Chapter 6 Cluster Analysis

So far we have methods to derive classification rules for data set with known membership information. This is known as supervised learning in Data Mining. Sometimes we may want to classified the data **without** membership information into clusters such as within each cluster the observations are as homogenous as possible. This is the problem of unsupervised learning. It is useful in many applications. For example, we may want to divide our customers into several segments so that we may have different strategies to promote our products.

Generally speaking, unsupervised learning is more difficult that supervised learning. Furthermore, there is a basic difficulty in unsupervised learning. There is no unique objective criterion to cluster our data. For example, in a deck of 52 playing cards, we can group the cards according to their color (red or black) or according to their suits (spades, heart, club and diamond). Or we can even group the cards according to letters or numbers (A,K,Q,J; 1-10) etc. There are two major types of methods in Cluster analysis, namely **hierarchical** and **non-hierarchical** method. Hierarchical method tries to link the two most similar observations in the data set to form a cluster, and then link up the next two most similar objects and so on. Finally it will build a tree-hierarchy structure with lines joining all the observations. However, hierarchical method may not be suitable in data mining when the number record is huge.

On the other hand, K-means clustering is a widely used non-hierarchical clustering method. It is fast and can classified large number of records into groups. Therefore it is suitable for Data Mining.

6.1 K-means clustering

In k-means clustering, we want to cluster n observations into k homogenous groups. Note that the number of clusters, k, has to be specified in advance. Here is a brief description of the algorithm:

1. Randomly chose k data points as the initial seeds.
2. For each observation, calculate the distance of this point to these k seeds, and assign this point to group j if it is closest to the j^{th} seed.
3. We compute the mean of the k clusters in (2). Then we re-compute the distance of each observation to the k cluster means. Re-assign the observation to group j^{th} group if it is closest to group j.
4. Repeat (3) until converge.

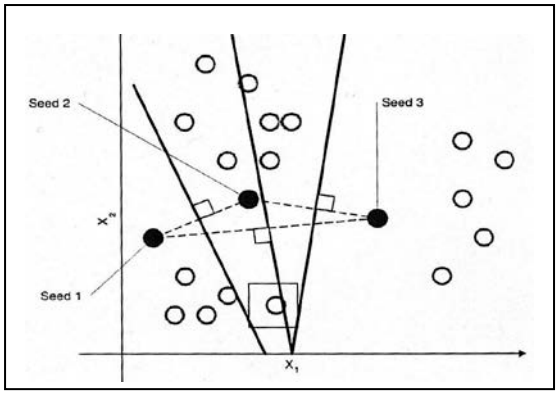


Fig. 1 Initial seeds determine the initial Cluster boundaries

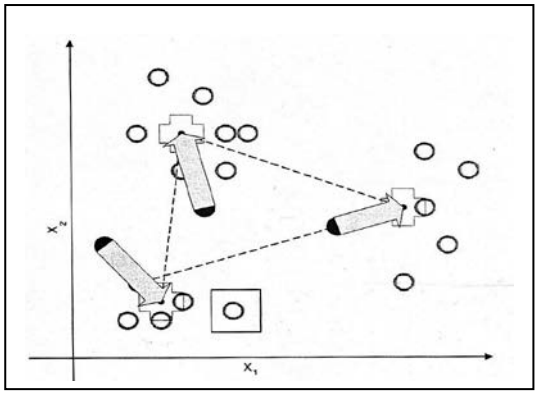


Fig. 2 Computing the centroids of the new clusters

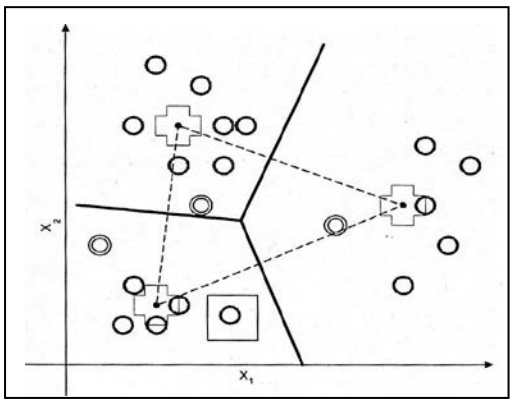


Fig 3. Re-assignment of records to new clusters

Mathematically speaking, k-means clustering is to find the class label *z* to minimize the within group sum of squares:

$$E(z)=\sum_{i=1}^{k}\sum_{j=1}^{n} z_{ij}(x_j-\bar{x}_i)^2 \text{ , where } z_{ij}=1 \text{ if } x_i \text{ is in group } j;\ z_{ij}=0 \text{ otherwise.} \qquad (6.1)$$

