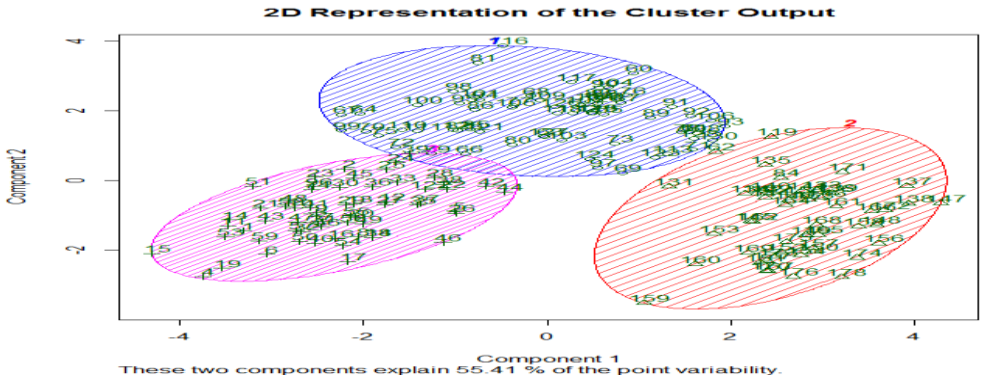
```
## Identify the optimal k ##
#Within Groups Sum of Squares#
wss <- 1:20
for(i in 1:20) {wss[i] <- sum(kmeans(WineData_Stand,i)$withinss)}
plot(1:20, wss[1:20], type="b", xlab="Number of Clusters", ylab="Within Groups Sum of Squares")
# type="b" creates a plot with lines between points #
```



The sharp decreases (elbow) from 1 to 3 clusters suggest a 3-cluster solution. You may plot the above graph first, and then determine the number of *k* for clustering.

The *clusplot* function in cluster package allows us to represent the cluster output into 2-dimensions (see more details from help(clusplot)):

```
## 2D Representation of the Cluster Solution ##
library(cluster)
clusplot(WineData_Stand, Wine_KM$cluster, main='2D Representation of the Cluster Output',
         color=TRUE, shade=TRUE, labels=2, lines=0)
```



where the Components 1 and 2 are the aggregated variables used in clustering. We are unable to identify what they are in the above graph.

Hierarchical Clustering:

Unlike k-means clustering, hierarchical clustering doesn't require us to specify the number of clusters beforehand. Below is the hierarchical clustering algorithm:

1. Put each data point in its own cluster
2. Identify the closest and combine them into one cluster
3. Repeat the above steps until all the data points are in a single cluster.

Hierarchical clustering uses a distance matrix (between records) as an input for the clustering algorithm. The choice of appropriate metric will influence the shape of the clusters, as some elements may be close to one another according to one distance and farther away according to another. In this exercise, we use the *Euclidean* distance as an input for the clustering algorithm (Then, Ward's minimum variance criterion minimizes the total within-cluster variance).

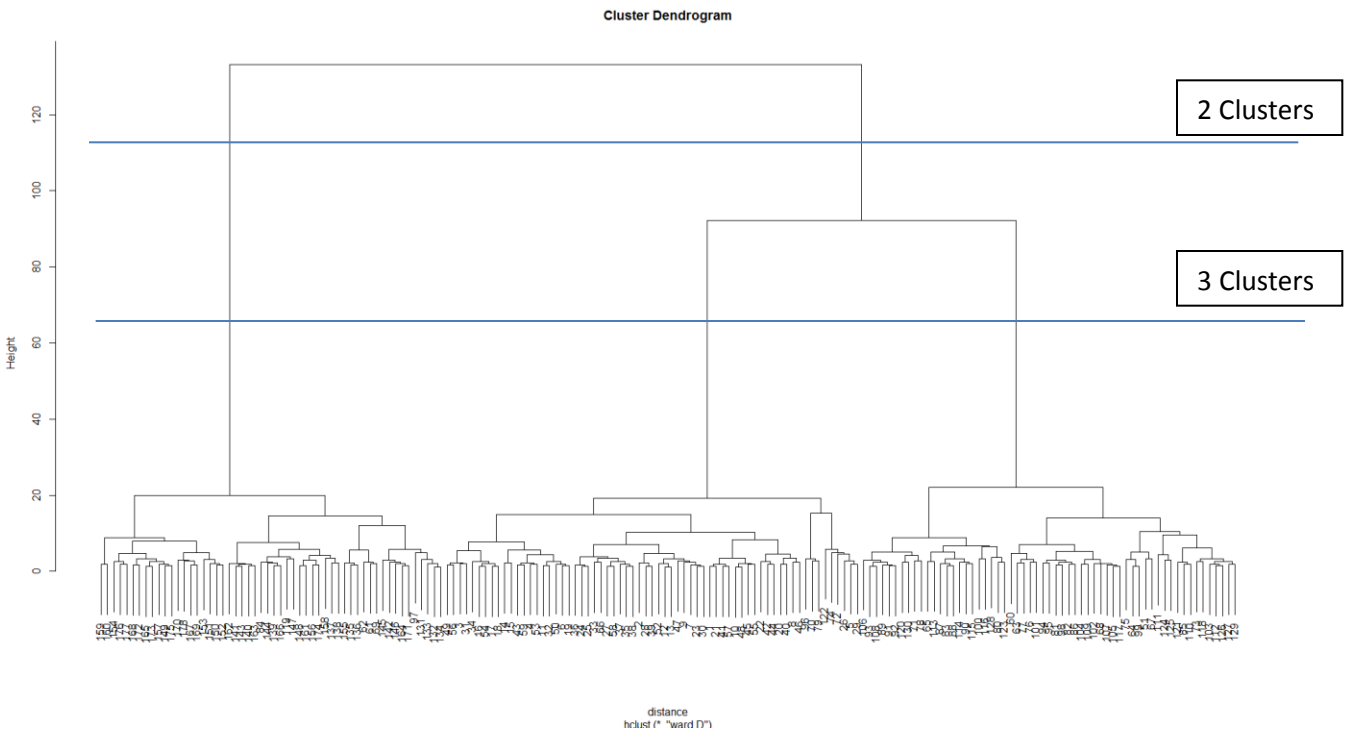
We can use the *hclust* for developing a hierarchical cluster analysis. The *hclust* function requires us to provide the data in the form of a distance matrix. We can do this by using the *dist* function.

```
distance <- dist(WineData_Stand, method = "euclidean") # Euclidean distance matrix

Wine_HC <- hclust(distance, method = "ward.D")
```

The clustering output can be displayed in a dendrogram as below:

```
plot(Wine_HC)
```



We can see from the figure the best choices for total number of clusters are either 2 or 3:

To do this, we can cut off the tree at the desired number of clusters using the *cutree* function.

```
Wine_HC_Cut <- cutree(Wine_HC, k=3) # cut tree into 3 clusters
Wine_HC_Cut
```

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 3 3 2 2 2 1 2 2 3 1 2 1 2 1
 [75] 2 2 2 2 1 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 1 3 2 2 2 2 2 2 2 2 2 2 2 2 2
[112] 2 2 2 2 2 2 2 3 2 2 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[149] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

Finally, we can see the memberships of wines with 3 clusters.

We can draw dendogram with read borders around 3 clusters using the *rect.hclust* function.

```
# Draw dendogram with red borders around the 3 clusters #
rect.hclust(Wine_HC, k=3, border="red")
```