Cluster Analysis

聚 类 分 析

肖磊，2026年5月19日

Outline

聚类分析 (Cluster Analysis)

- 问题 (The Problem)
- 相似性度量 (The Proximity Between Objects)
- 聚类算法 (Cluster Algorithms)
- 自适应权重聚类 (Adaptive Weights Clustering)
- 谱聚类 (Spectral Clustering)
- 波士顿房屋数据 (Boston Housing)

Preface 引言

- 接下来的两章内容我们从两个不同的角度讨论分类问题.
- 给定对若干个变量进行观测得到的一个数据集:
  - 我们想知道是否存在由某些个体自然构成的子群或类别.
  - 我们想根据现有的一组群体对某些个体进行分类.

Cluster Analysis 聚类分析

Discriminant Analysis 判别分析

- 聚类分析: 建立由某些个体构成的一些自然形成的子群或类，这可以通过根据适当的标准对“相似”的个体进行分组来实现.
- 聚类分析的应用领域广泛，如自然科学、工程领域、医学、经济学、营销学、心理学、社会学、语言学、生物学等.

Preface 引言

- 一旦形成聚类，通常使用第 1 章、第 10 章和第 11 章中的一些描述性工具来描述每一类会非常有帮助，以更好地理解类之间存在的差异.
- 用当前的语言来说，聚类分析通常被称为*无监督学习* (unsupervised learning).
  - 在这里，“*学习*”指的是统计学家根据数据来校准模型，反之亦然.
  - 所谓“*无监督*”指的是从观察到的数据当中创建模型的行为 (这里是将个体分成小组).
- 一旦定义了组或类，就采用*有监督学习* (supervised learning)，所谓“*有监督*”是指类的构成事先已知.

The Problem 问题

- 聚类分析是从多变量数据对象当中构建组 (类) 的一组工具.
  - 其目的是从由多种成分构成的大样本当中构建或寻找具有相同性质的组 (类).
  - 同组或同类应尽可能具有相同或相似的性质，不同组或不同类之间的差异应尽可能大.

The Problem 问题

- 聚类分析可以分为两个基本步骤.

1. 选择一个相似性的度量方法:

   检查每一对观测数据 (或观测对象) 的值是否相似. 相似性 (或接近度) 的度量被定义为观测对象的 “接近程度”. 它们越 “接近”， 同一类的可能性就越高.

2. 构建类的算法的选择:

   在相似性度量的基础上，将每一个观测对象分配给不同的类，使得类之间的差异足够大，类当中的观测值变得尽可能接近.

The Proximity Between Objects 相似性度量

- 聚类分析的起点是含有 p 个变量、n 个测量值（对象）的数据矩阵 $\mathcal{X}$.

$$\begin{matrix} & X_1 & X_2 & \cdots & X_p \\ & \downarrow & \downarrow & \cdots & \downarrow \end{matrix}$$

$$\mathcal{X}_{n\times p} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{nn2} & \cdots & x_{np} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \boldsymbol{x}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_n^{\mathrm{T}} \end{pmatrix}$$

The Proximity Between Objects 相似性度量

- 对象之间的相似性通过一个矩阵 $\mathscr{D}_{n\times n}$ 来描述

$$\mathscr{D} = \begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{pmatrix}$$

  - 矩阵 $\mathscr{D}$ 由 n 个对象的相似性或差异性的度量构成.
    - 如果 d_{ij} 表示距离，则它们是差异性度量. 距离越大，对象之间的相似性越小.

$$d_{ij} \triangleq \| \boldsymbol{x}_i - \boldsymbol{x}_j \|_2$$

    - 如果 d_{ij} 是相似性度量，则相似性的值越大，对象之间的相似程度就越高.

$$d'_{ij} \triangleq \max_{i,\,j} \left\{ d_{ij} \right\} - d_{ij}$$

The Proximity Between Objects 相似性度量

- 观测数据的性质在相似性度量的选择中发挥着重要作用.
  - 名义值 (如二元变量) 我们一般会使用相似性度量.
  - 连续型变量的值我们一般会使用距离矩阵.

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性

$$\boldsymbol{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}, \quad \boldsymbol{x}_j = \begin{pmatrix} x_{j1} \\ x_{j2} \\ \vdots \\ x_{jp} \end{pmatrix}, \qquad \begin{cases} x_{ik}\,,\, x_{jk} \in \{0\,,\,1\} \\ i\,,\, j = 1\,,\,2\,,\,\cdots\,,\,n \\ k = 1\,,\,2\,,\,\cdots\,,\,p \end{cases}$$

- 共有四种情形:

$$\begin{cases} x_{ik} = x_{jk} = 1 \\ x_{ik} = 0\,,\, x_{jk} = 1 \\ x_{ik} = 1\,,\, x_{jk} = 0 \\ x_{ik} = x_{jk} = 0 \end{cases} \Longrightarrow \begin{cases} a_1 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = x_{jk} = 1\right) \\ a_2 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = 0\,,\, x_{jk} = 1\right) \\ a_3 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = 1\,,\, x_{jk} = 0\right) \\ a_4 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = x_{jk} = 0\right) \end{cases}$$

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性

$$\boldsymbol{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}, \quad \boldsymbol{x}_j = \begin{pmatrix} x_{j1} \\ x_{j2} \\ \vdots \\ x_{jp} \end{pmatrix}, \qquad \begin{cases} x_{ik}, x_{jk} \in \{0, 1\} \\ i, j = 1, 2, \ldots, n \\ k = 1, 2, \ldots, p \end{cases}$$

  - 实际中常用到的相似性度量:

$$d_{ij} = \frac{a_1 + \delta a_4}{a_1 + \delta a_4 + \lambda (a_2 + a_3)}$$

δ 与 λ 是权重因子

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性

$$d_{ij} = \frac{a_1 + \delta a_4}{a_1 + \delta a_4 + \lambda\left(a_2 + a_3\right)}$$

Name	δ	λ	Definition
Jaccard	0	1	$d_{ij} = \frac{a_1}{a_1 + a_2 + a_3}$
Tanimoto	1	2	$d_{ij} = \frac{a_1 + a_4}{a_1 + 2\left(a_2 + a_3\right) + a_4}$
Simple Matching (M)	1	1	$d_{ij} = \frac{a_1 + a_4}{p}$
Russel and Rao (RR)	-	-	$d_{ij} = \frac{a_1}{p}$
Dice	0	$\frac{1}{2}$	$d_{ij} = \frac{2a_1}{2a_1 + \left(a_2 + a_3\right)}$
Kulczynski	-	-	$d_{ij} = \frac{a_1}{a_2 + a_3}$

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性
  - ***Example***: Car Marks 数据集. 定义新的二元数据如下

$$y_{ik} = \begin{cases} 1, & x_{ik} > \bar{x} \\ 0, & \text{otherv} \end{cases}$$

```
# clear variables and close windows
rm(list = ls(all = TRUE))
graphics.off()
library(readr)
Carmean = read_csv("Desktop/2023_Applied Mult
        col_names = TRUE)
head(Carmean)
x = Carmean[, 2:9]
head(x)
```

```
> head(Carmean)
# A tibble: 6 × 9
  cars  economic service value price  look sporty security  easy
  <chr>    <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>
1 A100       3.9     2.8   2.2   4.2   3      3.1      2.4   2.8
2 BMW3       4.8     1.6   1.9   5     2      2.5      1.6   2.8
3 CiAX       3       3.8   3.8   2.7   4      4.4      4     2.6
4 Ferr       5.3     2.9   2.2   5.9   1.7    1.1      3.3   4.3
5 FiUn       2.1     3.9   4     2.6   4.5    4.4      4.4   2.2
6 FoFi       2.3     3.1   3.4   2.6   3.2    3.3      3.6   2.8
```

```
> head(x)
# A tibble: 6 × 8
  economic service value price  look sporty security  easy
     <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>
1      3.9     2.8   2.2   4.2   3      3.1      2.4   2.8
2      4.8     1.6   1.9   5     2      2.5      1.6   2.8
3      3       3.8   3.8   2.7   4      4.4      4     2.6
4      5.3     2.9   2.2   5.9   1.7    1.1      3.3   4.3
5      2.1     3.9   4     2.6   4.5    4.4      4.4   2.2
6      2.3     3.1   3.4   2.6   3.2    3.3      3.6   2.8
```

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性
  - ***Example***: Car Marks 数据集. 定义新的二元数据如下

$$y_{ik} = \begin{cases} 1, & x_{ik} > \bar{x}_k \\ 0, & \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, p$$

```
nr_x = dim(x)[1] # 观测值个数（行数）
nc_x = dim(x)[2] # 变量个数（列数）

colmean_x = apply(x, 2, mean) # 计算每个变量的平均值
# 定义二元数据矩阵：大于各变量均值取 1，否则取 0
for (i in 1:nr_x) {
  for (j in 1:nc_x) {
    if (x[i, j] > colmean_x[j]) x[i, j] = 1
    else x[i, j] = 0
  }
}
head(x)
```

```
> head(x)
# A tibble: 6 x 8
  economic service value price  look sporty security  easy
     <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>
1        1       0     0     1     0      0        0     1
2        1       0     0     1     0      0        0     1
3        0       1     1     0     1      1        1     0
4        1       0     0     1     0      0        1     1
5        0       1     1     0     1      1        1     0
6        0       1     1     0     0      0        1     1
```

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性
  - ***Example***: Car Marks 数据集. 定义新的二元数据矩阵

$$y_{ik} = \begin{cases} 1, & x_{ik} > \bar{x}_k \\ 0, & \text{otherwise} \end{cases}$$

```
> x[17:19, ] # 以 17~19 这三行为例
# A tibble: 3 x 8
  economic service value price  look sporty security  easy
     <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>
1        0       1     1     0     0      0        0     0
2        1       0     0     1     0      0        0     1
3        0       0     1     0     0      0        0     1
```

```
x[17:19, ] # 以 17~19 这三行为例
```

- Jaccard measure $d_{ij} = \dfrac{a_1}{a_1 + a_2 + a_3}$

$$\mathscr{D} = \begin{pmatrix} 1.000 & 0.000 & 0.333 \\ & 1.000 & 0.250 \\ & & 1.000 \end{pmatrix}$$

$$d_{11} = \frac{2}{2} = 1$$

$$d_{12} = \frac{0}{0+3+2} = 0$$

$$d_{13} = \frac{1}{1+1+1} = \frac{1}{3}$$

$$d_{22} = \frac{3}{3} = 1$$

$$d_{23} = \frac{1}{1+1+2} = \frac{1}{4}$$

$$d_{33} = \frac{2}{2} = 1$$

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性
  - ***Example***: Car Marks 数据集. 定义新的二元数据矩阵

$$y_{ik} = \begin{cases} 1, & x_{ik} > \bar{x}_k \\ 0, & \text{otherwise} \end{cases}$$

```
> x[17:19, ] # 以 17~19 这三行为例
# A tibble: 3 x 8
  economic service value price  look sporty security  easy
     <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>
1        0       1     1     0     0      0        0     0
2        1       0     0     1     0      0        0     1
3        0       0     1     0     0      0        0     1
```

```
x[17:19, ] # 以 17~19 这三行为例
```

- Tanimoto measure $d_{ij} = \dfrac{a_1 + a_4}{a_1 + 2\left(a_2 + a_3\right) + a_4}$

$$\mathscr{D} = \begin{pmatrix} 1.000 & 0.231 & 0.556 \\ & 1.000 & 0.455 \\ & & 1.000 \end{pmatrix}$$

$$d_{11} = \frac{2+6}{2+2(0+0)+6} = 1$$

$$d_{12} = \frac{0+3}{0+2(3+2)+3} = \frac{3}{13}$$

$$d_{13} = \frac{0+5}{0+2(1+1)+5} = \frac{5}{9}$$

$$d_{22} = \frac{3+5}{3+2(0+0)+5} = 1$$

$$d_{23} = \frac{1+4}{1+2(1+2)+4} = \frac{5}{11}$$

$$d_{33} = \frac{2+6}{2+2(0+0)+6} = 1$$

The Proximity Between Objects 相似性度量

- 二元结构对象之间的相似性
  - ***Example***: Car Marks 数据集. 定义新的二元数据矩阵

$$y_{ik} = \begin{cases} 1, & x_{ik} > \bar{x}_k \\ 0, & \text{otherwise} \end{cases}$$

```
> x[17:19, ] # 以 17~19 这三行为例
# A tibble: 3 x 8
  economic service value price  look sporty security  easy
     <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>
1        0       1     1     0     0      0        0     0
2        1       0     0     1     0      0        0     1
3        0       0     1     0     0      0        0     1
```

```
x[17:19, ] # 以 17~19 这三行为例
```

- Simple Matching $d_{ij} = \dfrac{a_1 + a_4}{p}$

$$\mathscr{D} = \begin{pmatrix} 1.000 & 0.375 & 0.625 \\ & 1.000 & 0.625 \\ & & 1.000 \end{pmatrix}$$

$$d_{11} = \frac{2+6}{8} = 1$$

$$d_{12} = \frac{0+3}{8} = \frac{3}{8}$$

$$d_{13} = \frac{0+5}{8} = \frac{5}{8}$$

$$d_{22} = \frac{3+5}{8} = 1$$

$$d_{23} = \frac{1+4}{8} = \frac{5}{8}$$

$$d_{33} = \frac{2+6}{8} = 1$$

The Proximity Between Objects 相似性度量

- 连续型变量的距离度量
  - 利用 L_r 范数可以生成广泛用到的距离度量

$$d_{ij} = \| \boldsymbol{x}_i - \boldsymbol{x}_j \|_r = \left(\sum_{k=1}^{p} \left| x_{ik} - x_{jk} \right|^r \right)^{1/r}, \quad r \geq 1$$

    - $d_{ii} = 0, \quad i = 1, 2, \ldots, n.$
    - L_1-norm: $d_{ij} = \sum_{k=1}^{p} \left| x_{ik} - x_{jk} \right|.$
    - L_2-norm: $d_{ij} = \sqrt{\sum_{k=1}^{p} \left(x_{ik} - x_{jk} \right)^2}.$

The Proximity Between Objects 相似性度量

- 连续型变量的距离度量
  - ***Example***: 假设 $\boldsymbol{x}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\boldsymbol{x}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\boldsymbol{x}_3 = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$.
    - L_1 范数的距离矩阵:

$$d_{ij} = \sum_{k=1}^{p} \left| x_{ik} - x_{jk} \right|$$

$$\mathscr{D}_1 = \begin{pmatrix} 0 & 1 & 10 \\ 1 & 0 & 9 \\ 10 & 9 & 0 \end{pmatrix}$$

    - L_2 范数的距离矩阵:

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \left(x_{ik} - x_{jk} \right)^2}$$

$$\mathscr{D}_2 = \begin{pmatrix} 0 & 1 & \sqrt{50} \\ 1 & 0 & \sqrt{41} \\ \sqrt{50} & \sqrt{41} & 0 \end{pmatrix}$$

The Proximity Between Objects 相似性度量

- 连续型变量的距离度量

  - 应用基于 L_r 范数的距离的基本前提是假设变量的尺度相同.