6.2 K-means clustering using R

The K-means clustering method is implemented in R using a built-in function *kmeans()*. Let us illustrate this by Fisher's Iris data.

```
> d<-read.csv("iris.csv")      # read in data
> x<-d[,1:4]                   # save the the first columns to x
> km<-kmeans(x,3)              # K-means clustering
> print(km)                    # print result
$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
 [33] 3 3 3 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 1 1 1
 [65] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [97] 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1
[129] 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1

$centers
         V1       V2       V3       V4
1 5.901613 2.748387 4.393548 1.433871
2 6.850000 3.073684 5.742105 2.071053
3 5.006000 3.428000 1.462000 0.246000
```

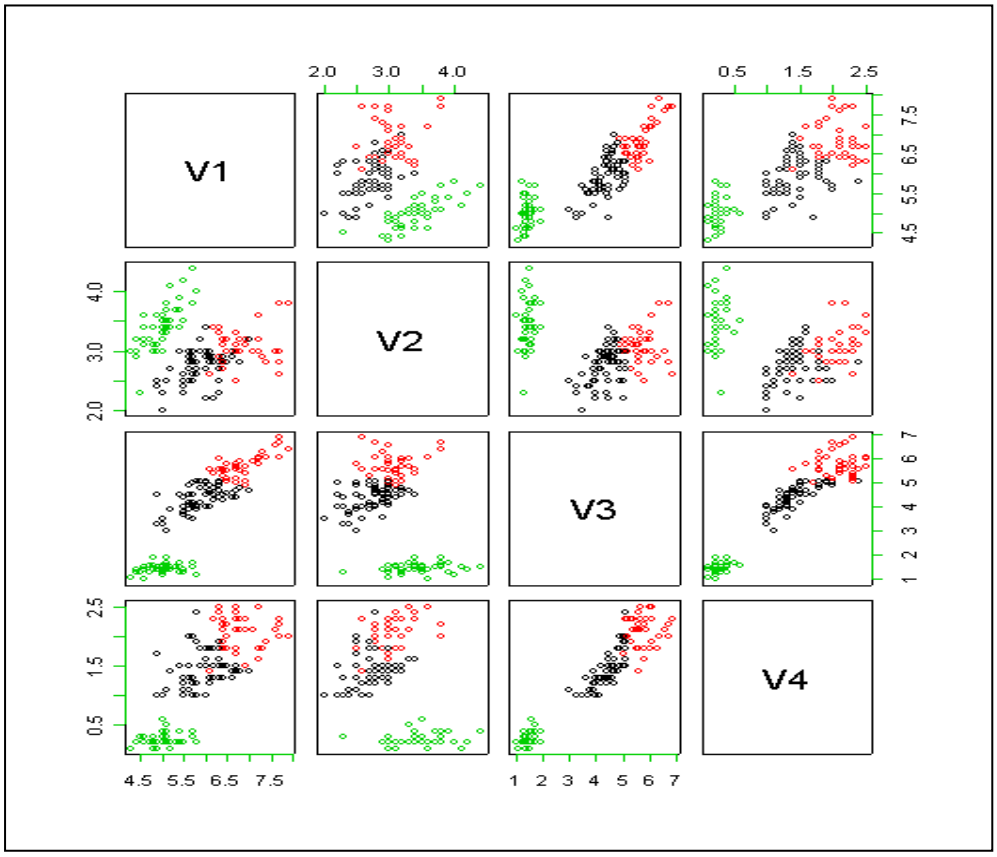
```
$withinss
[1] 39.82097 23.87947 15.15100
$size
[1] 62 38 50
```

The output of *kmeans()* contains 4 components: *cluster, centers, withinss* and *size*.