  - 更一般的以 $\mathscr{A}$ 为度量的 L_2 范数或欧几里得范数 (其中$\mathscr{A} > 0$):

$$d_{ij}^2 = \| \boldsymbol{x}_i - \boldsymbol{x}_j \|_{\mathscr{A}} = \left(\boldsymbol{x}_i - \boldsymbol{x}_j \right)^{\mathrm{T}} \mathscr{A} \left(\boldsymbol{x}_i - \boldsymbol{x}_j \right)$$

L_2 范数: $\mathscr{A} = \mathscr{I}_p$

  - 如果需要标准化，则需要以下的权重矩阵:

$$\mathscr{A} = \mathrm{diag}\left(s_{X_1X_1}^{-1}, s_{X_2X_2}^{-1}, \cdots, s_{X_pX_p}^{-1} \right) = \begin{pmatrix} s_{X_1X_1}^{-1} & & & \\ & s_{X_2X_2}^{-1} & & \\ & & \ddots & \\ & & & s_{X_pX_p}^{-1} \end{pmatrix} \Longrightarrow d_{ij}^2 = \sum_{k=1}^{p} \frac{\left(x_{ik} - x_{jk} \right)^2}{s_{X_kX_k}}$$

The Proximity Between Objects 相似性度量

- 连续型变量的距离度量
  - ***Example***: 法国食品支出数据集.

```
setwd("~/Desktop/2023_Applied Multivariate Statistical Analysis/R Codes with data/Data")
library(data.table)
x = read.csv("food.csv", header = TRUE)
(x = x[, 2:7])
```

```
> (x = x[, 2:7])
   bread veget fruit meat poult milk
1    332   428   354 1437   526  247
2    293   559   388 1527   567  239
3    372   767   562 1948   927  235
4    406   563   341 1507   544  324
5    386   608   396 1501   558  319
6    438   843   689 2345  1148  243
7    534   660   367 1620   638  414
8    460   699   484 1856   762  400
9    385   789   621 2366  1149  304
10   655   776   423 1848   759  495
11   584   995   548 2056   893  518
12   515  1097   887 2630  1167  561
```

The Proximity Between Objects 相似性度量

- 连续型变量的距离度量
  - ***Example***: 法国食品支出数据集.
    - 欧氏距离矩阵:

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \left(x_{ik} - x_{jk} \right)^2}$$

```
D_1 = dist(x, method = "euclidean", diag = TRUE, upper = FALSE, p = 2)
round(D_1, digits = 2)
```

```
> round(D_1, digits = 2)
        1      2      3      4      5      6      7      8      9     10     11     12
1     0.0
2   172.3    0.0
3   762.8  621.8    0.0
4   187.1  152.1  663.9    0.0
5   217.7  135.2  629.3   75.5    0.0
6  1227.6 1095.0  482.5 1128.8 1100.0    0.0
7   410.8  336.2  546.7  237.0  234.6  980.5    0.0
8   601.2  478.4  285.2  465.8  442.4  689.9  303.2    0.0
9  1207.8 1078.2  482.6 1111.0 1086.8  120.5  966.0  672.0    0.0
10  717.6  620.3  453.1  553.0  541.4  764.3  323.1  238.3  754.4    0.0
11 1006.9  874.3  435.9  849.7  819.7  537.1  643.0  422.1  543.0  361.1    0.0
12 1642.5 1506.0  929.5 1516.8 1483.5  540.7 1335.4 1054.9  564.4 1058.4  732.5    0.0
```

The Proximity Between Objects 相似性度量

- 连续型变量的距离度量
  - ***Example***: 法国食品支出数据集.
    - 取权重矩阵 $\mathscr{A} = \mathrm{diag}\left(s_{X_1X_1}^{-1}, s_{X_2X_2}^{-1}, \cdots, s_{X_pX_p}^{-1}\right)$，距离矩阵则为:

```
sx = scale(x)
D_2 = dist(sx, method = "euclidean", diag = TRUE, upper = FALSE, p = 2)
round(D_2, digits = 2)
```

```
> round(D_2, digits = 2)
      1    2    3    4    5    6    7    8    9   10   11   12
1  0.00
2  0.86 0.00
3  3.03 2.47 0.00
4  1.21 1.32 2.69 0.00
5  1.28 1.14 2.39 0.46 0.00
6  4.62 4.14 1.71 4.19 3.94 0.00
7  2.74 2.78 2.89 1.59 1.69 3.90 0.00
8  2.80 2.55 1.87 1.87 1.72 2.81 1.28 0.00
9  4.30 3.80 1.55 3.87 3.63 0.88 3.66 2.48 0.00
10 4.37 4.34 3.62 3.25 3.29 3.93 1.68 2.07 3.82 0.00
11 5.09 4.77 3.36 4.02 3.89 3.21 2.76 2.34 3.14 1.72 0.00
12 6.98 6.53 4.51 6.19 5.94 3.41 5.28 4.36 3.46 4.40 2.89 0.00
```

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \frac{\left(x_{ik} - x_{jk}\right)^2}{s_{X_kX_k}}}$$

The Proximity Between Objects 相似性度量

- 连续型变量的距离度量
  - 我们也可以采用相似性独立，例如相关系数

$$d_{ij} = \frac{\sum_{k=1}^{p} \left(x_{ik} - \bar{x}_i\right)\left(x_{jk} - \bar{x}_j\right)}{\sqrt{\left[\sum_{k=1}^{p} \left(x_{ik} - \bar{x}_i\right)^2\right] \cdot \left[\sum_{k=1}^{p} \left(x_{jk} - \bar{x}_j\right)^2\right]}}$$

Cluster Algorithms 聚类算法

- 三种经典的聚类算法

分层算法
- Hierarchical algorithms
  - 凝聚过程 Agglomerative procedures
  - 分裂过程 Splitting procedures
- 分区算法 Partitioning algorithms

  - 凝聚过程: 从每一个观测值自成一类开始，逐步合并分组.
  - 分裂过程: 从所有观测值归为一类开始，逐步分隔为更小的类.

某点一旦归类则不再变化！

  - 分区算法: 从事先给定的一些类开始，然后观测值在各类之间交换，直到满足某种最优标准. 也称为动态聚类.

点可以在不同类之间变化.

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - 先确定聚类的数目 k.
  - 数据看作是某个度量空间当中的点. 例如 $\mathbb{R}^p$ 空间当中的 n 个点.
  - 给度量空间中成对的点之间定义距离 (差异性度量).
  - 目标是将数据点分成 k 类，以使得基于点之间的距离以及点到重心的距离上的目标函数达到最大化或者最小化.

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - ***k-means*** 聚类: 最常用的分区算法.
    - 开始于某一随机分区或者点.
    - 目标函数为总的类内距离 (total intra-cluster distance) 或者平方误差函数 (squared error function).

$$\widehat{\mathcal{S}} = \underset{\mathcal{S}}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{i \in \mathcal{S}_j} \| \boldsymbol{x}_i - \boldsymbol{\mu}_j \|^2$$

k 为聚类数

$\boldsymbol{\mu}_j$ 是 $\mathcal{S}_j$ 的重心

$$\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k\}$$

$$\bigcup_{j=1}^{k} \mathcal{S}_j = \{1, 2, \ldots, n\}$$

n 是数据点的个数

$\mathcal{S}_j$ 给出第 j 类中点的下标

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - *k-means* 聚类: 最常用的分区算法.
    - *k-means* 聚类的标准算法

给定初始集 $\left\{\boldsymbol{\mu}_j^{(t)}\right\}_{j=1}^{k}$, $t=1$

赋值 $\hat{j}(i)=\underset{j}{\operatorname{argmin}}\left\|\boldsymbol{x}_i-\boldsymbol{\mu}_j^{(t)}\right\|^2$

$\boldsymbol{x}_i$ 属于类 $\hat{j}(i)$ 导致的 (新的) 分区

$$\bigcup_{j=1}^{k}\mathcal{S}_j^{(t)}=\{1,\,2,\,\ldots,\,n\}$$

更新 $\boldsymbol{\mu}_j^{(t+1)}=\left(\#\mathcal{S}_j^{(t)}\right)^{-1}\sum_{i\in\mathcal{S}_j^{(t)}}\boldsymbol{x}_i$

重复

赋值，更新

直至以 $\widehat{\mathcal{S}}=\underset{\mathcal{S}}{\operatorname{argmin}}\sum_{j=1}^{k}\sum_{i\in\mathcal{S}_j}\left\|\boldsymbol{x}_i-\boldsymbol{\mu}_j\right\|^2$ 收敛

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - ***k-means*** 聚类: 最常用的分区算法.
    - 该算法在很大程度上取决于重心的初始选择.
    - 其解可能不是最优的.
    - 从不同的初始条件进行更多的运行，可以改进聚类结果.
    - 据称可以选择更好的初始重心的其它方法有:

      ***k-means* ++**

      intelligent k-means

      genetic k-means

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - ***k-means*** 聚类: 最常用的分区算法.
  - ***Example***: 利用 ***k-means*** 标准算法将 8 个点聚为两类.

```
eight = cbind(c(-3, -2, -2, -2, 1, 1, 2, 4), c(0, 4, -1, -2, 4, 2, -4, -3))
eight
```

```
eight = eight[c(8, 7, 3, 1, 4, 2, 6, 5), ] # 数据集重新排序
eight
```

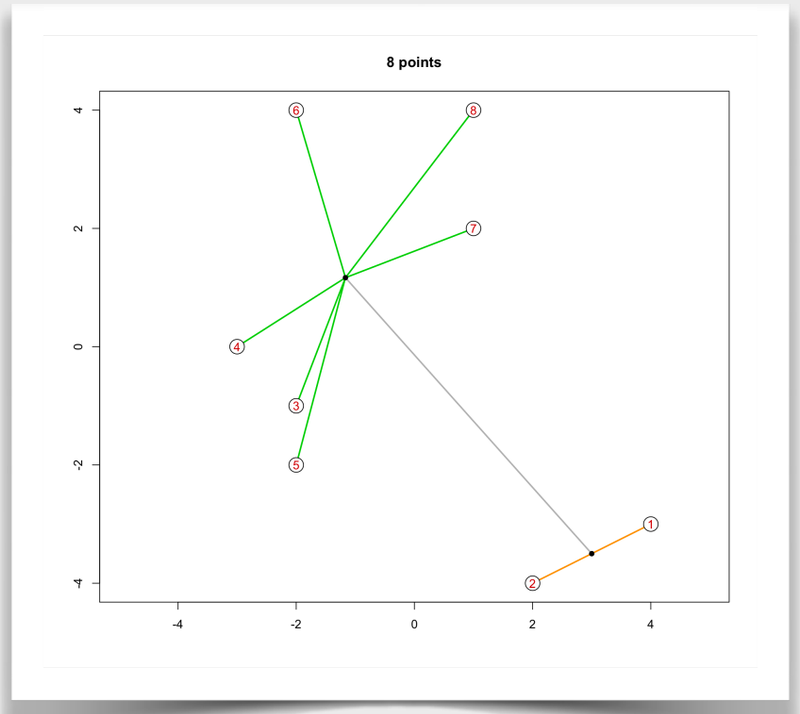
```
> eight
     [,1] [,2]
[1,]   -3    0
[2,]   -2    4
[3,]   -2   -1
[4,]   -2   -2
[5,]    1    4
[6,]    1    2
[7,]    2   -4
[8,]    4   -3
```

```
> eight
     [,1] [,2]
[1,]    4   -3
[2,]    2   -4
[3,]   -2   -1
[4,]   -3    0
[5,]   -2   -2
[6,]   -2    4
[7,]    1    2
[8,]    1    4
```

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - ***k-means*** 聚类: 最常用的分区算法.
  - ***Example***: 利用 ***k-means*** 标准算法将 8 个点聚为两类.

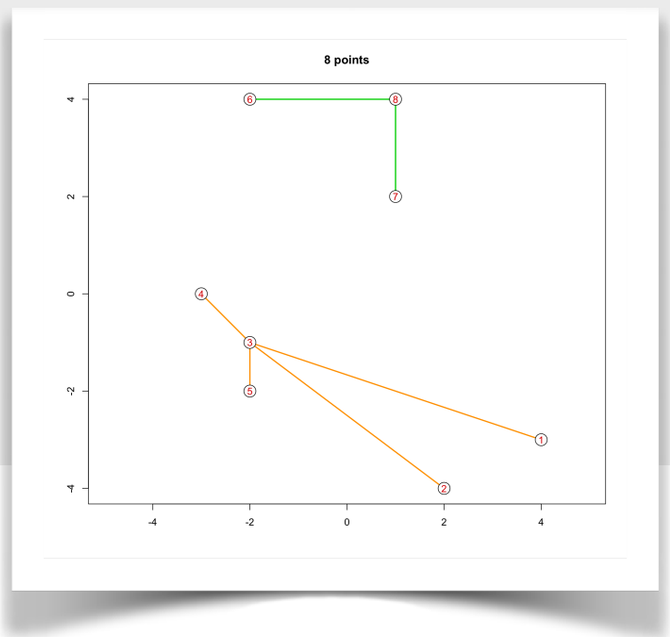
```
plot(eight, type = "n", xlab = "", ylab = "", xlim = c(-4, 4), ylim = c(-4, 4), main = "8 points", asp = 1)
p12 = eight[1:2, ]
p12.mean = apply(p12, 2, mean)
segments(p12.mean[1], p12.mean[2], eight[1, 1], eight[1, 2], lwd = 2, col = 'orange')
segments(p12.mean[1], p12.mean[2], eight[2, 1], eight[2, 2], lwd = 2, col = 'orange')
p3_8 = eight[3:8, ]
p3_8.mean = apply(p3_8, 2, mean)
segments(p3_8.mean[1], p3_8.mean[2], eight[3, 1], eight[3, 2], lwd = 2, col = 'green3')
segments(p3_8.mean[1], p3_8.mean[2], eight[4, 1], eight[4, 2], lwd = 2, col = 'green3')
segments(p3_8.mean[1], p3_8.mean[2], eight[5, 1], eight[5, 2], lwd = 2, col = 'green3')
segments(p3_8.mean[1], p3_8.mean[2], eight[6, 1], eight[6, 2], lwd = 2, col = 'green3')
segments(p3_8.mean[1], p3_8.mean[2], eight[7, 1], eight[7, 2], lwd = 2, col = 'green3')
segments(p3_8.mean[1], p3_8.mean[2], eight[8, 1], eight[8, 2], lwd = 2, col = 'green3')
segments(p12.mean[1], p12.mean[2], p3_8.mean[1], p3_8.mean[2], lwd = 2, col = 'grey')
points(p12.mean[1], p12.mean[2], pch = 16, cex = 1)
points(p3_8.mean[1], p3_8.mean[2], pch = 16, cex = 1)
points(eight, pch = 21, cex = 2.7, bg = "white")
text(eight, as.character(1:8), col = "red3", cex = 1)
```



Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - 下面给出的是一些其他常用的分区方法:
    - ***k-medoids***: 对每一类 j, 我们定义最接近重心位置的一个点 $\boldsymbol{x}_j$ 作为这一类的参考点 ***medoids***.
    - ***k-means*** 和 ***k-medoids*** 的差别在于：***k-means*** 可以选择 k 个虚拟的点作为类的重心，而 ***k-medoids*** 的重心应当是 k 个实际的点.

```
plot(eight, type = "n", xlab = "", ylab = "", xlim = c(-4, 4), ylim = c(-4, 4), main = "8 points", asp = 1)
segments(eight[3, 1], eight[3, 2], eight[1, 1], eight[1, 2], lwd = 2, col = 'orange')
segments(eight[3, 1], eight[3, 2], eight[2, 1], eight[2, 2], lwd = 2, col = 'orange')
segments(eight[3, 1], eight[3, 2], eight[4, 1], eight[4, 2], lwd = 2, col = 'orange')
segments(eight[3, 1], eight[3, 2], eight[5, 1], eight[5, 2], lwd = 2, col = 'orange')
segments(eight[8, 1], eight[8, 2], eight[6, 1], eight[6, 2], lwd = 2, col = 'green3')
segments(eight[8, 1], eight[8, 2], eight[7, 1], eight[7, 2], lwd = 2, col = 'green3')
points(eight, pch = 21, cex = 2.7, bg = "white")
text(eight, as.character(1:8), col = "red3", cex = 1)
```



Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - 下面给出的是一些其他常用的分区方法:
    - *k-mode*: 是拓展到处理分类数据集的一种方法，它用众数 (modes) 代替类的均值.

Cluster Algorithms 聚类算法

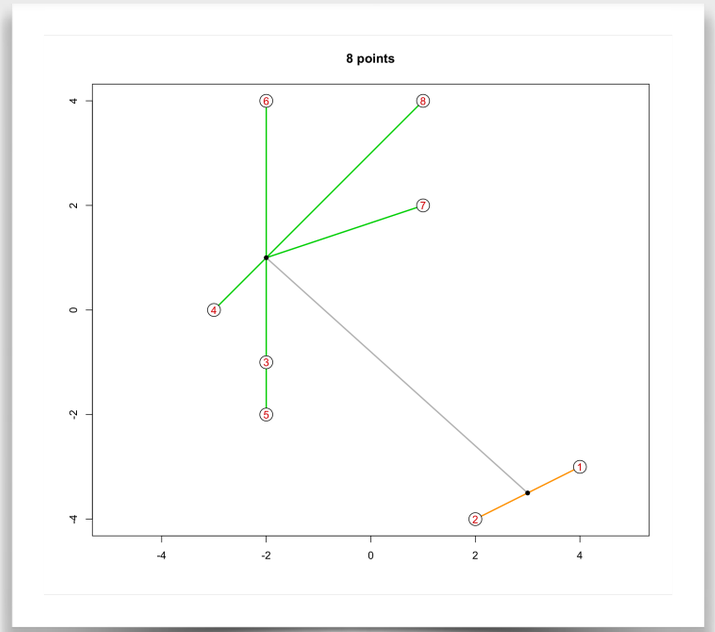
- Partitioning Algorithms 分区算法
  - 下面给出的是一些其他常用的分区方法:
    - ***k-median***: 为了克服 ***k-means*** 对异常值高度敏感而设计的一种算法，因为经验均值很容易收到极端值的影响.
    - 重心不是通过计算每个类的均值来确定，而是计算相对坐标的中位数来确定.
    - 目标函数采用的是 Manhattan 距离，而非欧氏距离的平方

$$\widehat{\mathcal{S}} = \underset{\mathcal{S}}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{i \in \mathcal{S}_j} \left| \boldsymbol{x}_i - \mathrm{med}_j \right|$$

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - 下面给出的是一些其他常用的分区方法:
    - *k-median*:

```
plot(eight, type = "n", xlab = "", ylab = "", xlim = c(-4, 4), ylim = c(-4, 4), main = "8 points", asp = 1)
p12 = eight[1:2, ]
p12.median = apply(p12, 2, median)
segments(p12.median[1], p12.median[2], eight[1, 1], eight[1, 2], lwd = 2, col = 'orange')
segments(p12.median[1], p12.median[2], eight[2, 1], eight[2, 2], lwd = 2, col = 'orange')
p3_8 = eight[3:8, ]
p3_8.median = apply(p3_8, 2, median)
segments(p3_8.median[1], p3_8.median[2], eight[3, 1], eight[3, 2], lwd = 2, col = 'green3')
segments(p3_8.median[1], p3_8.median[2], eight[4, 1], eight[4, 2], lwd = 2, col = 'green3')
segments(p3_8.median[1], p3_8.median[2], eight[5, 1], eight[5, 2], lwd = 2, col = 'green3')
segments(p3_8.median[1], p3_8.median[2], eight[6, 1], eight[6, 2], lwd = 2, col = 'green3')
segments(p3_8.median[1], p3_8.median[2], eight[7, 1], eight[7, 2], lwd = 2, col = 'green3')
segments(p3_8.median[1], p3_8.median[2], eight[8, 1], eight[8, 2], lwd = 2, col = 'green3')
segments(p12.median[1], p12.median[2], p3_8.median[1], p3_8.median[2], lwd = 2, col = 'grey')
points(p12.median[1], p12.median[2], pch = 16, cex = 1)
points(p3_8.median[1], p3_8.median[2], pch = 16, cex = 1)
points(eight, pch = 21, cex = 2.7, bg = "white")
text(eight, as.character(1:8), col = "red3", cex = 1)
```



Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - 下面给出的是一些其他常用的分区方法:
    - ***fuzzy k-means***: 该方法允许每个数据点依照不同的隶属度属于不同的类.
    - 相应的目标函数为使得下式最小化

$$\widehat{\mathcal{S}} = \underset{\mathcal{S}}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{i \in \mathcal{S}_j} \boldsymbol{u}_{i,j} \left\| \boldsymbol{x}_i - \boldsymbol{\mu}_j \right\|^2$$

    - $\boldsymbol{u}_{i,j}$ 是隶属度矩阵，它度量的是点 i (对应的位置为 $\boldsymbol{x}_i$) 处于类 j 中的隶属程度.

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - 下面给出的是一些其他常用的分区方法:
    - ***fuzzy k-means*** 算法

确定 隶属度矩阵的一个初始值 $\boldsymbol{u}_{i,j}^{(t)},\ t=1$

计算 类的重心:
$$\boldsymbol{\mu}_j = \frac{\sum_{i=1}^{n} \boldsymbol{u}_{i,j}^{(t)} \boldsymbol{x}_i}{\sum_{i=1}^{n} \boldsymbol{u}_{i,j}^{(t)}}$$

更新 $\boldsymbol{u}_{i,j}^{(t+1)}$:
$$\boldsymbol{u}_{i,j}^{(t+1)} = \frac{1}{\sum_{l=1}^{k} \left(\frac{\|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|}{\|\boldsymbol{x}_i - \boldsymbol{\mu}_l\|} \right)^{\frac{2}{t-1}}}$$

重复

计算, 更新

直至 收敛

Cluster Algorithms 聚类算法

- Partitioning Algorithms 分区算法
  - 分区聚类方法的一些缺点:
    - 类的数量 k 必须在开始时预先设定，这可能会导致最终出现有偏差的聚类结果.
    - 所列算法不能处理非凸形状的类.

Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - 聚集算法在实践中使用非常频繁.

1. 构建最佳分类
2. 计算距离矩阵 $\mathscr{D}$
3. 重复
4. 寻找距离最近的两类
5. 将这两类合并为一类
6. 计算新的类之间的距离，得到一个缩小的距离矩阵 $\mathscr{D}$
7. 直至 所有的类聚集在一起成为 $\mathscr{X}$

Cluster Algorithms 聚类算法

$n_p = \sum_{i=1}^{n} \mathrm{I}\left(\boldsymbol{x}_i \in P\right)$ 类 P 当中个体的数量.

- Hierarchical Algorithms, Agglomerative Techniqu
  - 如果两个对象 P 与 Q 合并在一起，则新类 $P+Q$ 与其它类 R 的距离定义为

$$d(R\ ,\ P+Q) = \delta_1 d(R\ ,\ P) + \delta_2 d(R\ ,\ Q) + \delta_3 d(P\ ,\ Q) + \delta_4 \left| d(R\ ,\ P) - d(R\ ,\ Q) \right|$$

类间距的计算

Name	δ_1	δ_2	δ_3	δ_4
Single linkage	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete linkage	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Average linkage (unweighted)	$\frac{1}{2}$	$\frac{1}{2}$	0	0
Average linkage (weighted)	$\frac{n_P}{n_P+n_Q}$	$\frac{n_Q}{n_P+n_Q}$	0	0
Centroid	$\frac{n_P}{n_P+n_Q}$	$\frac{n_Q}{n_P+n_Q}$	$-\frac{n_P n_Q}{\left(n_P+n_Q\right)^2}$	0
Median	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{4}$	0
Ward	$\frac{n_R+n_P}{n_R+n_P+n_Q}$	$\frac{n_R+n_Q}{n_R+n_P+n_Q}$	$-\frac{n_R}{n_R+n_P+n_Q}$	0

Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - 对于最常用的 Single 和 Complete 连接，我们有

1. 够建最佳分类
2. 计算距离矩阵 $\mathscr{D}$
3. 重复
4. 寻找 $\mathscr{D}$ 中 d 值 (个体 m 与 n 的距离) 的最小值 (Single 连接) 或者最大值 (Complete 连接)
5. 如果 m 和 n 不同类，将 m 和 n 的类合并在一起，并删除最小值
6. 直至 所有的类并为一类构成 $\mathscr{X}$ 或者 d 值超过了预设的水平

  - 在修正后的算法中，对原始距离矩阵进行线性搜索就足以进行聚类，而不是在每一步当中都要计算新的距离矩阵，这在实践当中更为有效.

Cluster Algorithms 聚类算法

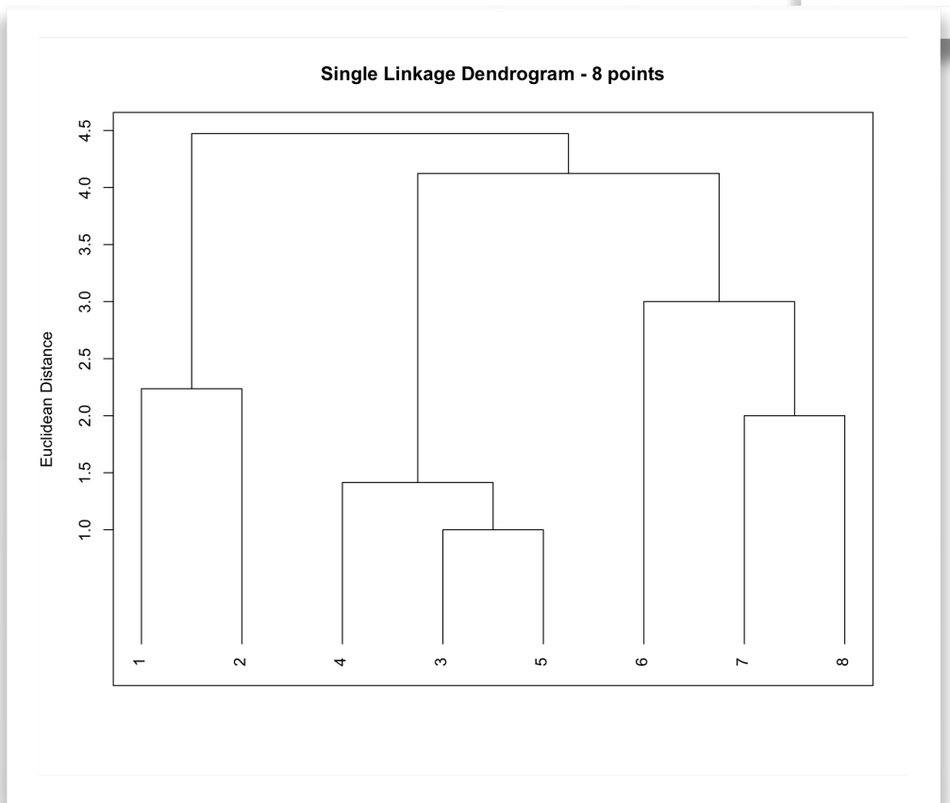
- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - 如果数据集当中观测结果众多，则聚类结果的可视化是必要的.
    - 谱系图 (***dendrogram***)：聚类序列的图形表示.
    - 横轴表示点代码.
    - 纵轴表示类间距.
    - 大的类间距表明差异较大群体的聚类.
    - 如果将谱系图在适当的水平切割，则相应的分支即对应于相应的聚类.

Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 8 个点的 Single linkage 算法

```
d = dist(eight, method = "euclidean", p = 2, diag = TRUE) # 欧氏距离矩阵
round(d, digits = 2)

s1 = hclust(d, method = "single") # cluster analysis with single linkage algorithm 聚类
graphics.off()
plot(s1, hang = -0.1, frame.plot = TRUE, ann = FALSE)
title(main = "Single Linkage Dendrogram - 8 points", ylab = "Euclidean Distance")
```

```
> round(d, digits = 2)
     1    2    3    4    5    6    7    8
1 0.00
2 2.24 0.00
3 6.32 5.00 0.00
4 7.62 6.40 1.41 0.00
5 6.08 4.47 1.00 2.24 0.00
6 9.22 8.94 5.00 4.12 6.00 0.00
7 5.83 6.08 4.24 4.47 5.00 3.61 0.00
8 7.62 8.06 5.83 5.66 6.71 3.00 2.00 0.00
```

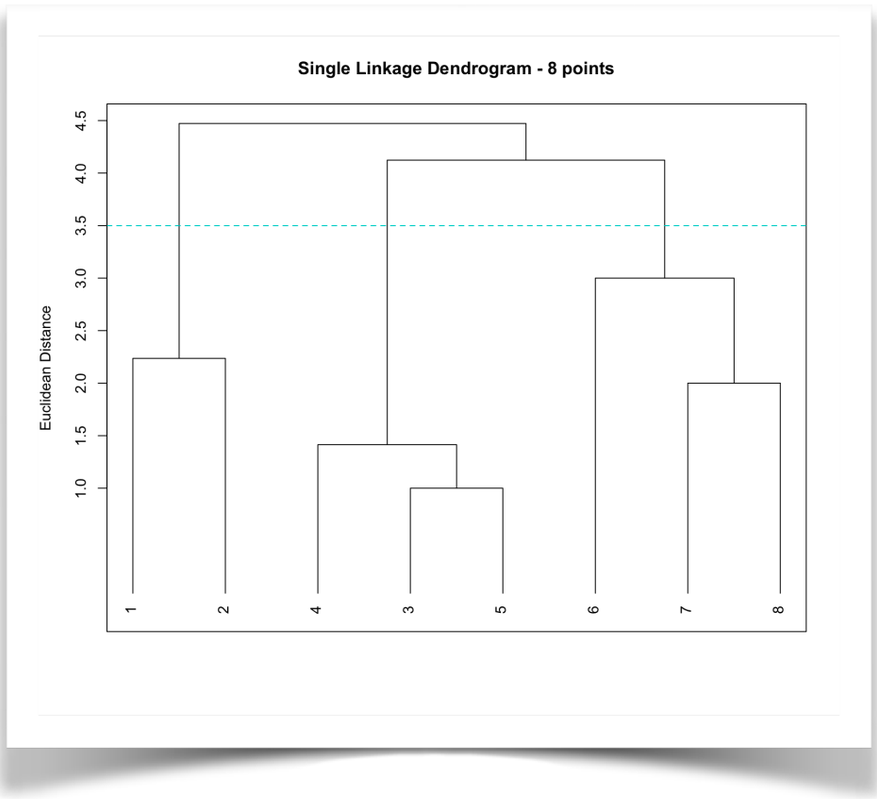


Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 8 个点的 Single linkage 算法
    - 如果我们在距离 3.5 处切谱系图，则聚为三类的结果为:

$$\{1,2\},\quad \{3,4,5\},\quad \{6,7,8\}$$

```
graphics.off()
plot(s1, hang = -0.1, frame.plot = TRUE, ann = FALSE)
title(main = "Single Linkage Dendrogram - 8 points", ylab = "Euclidean Distance")
abline(h = 3.5, lty = 2, col = 'cyan3') # cut the tree at distance 3.5
```



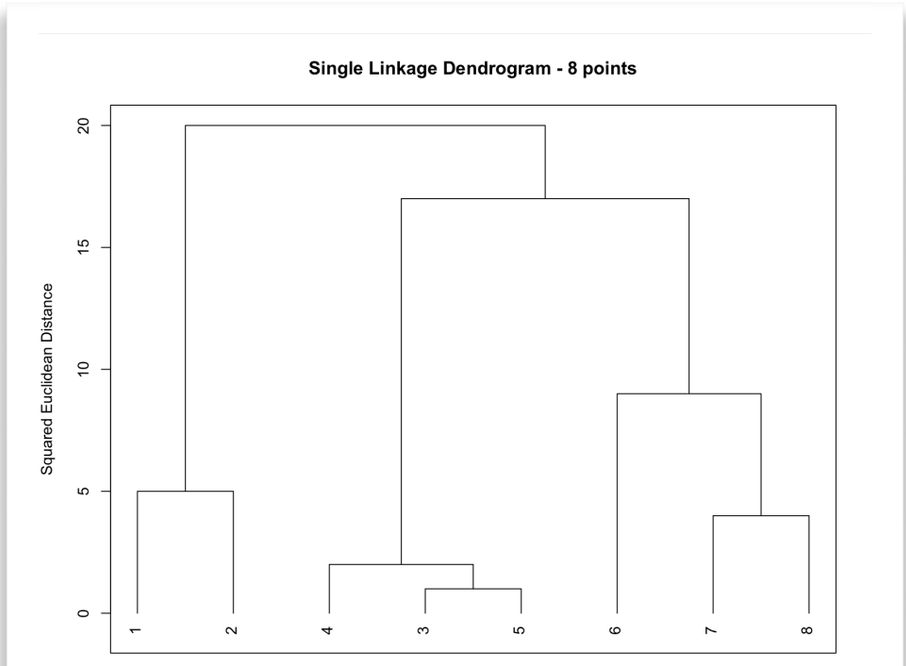
Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 8 个点的 Single linkage 算法

```
dd = d^2 # 欧氏距离平方的矩阵
round(dd, digits = 2)
```

```
s2 = hclust(dd, method = "single") # cluster analysis with single linkage algorithm 聚类分析-最短距离
# The dendrogram for the 8-point example, Single linkage algorithm.
graphics.off()
plot(s2, hang = -0.1, frame.plot = TRUE, ann = FALSE)
title(main = "Single Linkage Dendrogram - 8 points", ylab = "Squared Euclidean Distance")
```

```
> round(dd, digits = 2)
   1  2  3  4  5  6  7  8
1  0
2  5  0
3 40 25  0
4 58 41  2  0
5 37 20  1  5  0
6 85 80 25 17 36  0
7 34 37 18 20 25 13  0
8 58 65 34 32 45  9  4  0
```



Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***single linkage*** 算法定义类间距为两类的点之间距离的最小值

$$d(R\ ,\ P+Q)=\min\left\{d(R\ ,\ P)\ ,\ d(R\ ,\ Q)\right\}$$

  - ***complete linkage*** 算法定义类间距为两类的点之间距离的最大值

$$d(R\ ,\ P+Q)=\max\left\{d(R\ ,\ P)\ ,\ d(R\ ,\ Q)\right\}$$

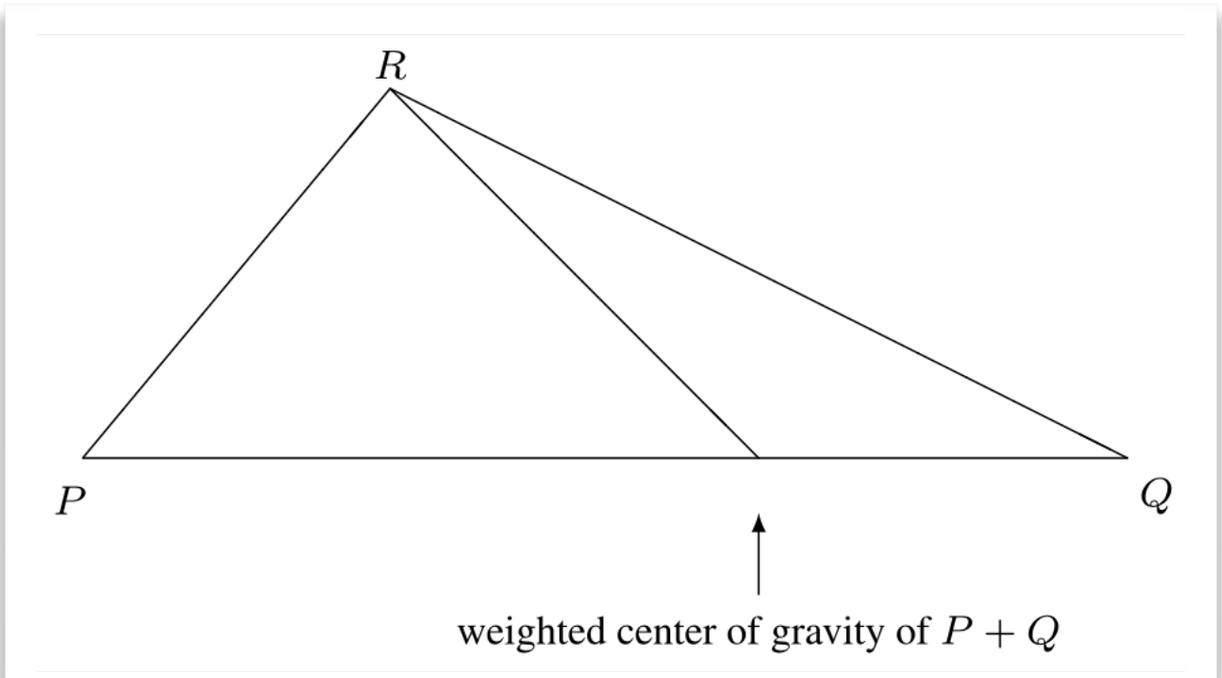
  - ***average linkage*** 算法 (加权或不加权) 定义类间距为上述两种算法的折中

$$d(R\ ,\ P+Q)=\frac{n_P}{n_P+n_Q}d(R\ ,\ P)+\frac{n_Q}{n_P+n_Q}d(R\ ,\ Q)$$

Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***centroid*** 算法采用的是 R 至 P 与 Q 的加权重心之间的几何距离

$$d(R\ ,\ P+Q)=\frac{n_P}{n_P+n_Q}d(R\ ,\ P)+\frac{n_Q}{n_P+n_Q}d(R\ ,\ Q)-\frac{n_Pn_Q}{n_P+n_Q}d(P\ ,\ Q)$$



Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Ward clustering*** 算法类间距的计算采用

$$d(R\ ,\ P+Q)=\frac{n_P}{n_P+n_Q}d(R\ ,\ P)+\frac{n_Q}{n_P+n_Q}d(R\ ,\ Q)-\frac{n_R}{n_R+n_P+n_Q}d(P\ ,\ Q)$$

    - Ward 算法并未采用将距离最短的类聚在一起
    - 而是将差异性度量 (measure of heterogeneity) 增加最小的类聚在一起

$$I_R=\frac{1}{n_R}\sum_{i=1}^{n_R}d^2\left(\boldsymbol{x}_i\ ,\ \bar{\boldsymbol{x}}_R\right)$$

类 R 的重心 (均值)

    - Ward 方法的目的是将使得类内部的变化尽可能小的类聚在一起: 聚类的结果会使各类当中的个体尽可能的相似

Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 从货币数据集当中随机选择 20 个观测值，利用欧氏距离进行 Ward 聚类.