1. *Cluster* is the group label of the cluster obtained from the k-means clustering. Note that the numbering of these group labels is arbitrary. We can labeled the first group as "1" instead of "3".
2. *Centers* contains the mean vector of these groups.
3. *Withinss* is the within group sum of squares.
4. *Size* is the size of each group.

We can also plot the observation with color for each group using

```
> plot(x,col=km$cluster)
```



Finally, we produce the classification table of the label from k-means clustering with the true species.

```
> table(km$cluster,d$Species)
    1  2  3
  1  0 48 14
  2  0  2 36
  3 50  0  0
```

Note that the numbering of the label is arbitrary. Therefore the actual misclassification rate is only (14+2)/150=10.67%.

6.3 Choosing the suitable k

An obviously but difficult question is how to choose the suitable *k* in the k-means clustering. There is a useful statistic to help us choosing a suitable k for k-means clustering. Recall that our objective in clustering to find groups in the data such that the records within the same group should be as homogenous as possible while records between different groups are as different as possible. Then the within group variation and between group variation will be useful statistics. First, we need to understand the output from k-means clustering, especially the within group sum of square statistic.

```
> d<-read.csv("iris.csv")         # read in data
> x<-d[,1:4]                      # save data to x
> km<-kmeans(x,3)                 # K-means with k=3
> names(km)                       # display items in km

K-means clustering with 3 clusters of sizes 50, 62, 38
Cluster means:
  Sepal_len Sepal_wid Petal_len Petal_wid
1  5.006000  3.428000  1.462000  0.246000
2  5.901613  2.748387  4.393548  1.433871
3  6.850000  3.073684  5.742105  2.071053

Clustering vector:
  [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 1 1
 [37] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [73] 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3
[109] 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3
[145] 3 3 2 3 3 2
Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)
Available components:
[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"

> km$betweenss                    # display between group ss
[1] 602.5192
```

Let us find out how this *withinss* and *betweenss* are computed.

```
> d1<-x[km$cluster==1,]                    # select group using cluster label
> d2<-x[km$cluster==2,]
> d3<-x[km$cluster==3,]
> n1<-dim(d1)[1]                           # save group size
> n2<-dim(d2)[1]                           # n1=50, n2=62, n3=38
> n3<-dim(d3)[1]

> apply(d1,2,mean)                         # compute group mean
Sepal_len Sepal_wid Petal_len Petal_wid
    5.006     3.428     1.462     0.246
> apply(d2,2,mean)
Sepal_len Sepal_wid Petal_len Petal_wid
 5.901613  2.748387  4.393548  1.433871
> apply(d3,2,mean)
Sepal_len Sepal_wid Petal_len Petal_wid
 6.850000  3.073684  5.742105  2.071053
```

```
> (n1-1)*var(d1)                        # compute SSCP matrix for group 1
             Sepal_len Sepal_wid Petal_len Petal_wid
Sepal_len       6.0882    4.8616    0.8014    0.5062
Sepal_wid       4.8616    7.0408    0.5732    0.4556
Petal_len       0.8014    0.5732    1.4778    0.2974
Petal_wid       0.5062    0.4556    0.2974    0.5442

> sum(diag((n1-1)*var(d1)))             # compute tr(SSCP) for each group
[1] 15.151
> sum(diag((n2-1)*var(d2)))
[1] 39.82097
> sum(diag((n3-1)*var(d3)))
[1] 23.87947

> m<-apply(x,2,mean)                            # overall mean
> m<-matrix(rep(m,3),nrow=3,byrow=T)            # create a matrix whose row is m
> dm<-(km$centers-m)^2                          # (group mean - overall mean)^2
> rowsum<-apply(dm,1,sum)                       # row sum
> sum(km$size*rowsum)                           # between group ss
[1] 602.5192
```

$tr(SSW) = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(x_{ij} - \bar{x}_i)^2$ is the within group sum of square = *sum(km$withinss),*

$tr(SSB) = \sum_{i=1}^{k} tr[n_i\ (\bar{x}_i - \bar{x})(\bar{x}_i - \bar{x})']$ is the between group ss = *km$betweenss,*

$\bar{x}_i$ and $\bar{x}$ is the sample mean vector of group *i* and whole dataset respectively.