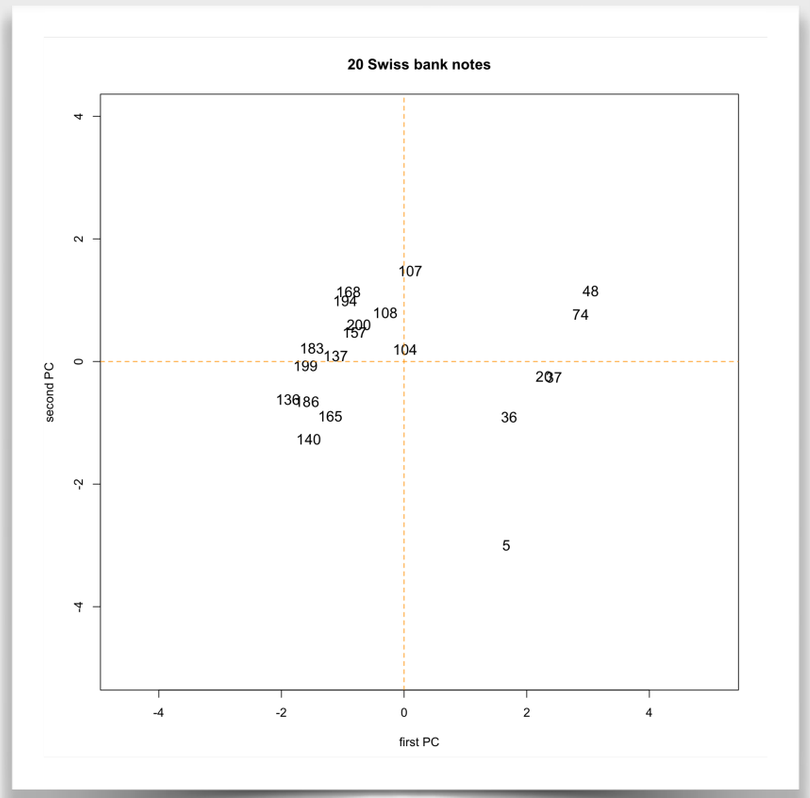
```
library(mclust)
data = banknote
set.seed(3)
xx = data[sample(1:nrow(data), 20, replace = F), ]  # sample of 20 randomly
xx = xx[, 2:7]
xx
```

```
> xx
    Length  Left Right Bottom  Top Diagonal
5    215.0 129.6 129.7   10.4  7.7    141.8
186  214.7 130.4 130.0   11.5 10.7    139.4
140  214.5 130.5 130.2   11.8 10.2    139.6
36   214.6 130.2 130.2    9.4  9.7    141.8
199  214.7 130.7 130.8   11.2 11.2    139.4
107  215.3 130.3 130.1    9.3 12.1    140.2
136  214.8 130.1 130.1   11.9 11.1    139.5
20   214.7 130.2 129.9    8.6 10.0    141.9
74   214.4 129.9 129.6    7.5 10.5    141.8
183  215.0 130.5 130.1   11.0 11.4    139.3
168  215.6 130.4 130.1    9.6 11.2    138.6
48   214.8 129.9 129.7    7.3 10.9    142.0
104  215.0 130.4 130.6    9.9 10.9    140.3
194  215.0 130.5 130.3    9.6 11.0    138.5
37   215.5 130.3 130.0    8.4  9.7    141.8
157  214.2 129.7 129.6   10.3 11.4    139.5
108  214.8 130.1 130.4    9.8 11.5    139.9
200  214.3 129.9 129.9   10.2 11.5    139.6
165  214.3 130.3 130.0   11.4 10.5    139.8
137  214.6 129.8 130.2   10.7 11.1    139.4
```

Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 从货币数据集当中随机选择 20 个观测值，利用欧氏距离进行 Ward 聚类.

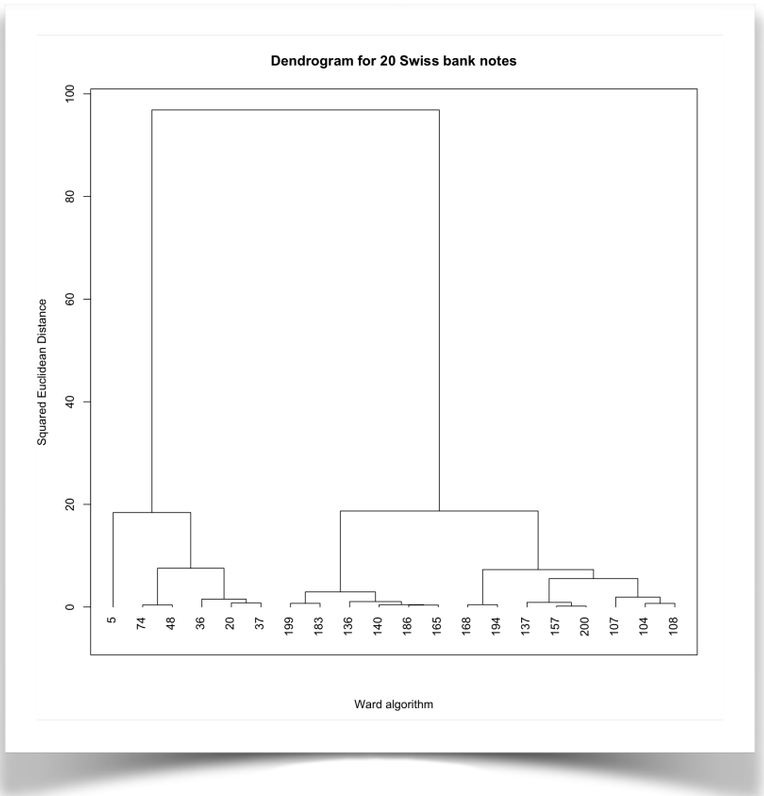
```
## PCA 主成分分析
mean = as.vector(colMeans(xx)) # mean vector
m = matrix(mean, nrow(xx), NROW(mean), byrow = T)
x = xx - m # centering
eig = eigen(cov(x)) # spectral decomposition
eva = eig$values # eigenvalues
eve = eig$vectors # eigenvectors
xm = as.matrix(x)
y = xm %*% eve # PC values
ym = y[, 1:2] # first two PCs

## Plot 1: PCA
graphics.off()
plot(ym, type = "n", xlab = "first PC", ylab = "second PC", main = "20 Swiss bank notes",
     ylim = c(-5, 4), xlim = c(-4, 4.5), asp = 1)
text(ym[, 1], ym[, 2], rownames(xx), cex = 1.2)
abline(h = 0, v = 0, lty = 2, col = 'orange')
```



Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 从货币数据集当中随机选择 20 个观测值，利用欧氏距离进行 Ward 聚类.

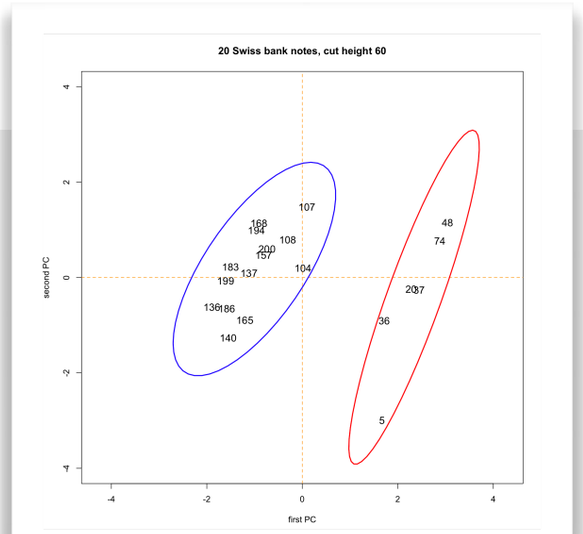
```
## Plot 2: Dendrogram for the 20 bank notes after applying the Ward algorithm
d = dist(xx, method = "euclidean", p = 2) # euclidean distance matrix
dd = d^2 # squared euclidean distance matrix
w = hclust(dd, method = "ward.D") # cluster analysis with ward algorithm
graphics.off()
plot(w, hang = -0.1, frame.plot = TRUE, ann = FALSE)
title(main = "Dendrogram for 20 Swiss bank notes", ylab = "Squared Euclidean Distance", xlab = "Ward algorithm")
```



Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 从货币数据集当中随机选择 20 个观测值，利用欧氏距离进行 Ward 聚类.

```
## Plot 3: PCA with clusters
library(car)
graphics.off()
groups = cutree(w, h = 60)
merg = matrix(c(ym, as.matrix(groups)), nrow = 20, ncol = 3)
merg = merg[sort.list(merg[, 3]), ]
merg1  = merg[1:6, 1:2]
merg2  = merg[7:20, 1:2]
plot(ym, type = "n", xlab = "first PC", ylab = "second PC", main = "20 Swiss bank notes, cut height 60",
    ylim = c(-4, 4), xlim = c(-4, 4), asp = 1)
abline(h = 0, v = 0, lty = 2, col = 'orange')
text(ym[, 1], ym[, 2], rownames(xx), cex = 1.2)
dataEllipse(x = merg1[, 1], y = merg1[, 2], center.pch = 0, col = "red", plot.points = F, add = T, levels = 0.85)
dataEllipse(x = merg2[, 1], y = merg2[, 2], center.pch = 0, col = "blue", plot.points = F, add = T, levels = 0.95)
```



Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 法国食品消费数据集.

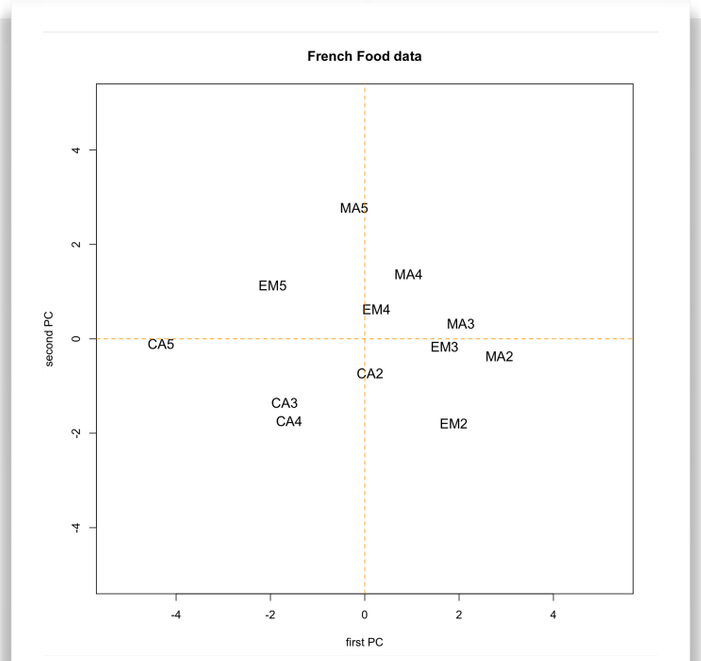
```
## Load data
setwd("~/Desktop/2023_Applied Multivariate Statistical Analysis/R Codes with data/Data")
library(readr)
fooddat = read_csv("food.csv")
food = as.data.frame(fooddat[, -1]) # delete the first column (types of families)
rownames(food) = c("MA2", "EM2", "CA2", "MA3", "EM3", "CA3", "MA4", "EM4", "CA4", "MA5", "EM5", "CA5") # define types of
families
f = scale(food) # standardize variables
round(f, digits = 3)
```

```
> round(f, digits = 3)
     bread  veget  fruit   meat  poult   milk   wine
MA2 -1.070 -1.607 -0.915 -1.136 -1.111 -0.950  0.814
EM2 -1.434 -0.914 -0.709 -0.909 -0.946 -1.018 -1.541
CA2 -0.697  0.185  0.345  0.155  0.496 -1.052  0.897
MA3 -0.380 -0.893 -0.993 -0.960 -1.038 -0.292  0.535
EM3 -0.566 -0.655 -0.660 -0.975 -0.982 -0.335 -0.078
CA3 -0.081  0.587  1.115  1.158  1.382 -0.984 -0.384
MA4  0.815 -0.381 -0.836 -0.674 -0.662  0.476  0.535
EM4  0.124 -0.174 -0.127 -0.078 -0.165  0.356  0.661
CA4 -0.576  0.301  0.703  1.211  1.386 -0.463 -1.206
MA5  1.944  0.233 -0.497 -0.098 -0.177  1.168  1.636
EM5  1.282  1.390  0.260  0.428  0.360  1.364 -0.691
CA5  0.638  1.929  2.314  1.878  1.458  1.731 -1.178
attr(,"scaled:center")
    bread     veget     fruit      meat     poult      milk      wine
 446.6667  732.0000  505.0000 1886.7500  803.1667  358.2500  368.5833
attr(,"scaled:scale")
    bread     veget     fruit      meat     poult      milk      wine
107.14759 189.18005 165.09226 395.75041 249.56064 117.12707  71.78181
```

Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 法国食品消费数据集.

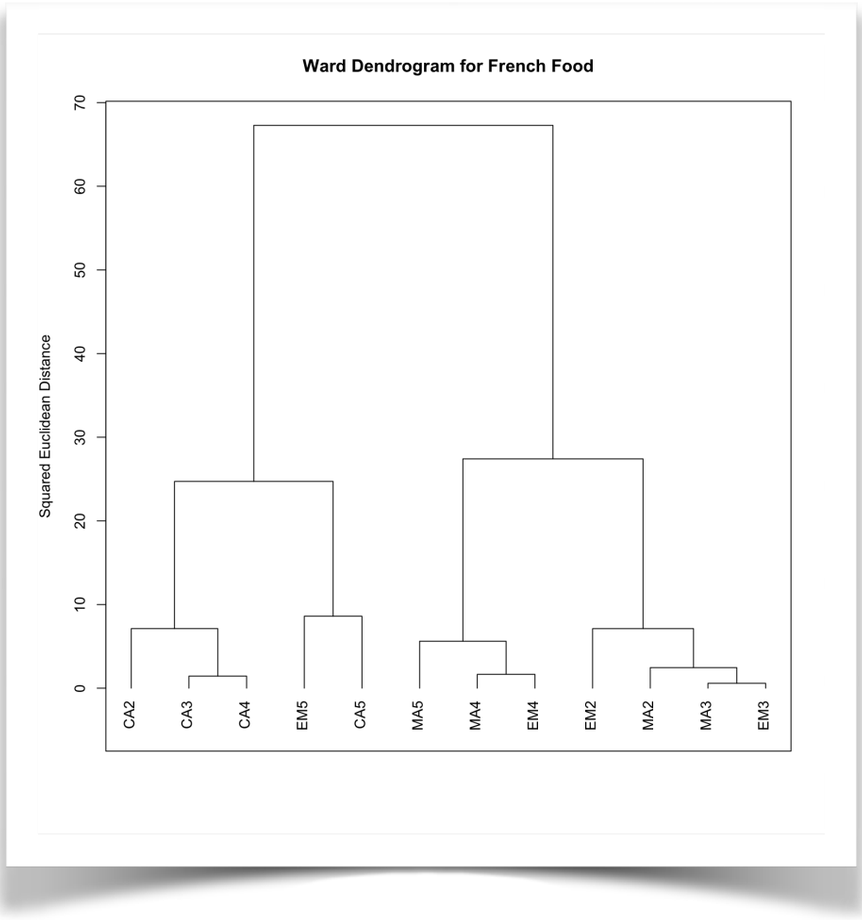
```
## PCA
mean = as.vector(colMeans(f))
m = matrix(mean, nrow(f), NROW(mean), byrow = T)
x = f - m
eig = eigen(cov(x)) # spectral decomposition
eva = eig$values
eve = eig$vectors
xm = as.matrix(x)
y = xm %*% eve
ym = y[, 1:2] # first two eigenvectors

## Plot 1: PCA
plot(ym, type = "n", xlab = "first PC", ylab = "second PC", main = "French Food data", ylim = c(-5, 5), xlim = c(-5, 5), asp = 1)
text(ym[, 1], ym[, 2], rownames(f), cex = 1.2)
abline(h = 0, v = 0, lty = 2, col = 'orange')
```



Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 法国食品消费数据集.

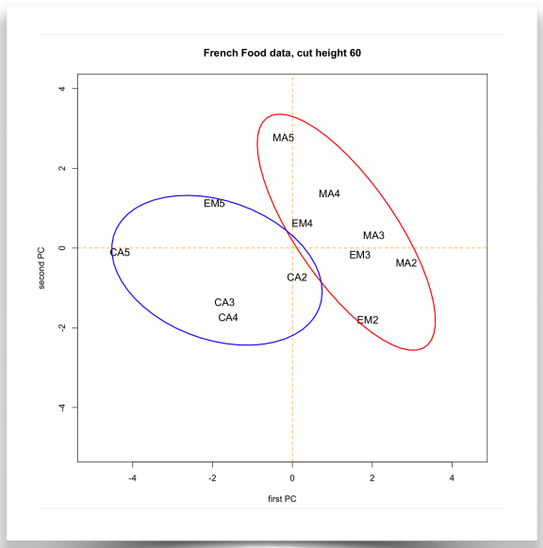
```
## Plot 2: Dendrogram for the standardized food.dat after Ward algorithm
d = dist(f, "euclidean", p = 2) # euclidean distance matrix
dd = d^2 # squared euclidean distance matrix
w = hclust(dd, method = "ward.D") # cluster analysis with ward algorithm
graphics.off()
plot(w, hang = -0.1, frame.plot = TRUE, ann = FALSE)
title(main = "Ward Dendrogram for French Food", ylab = "Squared Euclidean Distance")
```



Cluster Algorithms 聚类算法

- Hierarchical Algorithms, Agglomerative Techniques 分层算法，聚集方法
  - ***Example***: 法国食品消费数据集.

```
## Plot 3: PCA with clusters
groups = cutree(w, h = 60)
merg = matrix(c(ym, as.matrix(groups)), nrow = 12, ncol = 3)
merg = merg[sort.list(merg[, 3]), ]
merg1 = merg[1:7, 1:2]
merg2 = merg[8:12, 1:2]
graphics.off()
plot(ym, type = "n", xlab = "first PC", ylab = "second PC", main = "French Food data, cut height 60",
    ylim = c(-5, 4), xlim = c(-5, 4.5), asp = 1)
abline(h = 0, v = 0, lty = 2, col = 'orange')
text(ym[, 1], ym[, 2], rownames(f), cex = 1.2)
dataEllipse(x = merg1[, 1], y = merg1[, 2], center.pch = 0, col = "red", plot.points = F, add = T, levels = 0.8)
dataEllipse(x = merg2[, 1], y = merg2[, 2], center.pch = 0, col = "blue", plot.points = F, add = T, levels = 0.65)
```



Adaptive Weights Clustering 自适应权重聚类

- 一种基于非参数思想的聚类方法，通过同质性检验进行分离从而找到聚类结构的方法。
  - 在此介绍 Efimov 等人 (2017) 给出的方法.
  - 这是一种完全自适应的方法，不需要指定聚类的数量或者它们的结构.
  - 聚类结果对噪声和异常值不敏感，该方法能够实现对具有尖锐边缘或流形结构的不同聚类.

Adaptive Weights Clustering 自适应权重聚类

- 观测值：$\{\boldsymbol{X}_i\}_{i=1}^{n} \subset \mathbb{R}^p$.

  p 可以很大甚至无穷

  - 计算任意两点 $\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)$ 的距离 $d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)$.

    例如欧氏距离 $\left\|\boldsymbol{X}_i - \boldsymbol{X}_j\right\|$

  - 该方法仅依赖于对距离（或相似）矩阵 $\mathscr{D} = \left(d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)\right)_{i,\,j=1}^{n}$ 的操作.

  - 引入权重矩阵 $W = \left(w_{ij}\right)$, $i, j = 1, 2, \ldots, n$.

    $$w_{ij} = \begin{cases} 1, & \boldsymbol{X}_i, \boldsymbol{X}_j \text{ 属于同一类} \\ 0, & \boldsymbol{X}_i, \boldsymbol{X}_j \text{ 属于不同类} \end{cases}$$

  - 权重矩阵 $W = \left(w_{ij}\right)$ 是对称阵. 每个 1 对应的块描述了一个类.

Adaptive Weights Clustering 自适应权重聚类

- 观测值：$\{\boldsymbol{X}_i\}_{i=1}^{n} \subset \mathbb{R}^p$.

p 可以很大甚至无穷

```
eight = cbind(c(-3, -2, -2, -2, 1, 1, 2, 4), c(0, 4, -1, -2, 4, 2, -4, -3)) # 输入数据
eight = eight[c(8, 7, 3, 1, 4, 2, 6, 5), ] # 数据集重新排序
d = dist(eight, method = "euclidean", diag = TRUE, upper = TRUE, p = 2) # 计算欧氏距离
D = as.matrix(d) # 距离矩阵
nr = nrow(D) # 矩阵 D 的行数
nc = ncol(D) # 矩阵 D 的列数
W = matrix(0, nrow = nr, ncol = nc) # 权重矩阵的初始值
for (i in 1:nr){
  for (j in 1:nc) {
    if (D[i, j] < 4) W[i, j] = 1
  }
} # 距离小于 4 的归为一类
round(D, digits = 2) # 距离矩阵
W # 权重矩阵
```

```
> round(D, digits = 2) # 距离矩阵
     1    2    3    4    5    6    7    8
1 0.00 2.24 6.32 7.62 6.08 9.22 5.83 7.62
2 2.24 0.00 5.00 6.40 4.47 8.94 6.08 8.06
3 6.32 5.00 0.00 1.41 1.00 5.00 4.24 5.83
4 7.62 6.40 1.41 0.00 2.24 4.12 4.47 5.66
5 6.08 4.47 1.00 2.24 0.00 6.00 5.00 6.71
6 9.22 8.94 5.00 4.12 6.00 0.00 3.61 3.00
7 5.83 6.08 4.24 4.47 5.00 3.61 0.00 2.00
8 7.62 8.06 5.83 5.66 6.71 3.00 2.00 0.00
```

```
> W # 权重矩阵
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    1    0    0    0    0    0    0
[2,]    1    1    0    0    0    0    0    0
[3,]    0    0    1    1    1    0    0    0
[4,]    0    0    1    1    1    0    0    0
[5,]    0    0    1    1    1    0    0    0
[6,]    0    0    0    0    0    1    1    1
[7,]    0    0    0    0    0    1    1    1
[8,]    0    0    0    0    0    1    1    1
```

Adaptive Weights Clustering 自适应权重聚类

- 观测值：$\{\boldsymbol{X}_i\}_{i=1}^{n} \subset \mathbb{R}^p$.

  p 可以很大甚至无穷

  - 计算任意两点 $\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)$ 的距离 $d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)$.

    例如欧氏距离 $\left\|\boldsymbol{X}_i - \boldsymbol{X}_j\right\|$

  - 该方法仅依赖于对距离（或相似）矩阵 $\mathscr{D} = \left(d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)\right)_{i,\,j=1}^{n}$ 的操作.

  - 引入权重矩阵 $W = \left(w_{ij}\right)$, $i, j = 1, 2, \ldots, n$.

    $$w_{ij} = \begin{cases} 1, & \boldsymbol{X}_i, \boldsymbol{X}_j \text{属于同一类} \\ 0, & \boldsymbol{X}_i, \boldsymbol{X}_j \text{属于不同类} \end{cases}$$

  - 权重矩阵 $W = \left(w_{ij}\right)$ 是对称阵. 每个 1 对应的块描述了一个类.

  - 固定 i，相应的类 C_i 则由 j 取遍所有值时正的权重 $\left(w_{ij}\right)$ 对应的集合所确定.

```
> W # 权重矩阵
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    1    0    0    0    0    0    0
[2,]    1    1    0    0    0    0    0    0
[3,]    0    0    1    1    1    0    0    0
[4,]    0    0    1    1    1    0    0    0
[5,]    0    0    1    1    1    0    0    0
[6,]    0    0    0    0    0    1    1    1
[7,]    0    0    0    0    0    1    1    1
[8,]    0    0    0    0    0    1    1    1
```

Adaptive Weights Clustering 自适应权重聚类

- 该方法尝试从数据中恢复权重 w_{ij}，这就是名称 “自适应权重聚类” 的由来.
  - 它从非常局部的聚类结构 $C_i^{(0)}$ 开始，即，开始时正的权重 $w_{ij}^{(0)}$ 只局限于点 $\boldsymbol{X}_i$ 按照距离 $d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)$ 确定的临近点 $\boldsymbol{X}_j$.
  - 在第 $k \geq 1$ 步，通过检验 $C_i^{(k-1)}$ 和 $C_j^{(k-1)}$ 之间是否“无缝隙”来重新计算权重 $w_{ij}^{(k)}$，其中 $C_i^{(k-1)}$ 和 $C_j^{(k-1)}$ 分别是第 $k-1$ 步当中点 $\boldsymbol{x}_i$ 与 $\boldsymbol{x}_j$ 的局部聚类.
  - 我们只检查 $d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right) \leq h_k$ 的邻近点 $\boldsymbol{X}_i$，$\boldsymbol{X}_j$，但是，对每一个固定点 $\boldsymbol{X}_i$，其局部参数 h_k 和邻近点 $\boldsymbol{X}_j$ 的数量会随着每一步骤而增大.
  - 最终的权重矩阵 W 用于确定最后的聚类结果.
  - 该方法的核心是如何重新计算权重 $w_{ij}^{(k)}$.

Adaptive Weights Clustering 自适应权重聚类

- 半径序列
  - 观察半径的增长序列 $h_1 \le h_2 \le \cdots \le h_K$，它决定了算法从考虑局部结构到大规模对象的发展速度.
  - h_k 的每一个值可以看作是第 k 步当中方法的分辨率(标度).
  - 出于理论上的原因，规则必须确保第 k 步当中，每个 $\boldsymbol{X}_i$ 筛选出的邻近点的平均数量，当 $k \ge 1$ 时最多只能以指数形式增长.

Adaptive Weights Clustering 自适应权重聚类

- 初始化权重
  - 在初始化的步骤中，我们将每个点与其最近的 n_0 个邻近点连接起来：

$$w_{ij}^{(0)} = \mathrm{I}\left[d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right) \leq \max\left\{h_0\left(\boldsymbol{X}_i\right), h_0\left(\boldsymbol{X}_j\right)\right\}\right]$$

  - $h_0\left(\boldsymbol{X}_i\right)$ 表示点 $\boldsymbol{X}_i$ 与其最近的 n_0 个邻近点的距离.
  - 我们默认选取 $n_0 = 2p + 2$.

Adaptive Weights Clustering 自适应权重聚类

- 第 k 步时的更新
  - 假设前 $k-1$ 步的迭代已完成. 从而对每一个点 $\boldsymbol{X}_i$ 我们得到权重的集合

$$\left\{w_{ij}^{(k-1)},\ j=1,\ 2,\ \ldots,\ n\right\}$$

  - 这些权重给出了与 $\boldsymbol{X}_i$ 相关的局部“聚类”结果: 即，使得权重 $w_{ij}^{(k-1)}$ 取值为正的 $\boldsymbol{X}_j$ 构成的聚类.

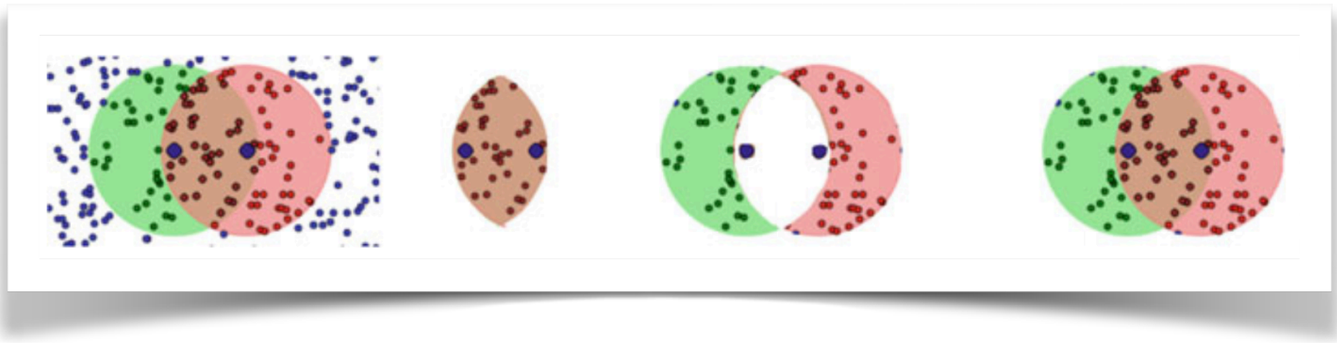
$$\boldsymbol{X}_j \in B\left(\boldsymbol{X}_i,\ h_{k-1}\right)=\left\{\boldsymbol{x}\ :\ d\left(\boldsymbol{X}_i,\ \boldsymbol{x}\right)\le h_{k-1}\right\}$$

以 $\boldsymbol{X}_i$ 为中心、h_{k-1} 为半径的球体

  - 在接下来的第 k 步中，我们选择一个更大的半径 h_k，并利用上一步的结果重新计算权重 $w_{ij}^{(k)}$.

Adaptive Weights Clustering 自适应权重聚类

- 第 k 步时的更新
  - 建立在 $w_{ij}^{(k)}$ 定义背后的基本思想是：对满足 $d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right) \leq h_k$ 的每一对 i, j，相应的类是否足够分离，或者它们可以聚成一类.
  - 通过比较点 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_j$ 对应的两个类的并集以及交集当中数据的密度进行检验.
  - 将 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_j$ 看作固定的两个点，利用上一步得到的权重 $w_{ij}^{(k-1)}$ 计算检验统计量 $T_{ij}^{(k)}$.
  - 正式的定义涉及两个球体 $B\left(\boldsymbol{X}_i, h_{k-1}\right)$ 和 $B\left(\boldsymbol{X}_j, h_{k-1}\right)$ 的交集与并集当中的加权经验密度.



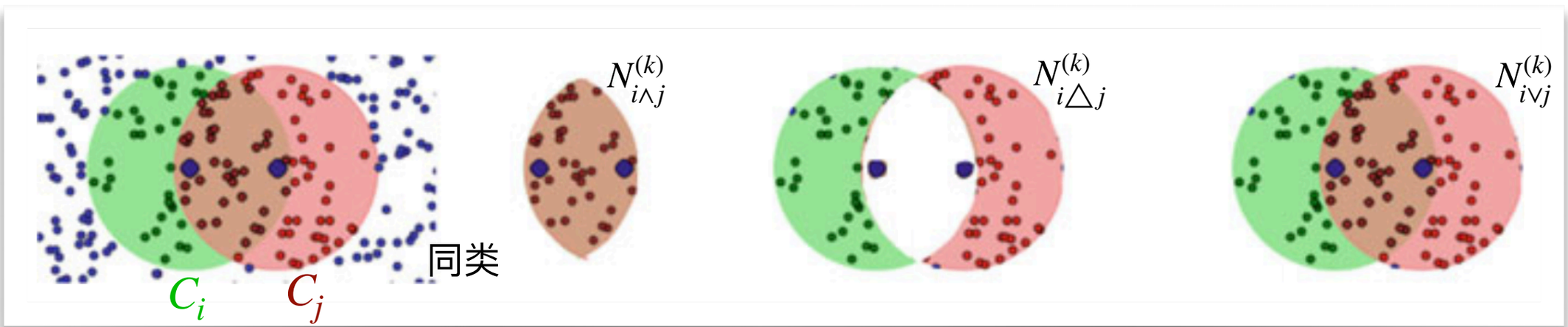
Adaptive Weights Clustering 自适应权重聚类

- 第 k 步时的更新
  - 建立在 $w_{ij}^{(k)}$ 定义背后的基本思想是：对满足 $d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right) \leq h_k$ 的每一对 i, j，相应的类是否足够分离，或者它们可以聚成一类.