The statistic $R = \frac{n-k}{k-1}\frac{tr(SSB)}{tr(SSW)}$ (proposed by Calinski and Harabasz) helps us to choose a suitable k, where *n* = total sample size (*=n1+n2+n3=150* in iris example). We want *tr(SSW)* to be as small as possible while *tr(SSB)* to be as large as possible. Therefore, we try several value of *k* so that this statistic *R* is maximized. The following function *kmstat()* is to compute this statistic:

```
# improved kmeans()
# Try kmeans(x,k) serveal times and output the best (largest ratio) trial
# x is the matrix of input variable, k is the no. of clusters
# try is no. of trials
# display cluster size and stat, output cluster label
kmstat<-function(x,k) {
  km<-kmeans(x,k)                                  # k-means clustering with k=3
  ng<-km$size                                      # sample size
  n<-dim(x)[1]
  ssw<-sum(km$withinss)                            # compute total within group ss
  ssb<-km$betweenss                                # between group ss
  out<-list((n-k)*ssb/((k-1)*ssw),ng,km$cluster) # save stat, ng and cluster index into a list
  names(out)<-c("stat","size","cluster")           # apply names to list
  out                                              # output
}
```

Note that this function outputs three items: the R stat, cluster size and the cluster label. We then have to write another function *km()* will perform k-means several times and output the best (largest *R* ratio) trial.

```
km<-function(x,k,try=5) {          # default no. of trial is 5
  res0<-kmstat(x,k)                # save the result of the first trial
  r0<-res0$stat                    # save the stat from the first trial

  for (i in 2:try) {
    res<-kmstat(x,k)               # new trial
    if (res$stat>r0) {             # if new trial is better
      r0<-res$stat                 # update r0 and res
      res0<-res
    }
  }
  cat("cluster size=",res0$size,"\n")  # display cluster size
  cat("stat=",res0$stat,"\n")          # display stat
  res0$cluster                         # output cluster label
}
```

Now we read in these functions and try different k.

```
> source("km.r")               # read in km() function
> km2<-km(x,2)                 # try k=2
cluster size= 53 97
stat= 513.9245
> km3<-km(x,3)                 # try k=3
cluster size= 50 62 38
stat= 561.6278
> km4<-km(x,4)                 # try k=4
cluster size= 32 50 28 40
stat= 530.7658
> km5<-km(x,5)                 # try k=5
cluster size= 40 28 32 22 28
stat= 459.5058
```

Clearly, *k=3* gives the max. value (*=561.6278*) and hence *k=3* is a suitable choice. Finally, we try the *kmeans()* using *HMEQ* dataset.

```
> source("km.r")
> d<-read.csv("hmeq1.csv")                # read in dataset
> x<-d[,-c(1,5,6)]                        # exclude nominal variables
> km2<-km(x,2,try=20)
cluster size= 580 2487
stat= 5076.652
> km3<-km(x,3,try=20)
cluster size= 1411 1223 433
stat= 5649.795
> km4<-km(x,4,try=20)
cluster size= 98 428 1325 1216
stat= 5706.283
> km5<-km(x,5,try=20)                     # max stat at k=5
cluster size= 1072 93 648 350 904
stat= 6255.202
> km6<-km(x,6,try=20)
cluster size= 777 93 380 348 876 593
stat= 6040.757
```

Although *k=5* gives the maximum *R-stat*, one cluster size (*=93*) may be too small to be useful. We can try re-scaling *x* into *[0,1]* first before using *kmeans()*.