交集的经验密度： $N_{i\wedge j}^{(k)} = \sum_{l\neq i,\, j} w_{il}^{(k-1)} w_{jl}^{(k-1)}$

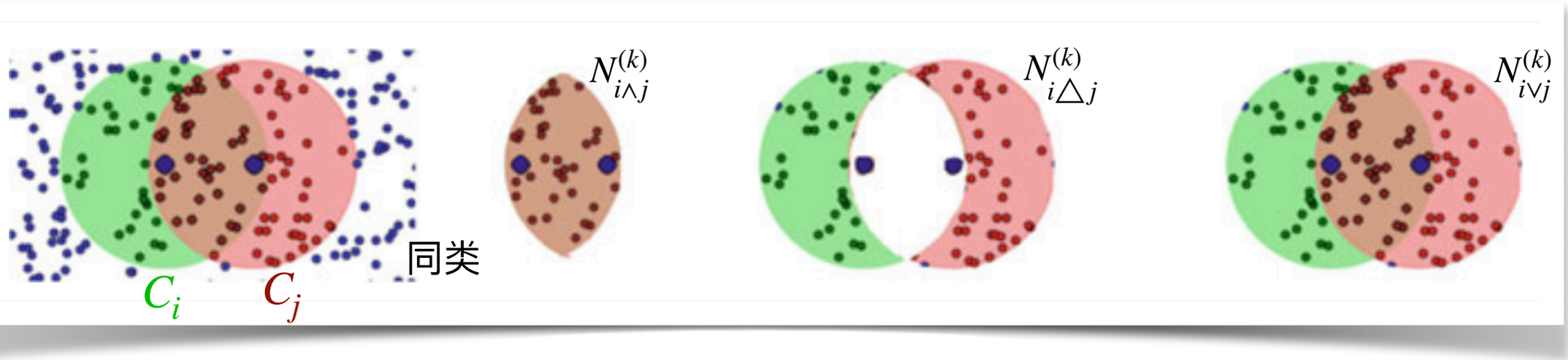
补集的经验密度： $N_{i\triangle j}^{(k)} = \sum_{l\neq i,\, j} \left\{ w_{il}^{(k-1)}\, \mathrm{I}\left[\boldsymbol{X}_l \notin B\left(\boldsymbol{X}_j, h_{k-1}\right)\right] + w_{jl}^{(k-1)}\, \mathrm{I}\left[\boldsymbol{X}_l \notin B\left(\boldsymbol{X}_i, h_{k-1}\right)\right] \right\}$

并集的经验密度： $N_{i\vee j}^{(k)} = N_{i\wedge j}^{(k)} + N_{i\triangle j}^{(k)}$



Adaptive Weights Clustering 自适应权重聚类

- 第 k 步时的更新
  - 为度量间隙，考虑两个密度之比: $\tilde{\theta}_{ij}^{(k)} = \dfrac{N_{i\wedge j}^{(k)}}{N_{i\vee j}^{(k)}}.$
  - $\tilde{\theta}_{ij}^{(k)}$ 可以看作是对两个局部区域 C_i 与 C_j 交集的平均密度与并集的平均密度之比 θ_{ij} 的一个估计.
  - 在局部同类的情形下，我们可以假定两个球体并集的密度几乎是一个常数.



Adaptive Weights Clustering 自适应权重聚类

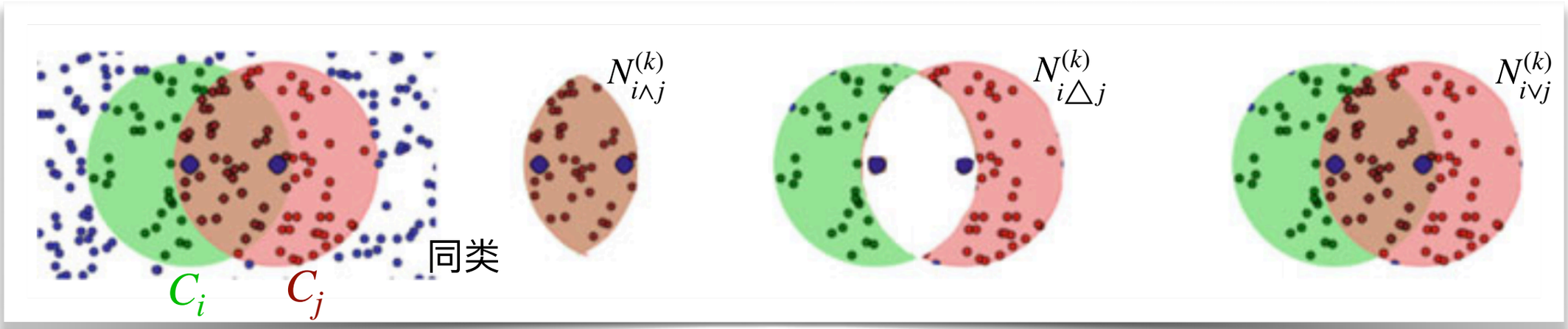
- 第 k 步时的更新
  - 此时，估计量 $\tilde{\theta}_{ij}^{(k)}$ 的值应接近于两个球体交集的体积与并集的体积之比：

$$\tilde{\theta}_{ij}^{(k)} \approx q_{ij}^{(k)} = \frac{V_{\cap}\left(d_{ij}\,,\,h_{k-1}\right)}{2V\left(h_{k-1}\right) - V_{\cap}\left(d_{ij}\,,\,h_{k-1}\right)}$$

半径为 h_{k-1}、球心距离为 $d = d\left(\boldsymbol{X}_i\,,\,\boldsymbol{X}_j\right)$ 的两个球体交集部分的体积

半径为 h_{k-1} 的球体体积

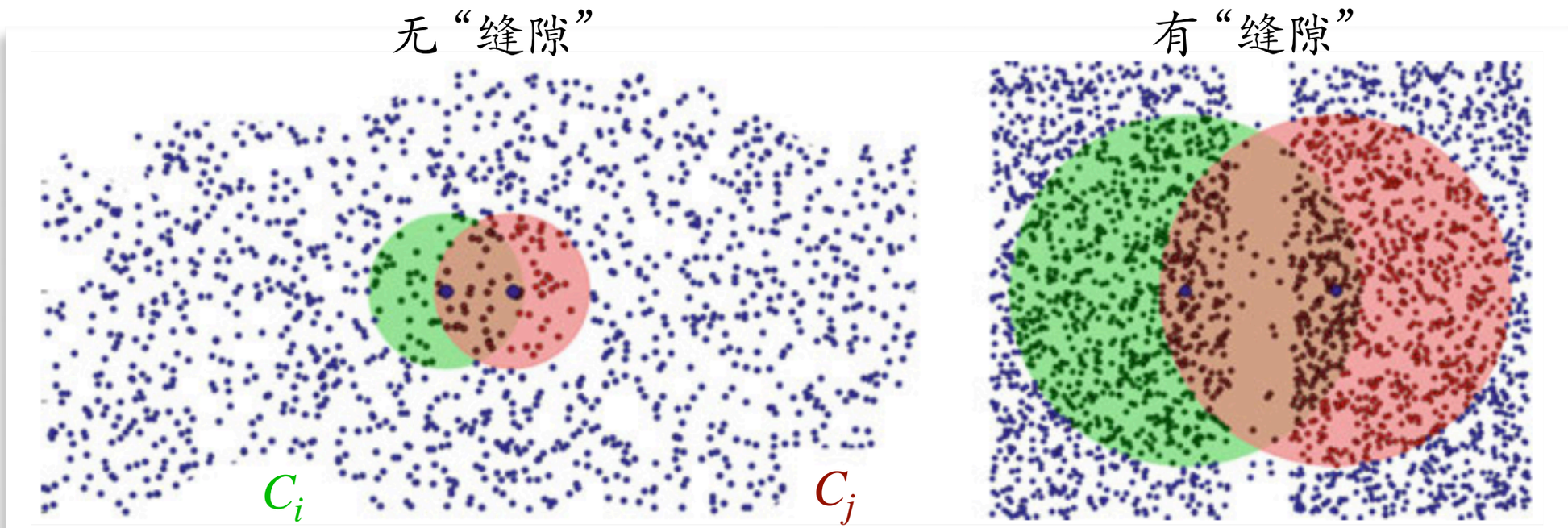
  - 新的 $w_{ij}^{(k)}$ 的值可用来检验 $\begin{cases} H_0: & \boldsymbol{X}_i \text{ 与 } \boldsymbol{X}_j \text{ 无“缝隙”} \\ H_1: & \boldsymbol{X}_i \text{ 与 } \boldsymbol{X}_j \text{ 存在显著“缝隙”} \end{cases}$



Adaptive Weights Clustering 自适应权重聚类

- 第 k 步时的更新
  - 表示为统计推断问题:

$$\begin{cases} H_0: \ \tilde{\theta}_{ij}^{(k)} > q_{ij}^{(k)} & (\text{无“缝隙”}) \\ H_1: \ \tilde{\theta}_{ij}^{(k)} \le q_{ij}^{(k)} & (\text{有显著“缝隙”}) \end{cases}$$



Adaptive Weights Clustering 自适应权重聚类

- 第 k 步时的更新
  - 统计量： $T_{ij}^{(k)} = N_{i \vee j}^{(k)} \cdot KL\left(\tilde{\theta}_{ij}^{(k)}, q_{ij}^{(k)}\right) \cdot \left[\mathrm{I}\left(\tilde{\theta}_{ij}^{(k)} \le q_{ij}^{(k)}\right) - \mathrm{I}\left(\tilde{\theta}_{ij}^{(k)} > q_{ij}^{(k)}\right)\right]$

$$KL(\theta, \eta) = \theta \log \frac{\theta}{\eta} + (1-\theta)\log \frac{1-\theta}{1-\eta}$$

  - $KL(\theta, \eta)$ 是参数为 θ 和 η 的两个伯努利定律之间的 Kullback-Leibler (KL) 分叉.
  - 最后，对所有满足 $d\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right) \le h_k$ 的两个点 $\boldsymbol{X}_i, \boldsymbol{X}_j$，我们更新权重 $w_{ij}^{(k)}$ 如下：

$$w_{ij}^{(k)} = \mathrm{I}\left(T_{ij}^{(k)} \le \lambda\right)$$

  - 统计量 $T_{ij}^{(k)}$ 由全局常数 λ 缩放，它是该方法唯一的调整参数，对方法的表现有着重要影响.

λ 过大会导致不同类聚在一起

λ 过小则相反，会导致出现人为的聚类

Adaptive Weights Clustering 自适应权重聚类

- 参数调整
  - 基于最终权重 w_{ij}^K 的总和给出的有效聚类大小，启发我们给出 λ 的选择.
  - 设 $w_{ij}^K(\lambda)$ 是通过参数为 λ 的过程得到的最终权重的集合. 定义

$$S(\lambda) = \sum_{i,\, j=1}^{n} w_{ij}^K(\lambda)$$

  - 随着 λ 的增加会得到更大的同类块，因此，$S(\lambda)$ 的值也会更大.
  - 建议: 观察 $S(\lambda)$ 的图形，选取图形产生巨大跳跃之前对应的一个点作为 λ 值.
  - 在聚类结构复杂的情况下，可能会观察到若干个跳跃点，每个跳跃点都对应有相应的 λ 值.
  - 在这种情形下，应检查所有的 λ 值并与获得的聚类结果进行比较.

Adaptive Weights Clustering 自适应权重聚类

- 自适应权重聚类算法 AWC

1. 选定半径序列 $h_1 \le h_2 \le \cdots \le h_K$
2. 初始化权重: $w_{ij}^{(0)} = \mathrm{I}\left[d\left(\boldsymbol{X}_i\,,\boldsymbol{X}_j\right) \le \max\left(h_0\left(\boldsymbol{X}_i\right)\,,h_0\left(\boldsymbol{X}_j\right)\right)\right]$
3. 第 k 步的更新
4. 计算 $T_{ij}^{(k)} = N_{i\vee j}^{(k)} \cdot KL\left(\tilde{\theta}_{ij}^{(k)}\,,q_{ij}^{(k)}\right) \cdot \left[\mathrm{I}\left(\tilde{\theta}_{ij}^{(k)} \le q_{ij}^{(k)}\right) - \mathrm{I}\left(\tilde{\theta}_{ij}^{(k)} > q_{ij}^{(k)}\right)\right]$
5. $w_{ij}^{(k)} = \mathrm{I}\left[d\left(\boldsymbol{X}_i\,,\boldsymbol{X}_j\right) \le h_k\right] \cdot \mathrm{I}\left(T_{ij}^{(k)} \le \lambda\right)$
6. 重复进行 直至 $k = K$.

Spectral Clustering 谱聚类

- 谱聚类是基于图论的方法将数据点划分为具有相同结构的组 (类).
  - 所有方法都基于来自数据矩阵 $\mathcal{X}_{n\times p}$ 中的观测值 $\{\boldsymbol{x}_i\}_{i=1}^{p} \in \mathbb{R}^p$ 以及数据点之间的相似性:

$$d_{ij} = \frac{a_1 + \delta a_4}{a_1 + \delta a_4 + \lambda\left(a_2 + a_3\right)}$$

Name	δ	λ	Definition
Jaccard	0	1	$d_{ij} = \frac{a_1}{a_1 + a_2 + a_3}$
Tanimoto	1	2	$d_{ij} = \frac{a_1 + a_4}{a_1 + 2\left(a_2 + a_3\right) + a_4}$
Simple Matching (M)	1	1	$d_{ij} = \frac{a_1 + a_4}{p}$
Russel and Rao (RR)	-	-	$d_{ij} = \frac{a_1}{p}$
Dice	0	$\frac{1}{2}$	$d_{ij} = \frac{2a_1}{2a_1 + \left(a_2 + a_3\right)}$
Kulczynski	-	-	$d_{ij} = \frac{a_1}{a_2 + a_3}$

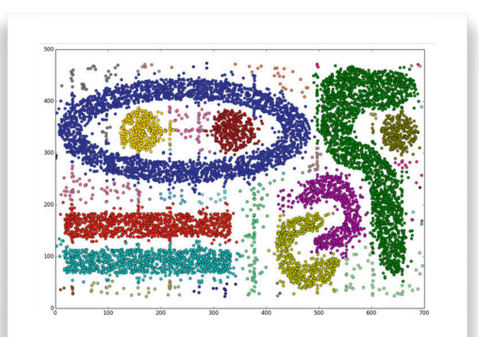
二元结构数据的相似性

$$\begin{cases} x_{ik} = x_{jk} = 1 \\ x_{ik} = 0\ ,\ x_{jk} = 1 \\ x_{ik} = 1\ ,\ x_{jk} = 0 \\ x_{ik} = x_{jk} = 0 \end{cases}$$

$$\Longrightarrow \begin{cases} a_1 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = x_{jk} = 1\right) \\ a_2 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = 0\ ,\ x_{jk} = 1\right) \\ a_3 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = 1\ ,\ x_{jk} = 0\right) \\ a_4 \triangleq \sum_{k=1}^{p} \mathrm{I}\left(x_{ik} = x_{jk} = 0\right) \end{cases}$$

Spectral Clustering 谱聚类

- 谱聚类是基于图论的方法将数据点划分为具有相同结构的组 (类).
  - 谱聚类方法的介绍，我们采用 Luxburg 2007 年给出的结果.
  - 从图论的观点来看 $\mathscr{D}$，是基于无向图 (undirected graph) $\mathscr{G}=\left(\mathscr{V},\mathscr{E}\right)$ 而言，其中的顶点 $\boldsymbol{v}_i\in\mathscr{V}$ 对应于数据点 $\boldsymbol{x}_i$.
  - 如果 $d_i>0$，则连通相应的两个 $\boldsymbol{v}$，并且给 $\boldsymbol{v}_i$ 和 $\boldsymbol{v}_j$ 之间的边赋值权重 d_i.
  - 聚类则归结为寻找 $\mathscr{G}$ 的一种划分，就矩阵 $\mathscr{D}$ 的值而言，使得不同类之间的边 e_{ij} 具有较低的权重.
  - 相似矩阵也可以称为 $\mathscr{G}$ 的邻 (连) 接矩阵 (adjacency matrix) $\mathscr{W}$，因为对于 $\mathscr{W}$ 的元素 w_{ij} 而言，$w_{ij}=0$ 对应于节点 $\boldsymbol{v}_i$ 和 $\boldsymbol{v}_j$ 未连通 (或不相似).



Spectral Clustering 谱聚类

- 由于聚类的结果随着数据点的局部接近程度而不同，因此通过 $d_i = \sum_{j=1}^{n} w_{ij}$ 来研究其连通性是有意义的.
  - 注意到 $d_{ii} = w_{ii}$ 使得 $\boldsymbol{x}_i$ 成为单独类，因为它没有连通到任何其他向量.
  - 定义

$$\mathscr{D} = \operatorname{diag}\left(d_i\right) = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}$$

指示向量: $\mathbf{1}_A = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$ → $f_i = \mathrm{I}\left(\boldsymbol{x}_i \in A\right)$

$A \subset \mathscr{V}$

Spectral Clustering 谱聚类

- 由于聚类的结果随着数据点的局部接近程度而不同，因此通过 $d_i = \sum_{j=1}^{n} w_{ij}$ 来研究其连通性是有意义的.
  - 子集 $A \subset \mathcal{V}$ 的大小可以通过 $|A| \overset{\text{def}}{=} A$ 的顶点数 和 $\text{vol}(A) \overset{\text{def}}{=} \sum_{i \in A} d_i$ 度量.
  - 记 $\overline{A} = \mathcal{V} \backslash A$，如果不存在点 w_{ij} 使得 $i \in A$，$j \in \overline{A}$，则我们定义 A 是连通的.