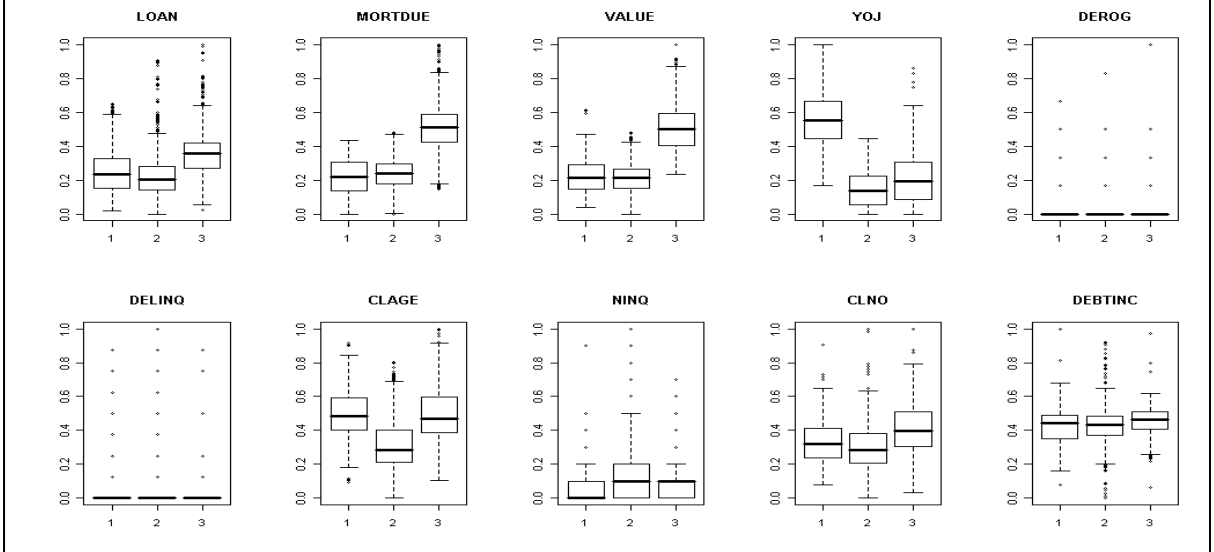
```
> source("scale.r")          # load scale
> z<-scale.con(x)            # rescale continuous variables
> w<-scale.dum(d[,5:6])      # rescale nominal variables
> z<-cbind(z,w)              # combine z and w column-wise
> km2<-km(z,2,try=20)
cluster size= 802 2265
stat= 724.182
> km3<-km(z,3,try=20)        # max stat at k=3
cluster size= 1734 680 653
stat= 788.6409
> km4<-km(z,4,try=20)
cluster size= 1130 600 882 455
stat= 667.9374
> km5<-km(z,5,try=20)
cluster size= 526 892 473 617 559
stat= 590.781
> km6<-km(z,6,try=20)
cluster size= 465 435 532 653 699 283
stat= 532.7219
```

Now the best clustering solution is at *k=3* and each cluster have reasonably large sample size.

6.4 Finding the characteristic of clusters

After grouping the dataset into k homogenous clusters, the next important job is to find out the hidden characteristic of these clusters. A good graphical method to find this out is to produce a side-by-side boxplots to compare all the input variables. Let us continue with the *HMEQ* example the group labels obtained from *km3*.

```
> par(mfrow=c(2,5))                  # define 2x5 multiframe graphic
> lab<-factor(km3)                   # convert label into factor
> plot(lab,z$LOAN,main="LOAN")       # boxplots with group labels from km3
> plot(lab,z$MORTDUE,main="MORTDUE")
> plot(lab,z$VALUE,main="VALUE")
> plot(lab,z$YOJ,main="YOJ")
> plot(lab,z$DEROG,main="DEROG")
> plot(lab,z$DELINQ,main="DELINQ")
> plot(lab,z$CLAGE,main="CLAGE")
> plot(lab,z$NINQ,main="NINQ")
> plot(lab,z$CLNO,main="CLNO")
> plot(lab,z$DEBTINC,main="DEBTINC")
```



From these boxplots, we can roughly describe the characteristics of cluster 1, 2 and 3:
For cluster 1, *YOJ* have relatively higher value than those in clusters 2 and 3.
For cluster 2, *CLAGE* have relatively lower value than those in clusters 1 and 3.
For cluster 3, *MORTDUE* and *VALUE* has higher value than those in cluster 1 and 2.

Alternatively, we can use the results from *k-means* clustering and apply *CTREE* to find out the characteristics of these clusters.

```
> library(rpart)                                                    # load library
> ctree<-rpart(km3~.,data=x,method="class",maxdepth=4)              # apply rpart()
> print(ctree)                                                      # print result

n= 3067
node), split, n, loss, yval, (yprob)
      * denotes terminal node
 1) root 3067 1334 2 (0.222041082 0.565047277 0.212911640)
   2) YOJ>=13.5 744  146 1 (0.803763441 0.075268817 0.120967742)
     4) MORTDUE< 103989 654   61 1 (0.906727829 0.085626911 0.007645260)
       8) CLAGE>=106.0911 601   22 1 (0.963394343 0.028286190 0.008319468) *
       9) CLAGE< 106.0911 53   14 2 (0.264150943 0.735849057 0.000000000) *
     5) MORTDUE>=103989 90    5 3 (0.055555556 0.000000000 0.944444444) *
   3) YOJ< 13.5 2323  646 2 (0.035729660 0.721911322 0.242359019)
     6) VALUE< 125751 1783  156 2 (0.041503085 0.912507011 0.045989905)
      12) CLAGE< 202.58 1353   19 2 (0.005912786 0.985957132 0.008130081) *
      13) CLAGE>=202.58 430  137 2 (0.153488372 0.681395349 0.165116279)
        26) YOJ>=10.5 64    9 1 (0.859375000 0.125000000 0.015625000) *
        27) YOJ< 10.5 366   81 2 (0.030054645 0.778688525 0.191256831) *
     7) VALUE>=125751 540   59 3 (0.016666667 0.092592593 0.890740741) *
```

The classification rules generated from CTREE are consistent with the findings in boxplots.

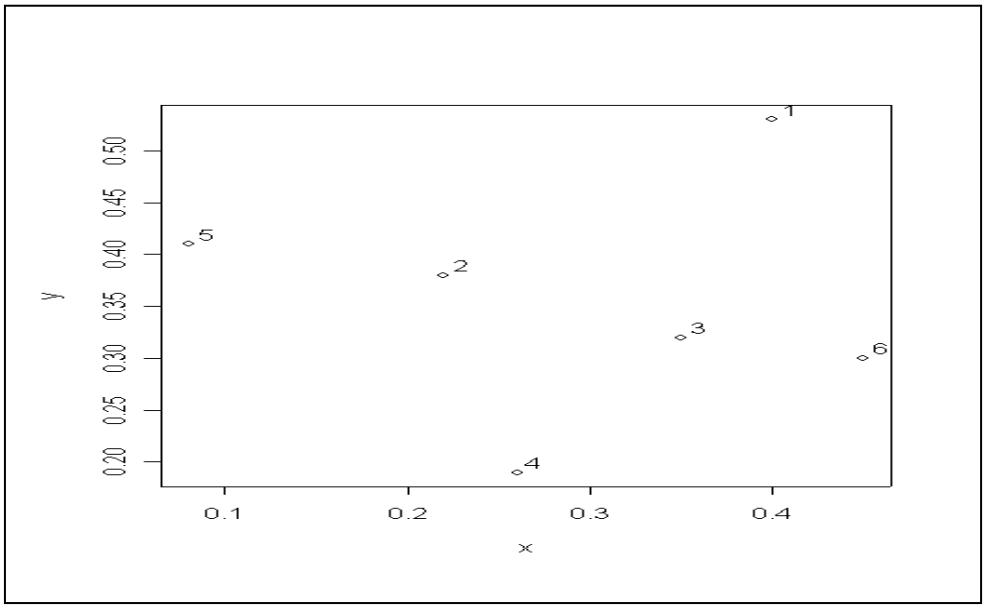
6.5 Hierarchical Clustering (HCLUS)