Spectral Clustering 谱聚类

- 谱聚类的基本思想是对所谓的拉普拉斯矩阵 $\mathscr{L} = \mathscr{D} - -\mathscr{W}$ 进行特征分析.
  - Luxburg (2007) 给出了拉普拉斯矩阵 (Laplacian matrix) 的如下性质：
    1. $\forall \boldsymbol{f} \in \mathbb{R}^n, \boldsymbol{f}^{\mathrm{T}} \mathscr{L} = \frac{1}{2} \sum_{i,\, j=1}^{n} w_{ij} \left(f_i - f_j \right)^2$
    2. $\mathscr{L}$ 的最小特征值为 0，对应的特征向量是 $\mathbf{1}_n$
    3. $\mathscr{L}$ 是对称、半正定矩阵
    4. $\mathscr{L}$ 有 n 个特征值 $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$
  - 未标准化的拉普拉斯矩阵为我们提供了一种检测连通性的工具.

Spectral Clustering 谱聚类

- 命题 2 (Luxburg 2007) (连通点的数量与 $\mathscr{L}$ 的谱). 设 $\mathscr{G}$ 是一个具有非负权重的无向图. 则 $\mathscr{L}$ 的 0 特征值的重数 k 等于图的连通组件 $A_1, A_2, \ldots, A_k$ 的个数. 特征值 0 的特征空间可以由这些组件的指示向量 $\mathbf{1}_{A_1}, \mathbf{1}_{A_2}, \ldots, \mathbf{1}_{A_k}$ 拓展生成. 事实上，我们可以将 $\mathscr{G}$ 含有 k 个组件的结构描述为 $\mathscr{L}$ 的块对角结构.

谱聚类算法

1. 从相似矩阵构造邻接矩阵 $\mathscr{W}$
2. 计算非标准化的拉普拉斯矩阵 $\mathscr{L}$
3. 计算 $\mathscr{L}$ 的前 k 个特征向量 $\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_k$
4. 定义 $n \times k$ 矩阵 $\mathscr{V} = \left(\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_k\right)$
5. 取 $\mathscr{V}$ 的第 i 行作为 $\boldsymbol{y}_i \in \mathbb{R}^k$，的数据矩阵 $\mathscr{Y}_{n\times k}$
6. 利用 ***k-means*** 聚类方法将 $\mathscr{Y}$ 聚类为 $C_1, C_2, \ldots, C_k$
7. 输出聚类结果：$A_1, A_2, \ldots, A_k$，其中 $A_i = \left\{ j \ : \ \boldsymbol{y}_j \in C_j \right\}$

Spectral Clustering 谱聚类

```
library(kernlab)
?specc
```

聚类的数目，或初始类的中心

用于计算相似度矩阵的核函数：rbfdot, polydot, vanilladot, tanhdot, laplacedot, besseldot, anovadot, splinedot, stringdot

```
specc(x, centers, kernel = "stringdo",  iterations = 200,
      na.action = na.omit, ...)
```

数据矩阵

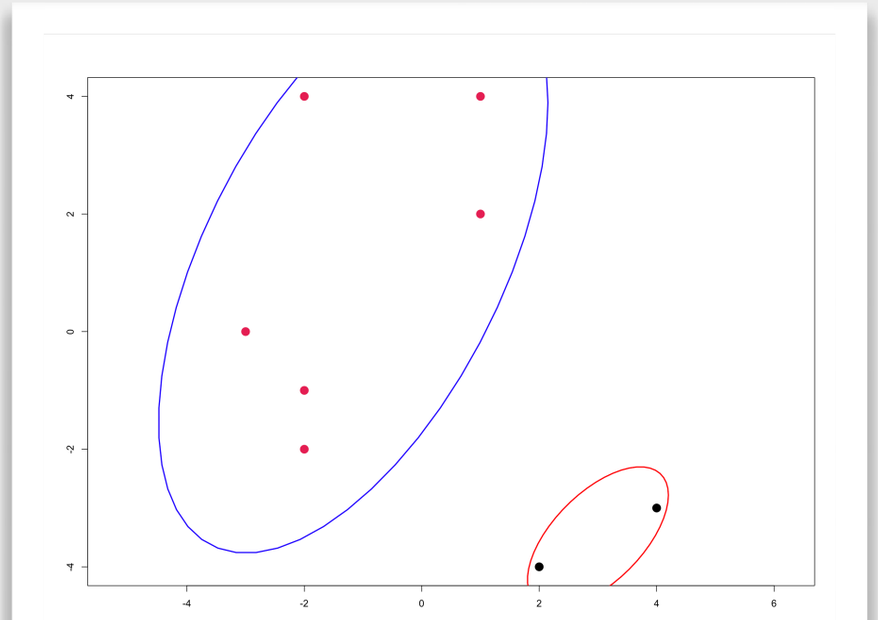
对 NA 值如何操作

最大迭代次数

Spectral Clustering 谱聚类

- Example: 利用谱聚类将 8 个点聚成两类.

```
eight = cbind(c(-3, -2, -2, -2, 1, 1, 2, 4), c(0, 4, -1, -2, 4, 2, -4, -3))
eight
eight = eight[c(8, 7, 3, 1, 4, 2, 6, 5), ] # 数据集重新排序
eight

sc.eight = specc(eight, centers = 2) # 谱聚类
sc.eight
centers(sc.eight) # 类中心
withinss(sc.eight) # 类的组内平方和

merg1 = eight[1:2, 1:2]
merg2 = eight[3:8, 1:2]
plot(eight, col = sc.eight, pch = 16, cex = 2, xlab = '', ylab = '', asp = 1)
ellipse(center = c(3, -3.5), shape = matrix(c(1, 0.6, 0.6, 1), nrow = 2), center.pch = 0, col = "red", radius = 1.2)
dataEllipse(x = merg2[, 1], y = merg2[, 2], center.pch = 0, col = "blue", plot.points = F, add = T, levels = 0.75)
```



iris 数据集的聚类分析

```
?hclust
```

聚类方法：ward.D, ward.D2, single, complete, average, mcquitty, median, centroid

```
hclust(d, method = "complete")
```

用 dist 产生的距离矩阵

字符向量，聚类树的标签

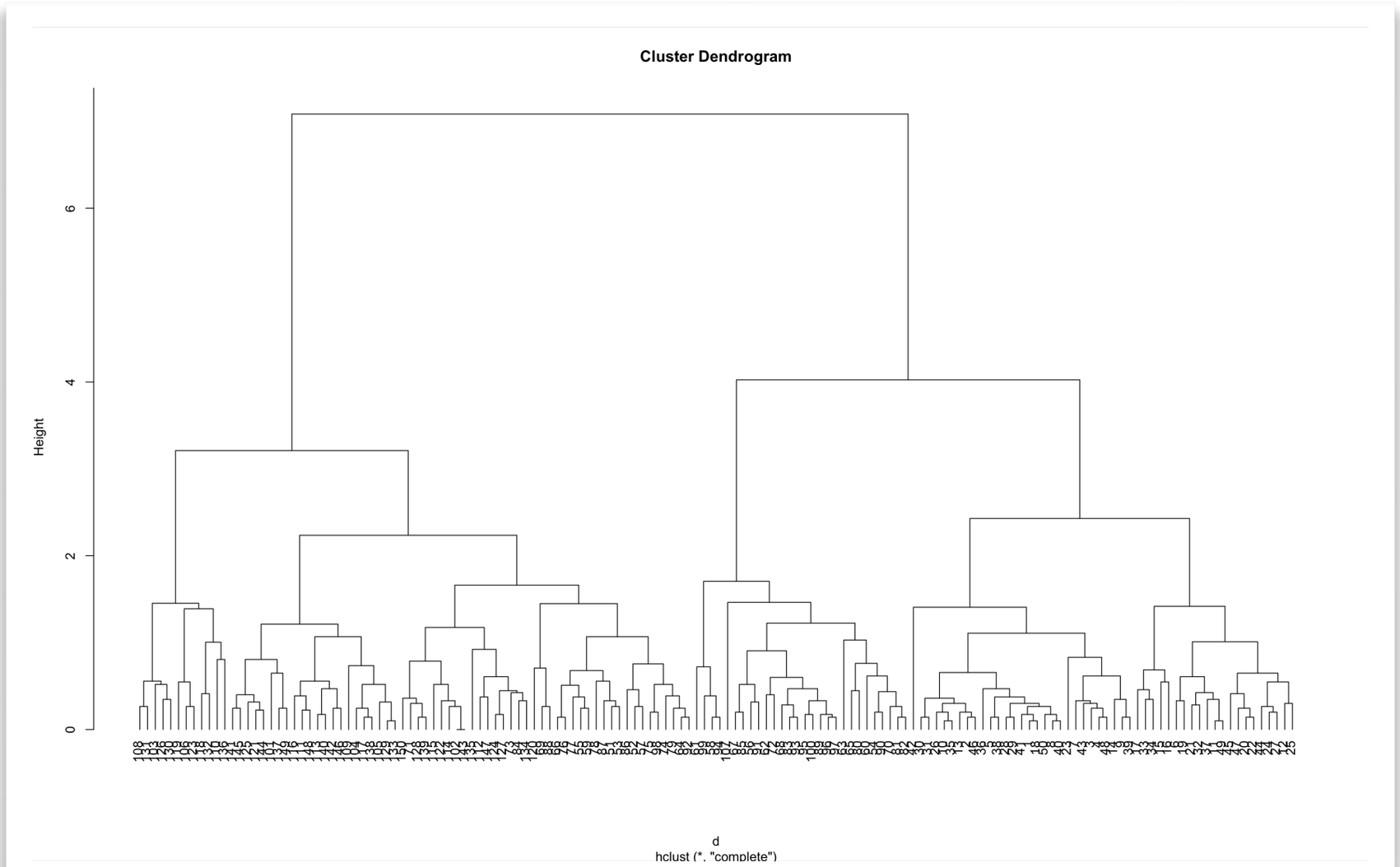
标签应位于绘图部分下方的比例，负值会使标签从 0 开始

```
plot(x, labels = NULL, hang = 0.1, axes = TRUE, frame,plot =
     FALSE, main = "Cluster Dendrogram", ...)
```

由 hclust 得到的对象

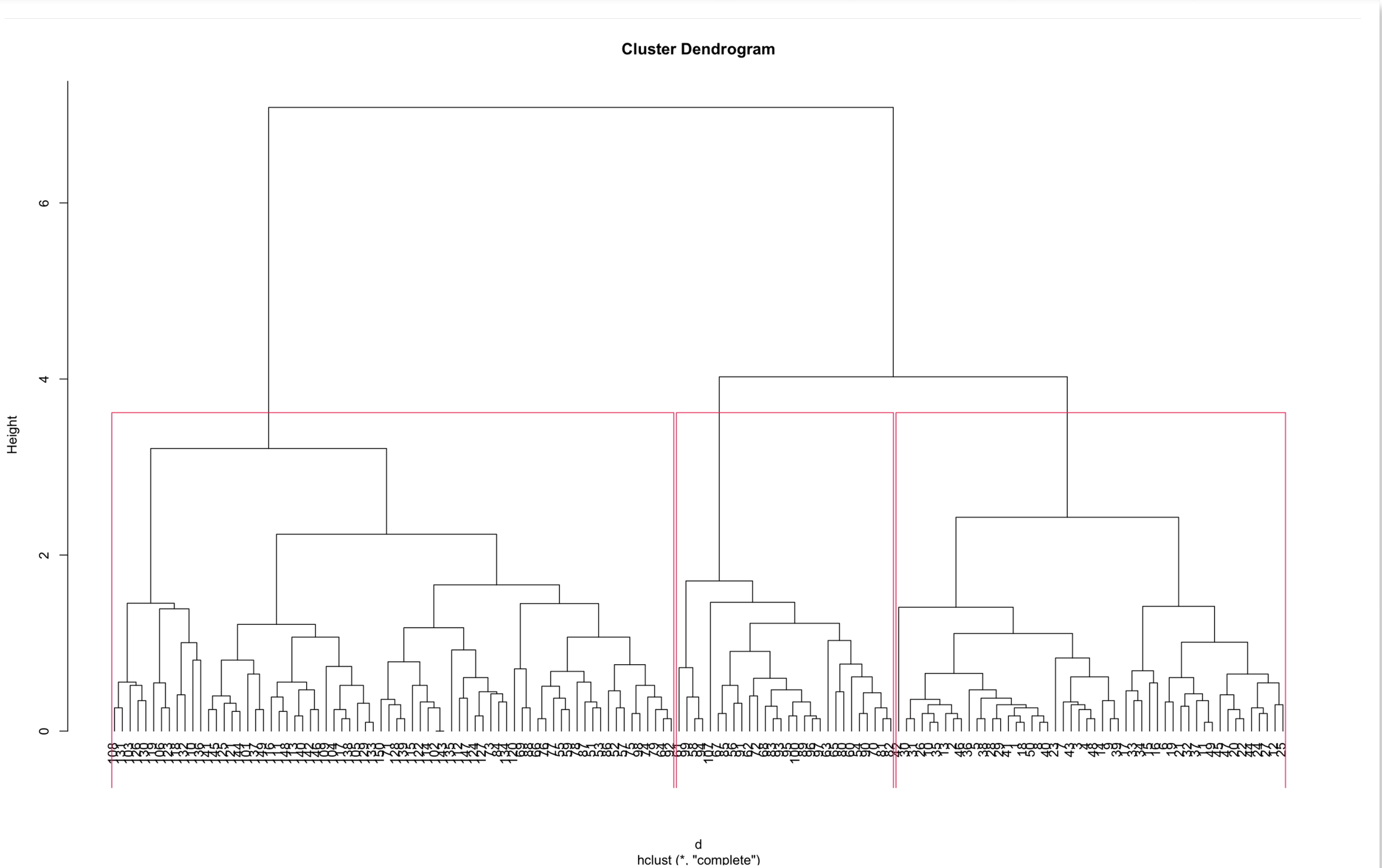
iris 数据集的聚类分析

```
rm(list = ls(all = TRUE))
x = iris
head(x)
d = dist(x[, 1:4]) # 计算距离矩阵
iris.hc1 = hclust(d, method = "complete") # 最长距离法
plot(iris.hc1, hang = -1) # 聚类树
```



iris 数据集的聚类分析

```
re1 = rect.hclust(iris.hc1, k = 3) # 聚类树分成三类
```



iris 数据集的聚类分析

```
Re1 # 分成三类的结果
```

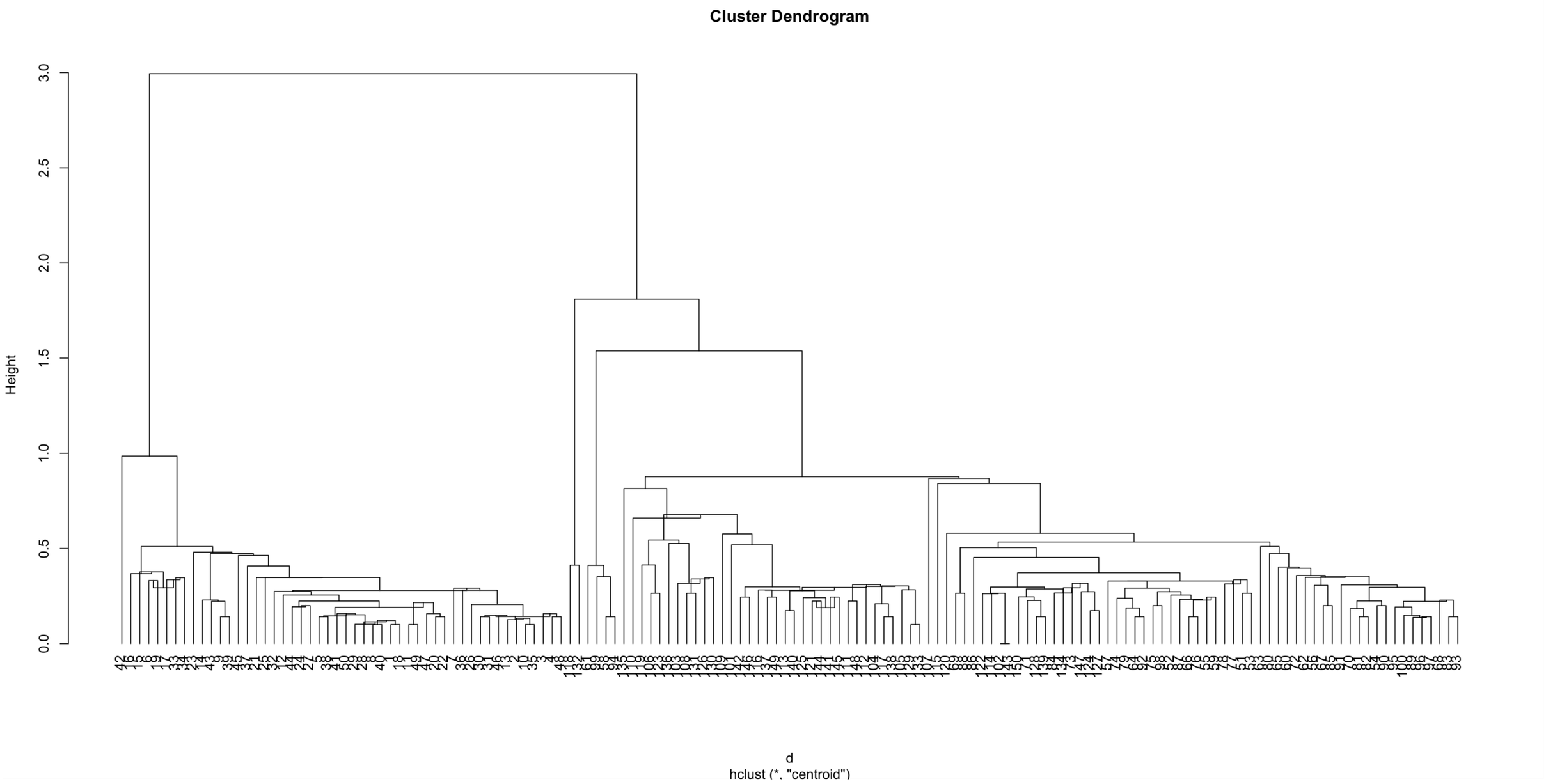
```
> re # 分成三类的结果
[[1]]
 [1]  51  52  53  55  57  59  64  66  69  71  73  74  75  76  77  78  79  84  86  87  88  92  98 101 102 103 104 105
[29] 106 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134
[57] 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150

[[2]]
 [1]  54  56  58  60  61  62  63  65  67  68  70  72  80  81  82  83  85  89  90  91  93  94  95  96  97  99 100 107

[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
[39] 39 40 41 42 43 44 45 46 47 48 49 50
```

```
iris.id1 = cutree(iris.hc1, 3) # 编成三组
table(iris.id1, x$Species) # 分类结果对比
```

```
> iris.id = cutree(iris.hc1, 3) # 编成三组
> table(iris.id, x$Species) # 分类结果对比

iris.id setosa versicolor virginica
      1     50          0         0
      2      0         23        49
      3      0         27         1
```