Although hierarchical clustering is not suitable for large dataset, it is conceptually simple and important in certain area of applications that requires hierarchical structure. Let us illustrate this by a very simply example. The file "*hclus-ex.csv*" contains the x- and y-coordinates of six points.

```
> d<-read.csv("hclus-ex.csv")     # read in data
> dist(d,diag=T,upper=T)          # compute distance matrix
          1         2         3         4         5         6
1 0.0000000 0.2343075 0.2158703 0.3676955 0.3417601 0.2353720
2 0.2343075 0.0000000 0.1431782 0.1941649 0.1431782 0.2435159
3 0.2158703 0.1431782 0.0000000 0.1581139 0.2846050 0.1019804
4 0.3676955 0.1941649 0.1581139 0.0000000 0.2842534 0.2195450
5 0.3417601 0.1431782 0.2846050 0.2842534 0.0000000 0.3860052
6 0.2353720 0.2435159 0.1019804 0.2195450 0.3860052 0.0000000
```

Let us first plot and label these points:

```
> plot(d)
> text(d$x+0.01,d$y+0.01,1:6)      # add labels to points
```



The basic algorithm for *HCLUS* is rather simple:

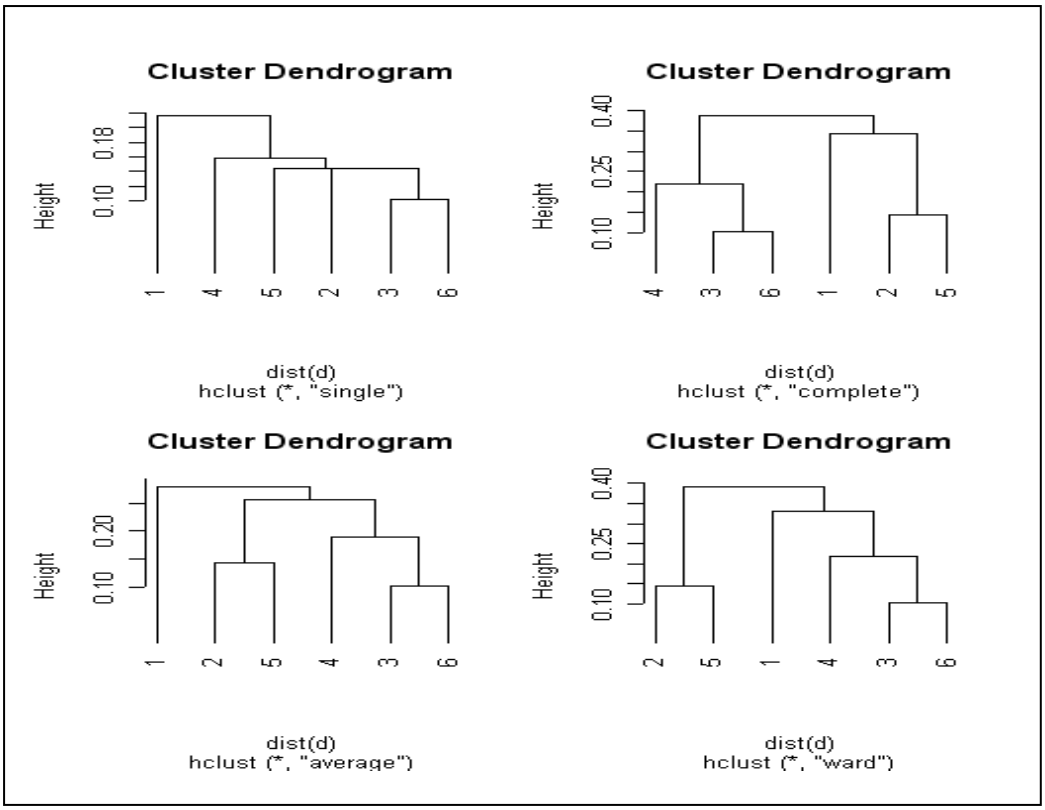
1. Start with every points as a cluster, find the closest two points.
2. Merge the closest two points or clusters into a new cluster.
3. Recompute the distance of other points to this cluster.
4. Find the closest two points or clusters.
5. Repeat steps 2 to 4 until only one cluster remains.

The key question remains is how to define the distance between two clusters. There are several commonly used definitions:

1. Single linkage or Min: $d(C_1, C_2) = \min\{d(x, y) : x \in C_1, y \in C_2\}$
   Minimum of the distances between any two points in different clusters.
2. Complete linkage or Max: $d(C_1, C_2) = \max\{d(x, y) : x \in C_1, y \in C_2\}$
   Maximum of the distances between any two points in different clusters.
3. Group average: $d(C_1, C_2) = average\{d(x, y) : x \in C_1, y \in C_2\}$
   Average of the distances between all pairs of points in different clusters.
4. Ward or Min. variance: $d(C_1, C_2) = \frac{n_1 n_2}{n_1 + n_2} \|m_1 - m_2\|^2$, where m_i is the mean of C_i.
   Weighted average of the distances between the mean of two different clusters.

Let us try *hclust()* using the above example.

```
> hc1<-hclust(dist(d),method="single")          # try different linkage methods
> hc2<-hclust(dist(d),method="complete")
> hc3<-hclust(dist(d),method="average")
> hc4<-hclust(dist(d),method="ward")
> par(mfrow=c(2,2))
> plot(hc1,hang=-1)                             # plots
> plot(hc2,hang=-1)
> plot(hc3,hang=-1)
> plot(hc4,hang=-1)
```



6.6 Strengths and Weakness

As mentioned earlier, K-means clustering is suitable for clustering a huge dataset while hierarchical clustering can build a hierarchical structure. Here is more detailed discussion about the strengths and weakness about these two methods.

1. K-means clustering requires specifying the number of cluster k in advance.
2. K-means clustering is sensitive to initial seed. We may need to try several times and compare the solutions.
3. K-means clustering is sensitive to outliers. It is advisable to remove possible outliers before K-means clustering in practice.
4. K-means clustering is fast and efficient, suitable for huge dataset.
5. K-means clustering can be viewed as an optimization problem. For example, if Euclidean distance is used, the cluster mean will minimize the within cluster SSE; while Manhattan distance is used, the cluster median will minimize the within cluster sum of absolute deviation.
6. HCLUSTER is computationally intensive and not suitable for huge dataset.
7. Different linkage methods used in HCLUSTER may leads to different results.
8. Once two clusters are merged in HCLUSTER, it cannot be undone at a later stage. The fact that these merges are final can also cause trouble for noisy, high-dimensional data.

6.7 K-means using EXCEL

Although the k-means clustering algorithm is iterative in nature, we can find the solution by using the *solver()* function to minimize $E(z)$ in equation (6.1). We use the iris data as our example.

1. The data is in cells B6 to E156.
2. The cluster center c1, c2 and c3 for group 1 to 3 are in B2 to E2. We may first use the 1st, 51st and 101st observation as the center for group1 to 3. That is, copy B7:E7 to B2:E2; B57:E57 to B3:E3; and B107:E107 to B4:E4.
3. We compute x-c1 by entering =B7-B$2 in H7 and copy it to K156. Similarly we compute x-c2 by entering =B7-B$3 in M7 and copy it to P156; and compute x-c3 by entering =B7-B$4 in R7 and copy it to U156.
4. We compute the squared Euclidean distance of the 1st observation to c1 by entering =SUMSQ(H7:K7) in W7. Similarly we compute the squared distance to c2 by entering =SUMSQ(M7:P7) in X7 and the squared distance to c3 by entering =SUMSQ(R7:U7) in Y7. Then we copy W7:Y7 down to W156:Y156.
5. Now we find the minimum distance by entering =MIN(W7:Y7) in AA7 and copy it down to AA156.
6. We obtain the class label by entering =1*(W7=AA7)+2*(X7=AA7)+3*(Y7=AA7) in Z7. This gives us the group label such that it is closest to the 1st observation. We copy it down to Z156.
7. We compute the sum of all these minimum squared distance by entering =SUM(AA7:AA156) in AA158. This the objective function value we need to minimize using solver().
8. We use solver and enter $AA$158 as the objective cell to minimize and $B$2:$E$4 as the variable cells. Click solve to obtain the clustering result.
9. Finally we can obtain the cluster size for group 1 by entering =COUNTIF($Z$7:$Z$156,"=1"). We can find cluster size for other groups similarly.

Reference:

Chapter 8 of Introduction to Data Mining by Tan, Steinbach and Kumar, Addison Wesley.