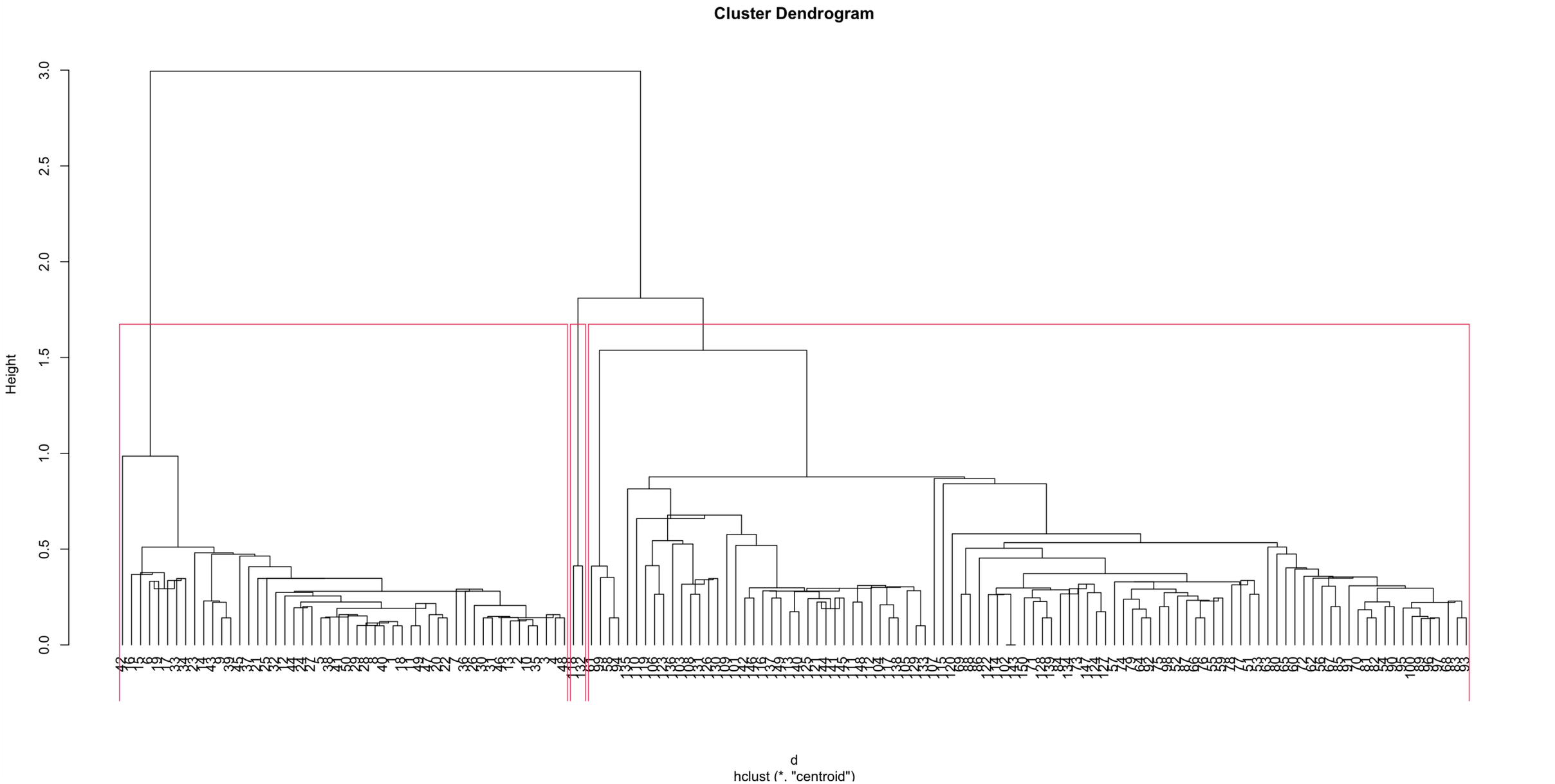
iris 数据集的聚类分析

```
iris.hc2 = hclust(d, method = "centroid") # 重心法
plot(iris.hc2, hang = -1) # 聚类树
```



iris 数据集的聚类分析

```
re2 = rect.hclust(iris.hc2, k = 3) # 聚类树分成三类
```



iris 数据集的聚类分析

```
re2 # 分成三类的结果
```

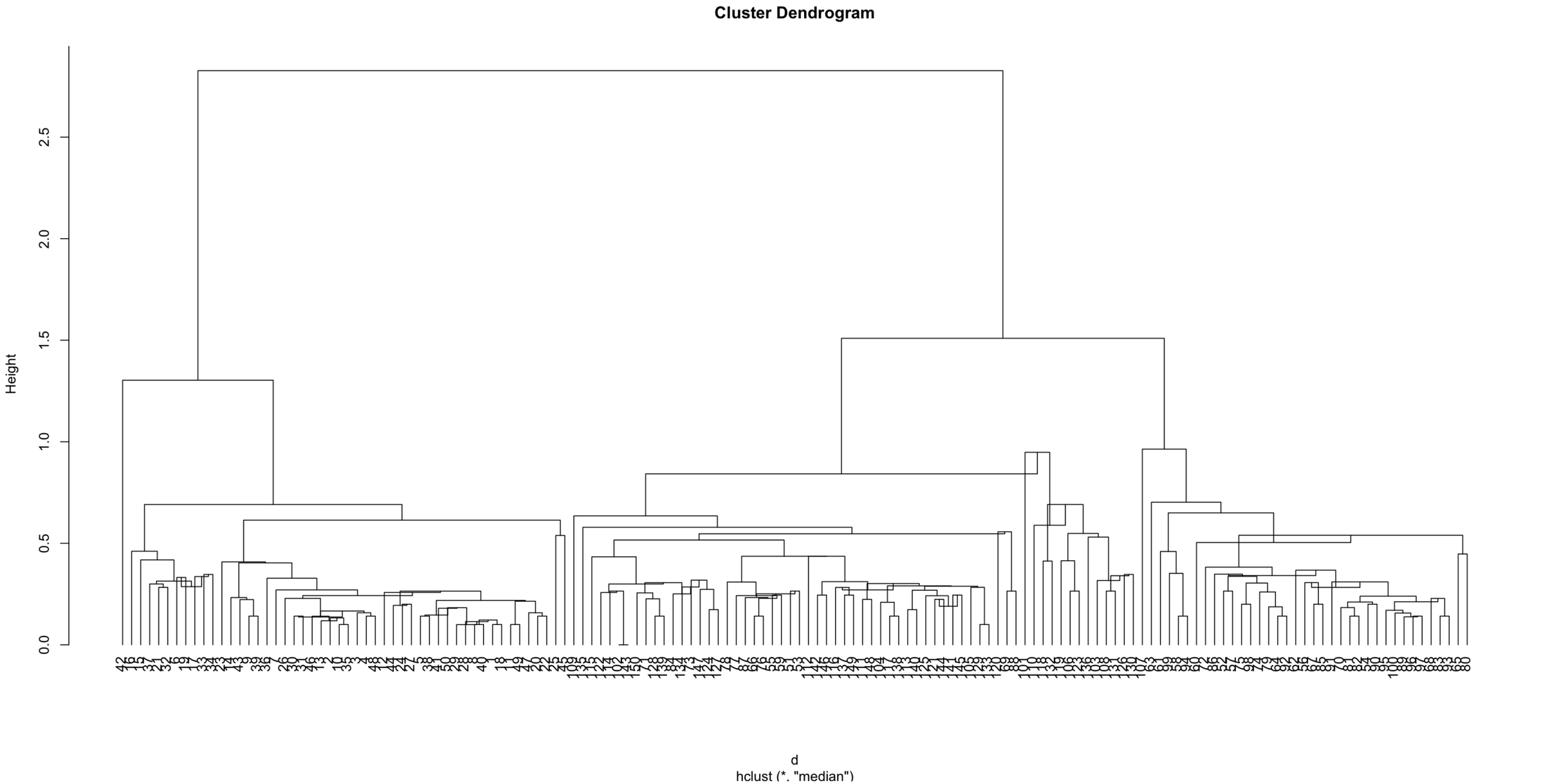
```
> re2 # 分成三类的结果
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
[39] 39 40 41 42 43 44 45 46 47 48 49 50

[[2]]
[1] 118 132

[[3]]
 [1]  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
[29]  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106
[57] 107 108 109 110 111 112 113 114 115 116 117 119 120 121 122 123 124 125 126 127 128 129 130 131 133 134 135 136
[85] 137 138 139 140 141 142 143 144 145 146 147 148 149 150
```

```
iris.id2 = cutree(iris.hc2, 3) # 编成三组
table(iris.id2, x$Species) # 分类结果对比
```

```
> table(iris.id2, x$Species) # 分类结果对比

iris.id2 setosa versicolor virginica
       1     50          0         0
       2      0         50        48
       3      0          0         2
```

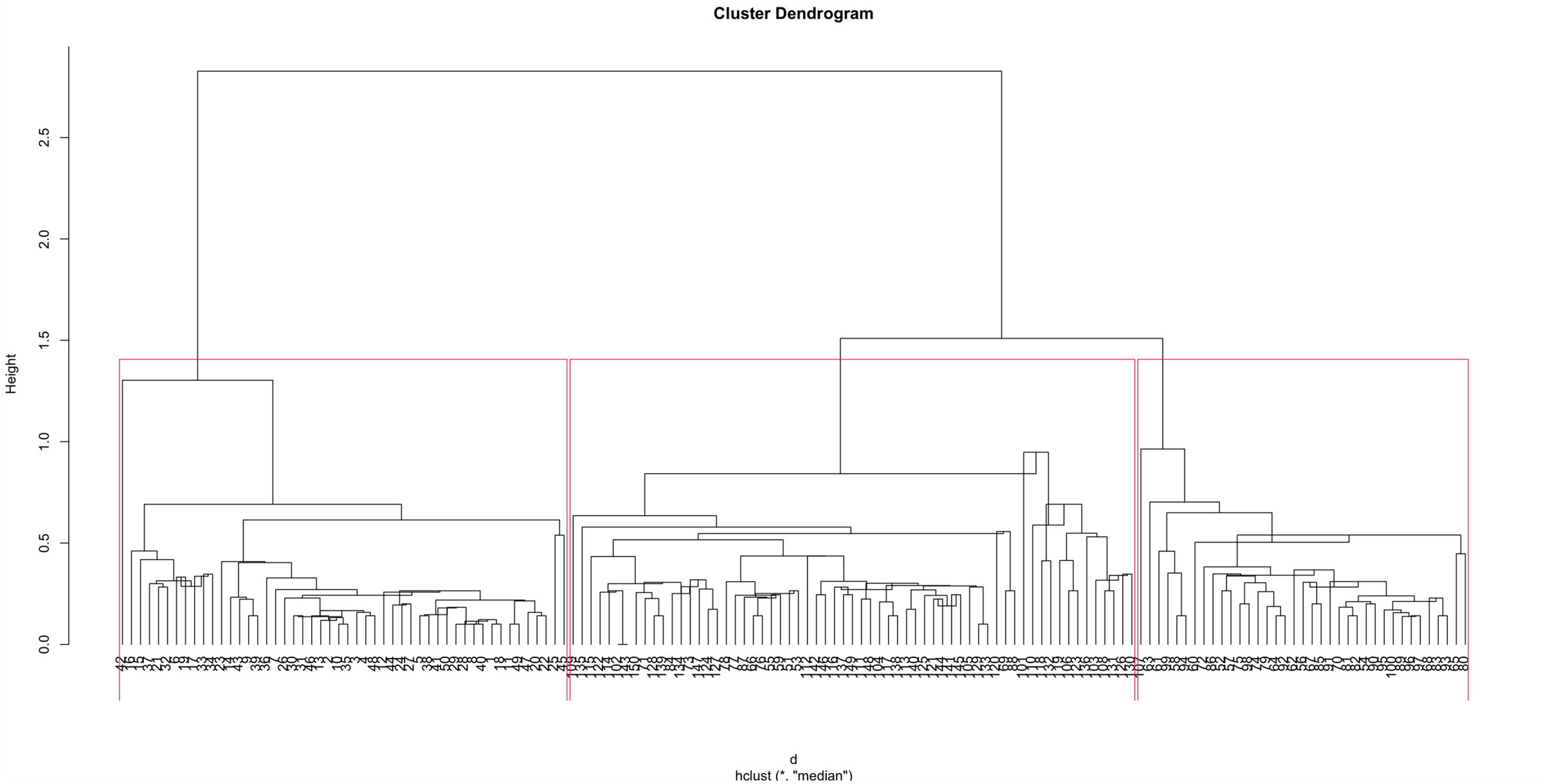
iris 数据集的聚类分析

```
iris.hc3 = hclust(d, method = "median") # 中位数法
```



iris 数据集的聚类分析

```
re3 = rect.hclust(iris.hc3, k = 3) # 聚类树分成三类
```



iris 数据集的聚类分析

```
re3 # 分成三类的结果
```

```
> re3 # 分成三类的结果
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
[39] 39 40 41 42 43 44 45 46 47 48 49 50

[[2]]
 [1]  51  53  55  59  66  69  71  73  76  77  78  84  87  88 101 102 103 104 105 106 108 109 110 111 112 113 114 115
[29] 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
[57] 144 145 146 147 148 149 150

[[3]]
 [1]  52  54  56  57  58  60  61  62  63  64  65  67  68  70  72  74  75  79  80  81  82  83  85  86  89  90  91  92
[29]  93  94  95  96  97  98  99 100 107
```

```
iris.id3 = cutree(iris.hc3, 3) # 编成三组
table(iris.id3, x$Species) # 分类结果对比
```

```
> table(iris.id3, x$Species) # 分类结果对比

iris.id3 setosa versicolor virginica
       1     50          0         0
       2      0         14        49
       3      0         36         1
```

USArrests 数据集的聚类分析

?kmeans

聚类数，或初始类的中心

```
kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong",
       "Lloyd", “Forgy", “MacQueen”), trace = FALSE)
```

数据矩阵

USArrests 数据集的聚类分析

```
## 动态聚类法 kmeans()
rm(list = ls(all = TRUE))
library(factoextra)
library(cluster)
df = USArrests # 载入数据
df = na.omit(df) # 移除有缺失值的数据，该数据集没有缺失数据
df = scale(df) # 标准化
head(df)
fviz_nbclust(df, kmeans, method = "wss") # 聚类数量与总体平方和的关系图，聚类数 k 应选择使总体平方和趋缓时的值 k = 4
gap_stat = clusGap(df, FUN = kmeans, nstart = 25, K.max = 10, B = 50) # 根据聚类数量计算差距统计量
fviz_gap_stat(gap_stat) # 聚类数量与差距统计量的关系图，聚类数 k 应选择使差距统计量最大时的值 k = 4
km_df = kmeans(df, centers = 4, nstart = 25) # 聚为 4 类的 kmeans 统计
km_df # 查看聚类结果
fviz_cluster(km_df, data = df) # 聚类图
aggregate(USArrests, by = list(cluster = km_df$cluster), mean) # 计算每一类当中各变量的均值
final_data = cbind(USArrests, Cluster = km_df$cluster) # 将聚类结果添加到原始数据当中
final_data
```

spirals 数据集的聚类分析

```
## 谱聚类法 specc()
rm(list = ls(all = TRUE))
library(kernlab)
data(spirals) # 载入数据
x = spirals
head(x)
plot(x, xlab = '', ylab = '', pch = 16, cex = 1.5) # 数据集的散点图
sc.x = specc(x, centers = 2) # 聚成 2 类的谱聚类
sc.x # 聚类结果
centers(sc.x) # 类的中心
size(sc.x) # 每一类中数据点的数量
withinss(sc.x) # 每一类的组内平方和
plot(x, xlab = '', ylab = '', pch = 16, cex = 1.5, col = sc.x) # 按照分类结果的散点图
```