R 机器学习

第 16 讲　聚类分析

张敬信

2022 年 10 月 27 日

哈尔滨商业大学

- 聚类分析是无监督学习，相当于”物以类聚”，事先并不知道类别的个数与结构，根据对象之间的相似性或相异性（距离度量）对研究对象进行分类。
- 通常所说的聚类分析是指样本聚类（Q 型）：将每个观测看作空间中的一个点，计算两两观测之间的距离或相似度（距离/相异度矩阵），将距离或相似度接近的点聚合成一类。
- 常用聚类分析的算法有：K-means、K-Medoids、DBSCAN（基于密度）、层次聚类、谱聚类、EM 聚类等。
- 聚类分析常用于挖掘客户价值、根据指标数据划分等级、识别异常等。

一. 距离/相异度度量

- 聚类分析中，距离/相异度度量的选择至关重要，因为它会很大程度地影响聚类结果。对两个观测 (x, y), 介绍几种常用的距离度量。
  (1) 欧氏距离
  (2) 曼哈顿距离
  (3) 马氏距离
  (4) Pearson 相关系数：度量两个观测的线性相关程度
  (5) Spearman 相关系数
  (6) Kendall 相关系数
  (7) 余弦相似度
  (8) Gower 距离

二. 数据标准化

- 数据不能有缺失值，当变量的量纲不同时，非常有必要对数据做标准化处理，目的是使其具有可比性：

$$\frac{x_i - center(x)}{scale(x)}$$

其中，$center(x)$ 可以是 x 的均值或中位数，$scale(x)$ 可以是标准差、四分位距、绝对中位差（Median absolute deviation）。

- 用 R 自带的 `scale()` 可以做数据标准化，只接受连续型数据。
- **注 1**：标准化让欧氏距离、曼哈顿距离、相关系数、余弦相似度更加相似。
- **注 2**：对于标准化数据，欧氏距离与相关系数具有如下函数关系：

$$d_2(x, y) = \sqrt{2m[1 - r(x, y)]}$$

其中，m 为向量的维数。

3. 计算样本间的距离矩阵

- `dist()`：只接受数值型输入，参数 `method` 可设置为"euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski" 以计算不同的（行间）距离。
- `factoextra::get_dist()`：只接受数值型输入，参数 `method` 除了支持上述距离外，还支持相关系数距离："pearson", "spearman", "kendall".
- `cluster::daisy()`：支持非数值类型，如名义型、有序型、(非) 对称二值型，适合混合类型数据。

计算欧氏距离

```
library(cluster)
library(factoextra)

df = mtcars[, 3:7]    # 数值型列
df = scale(df)        # 数据标准化

dist2 = dist(df, method = "euclidean")
as.matrix(dist2)[1:3, 1:3]
#>                Mazda RX4 Mazda RX4 Wag Datsun 710
#> Mazda RX4          0.000         0.408        1.34
#> Mazda RX4 Wag      0.408         0.000        1.17
#> Datsun 710         1.337         1.166        0.00
```

计算马氏距离

```r
dist.maha = function(dat) {
  X = as.matrix(na.omit(dat))
  V = cov(X)                                  # 协方差矩阵，正定
  L = t(chol(V))                              # 下三角
  stdX = t(forwardsolve(L, t(X)))             # 标准化
  dist(stdX)
}

dist_m = dist.maha(mtcars[, 3:7])
as.matrix(dist_m)[1:3, 1:3]
#>       1     2    3
#> 1 0.000 0.621 1.93
#> 2 0.621 0.000 1.84
#> 3 1.934 1.838 0.00
```

注： R 自带的 `mahalanobis()` 函数，计算 `x` 各行（样本）到样本中心 `center` 的马氏距离：

```
mahalanobis(datas, colMeans(datas), cov(datas))
```

计算相关系数距离

```
dist_cor = get_dist(df, method = "pearson")
as.matrix(dist_cor)[1:3, 1:3]
#>                Mazda RX4 Mazda RX4 Wag Datsun 710
#> Mazda RX4         0.0000         0.0406      0.506
#> Mazda RX4 Wag     0.0406         0.0000      0.368
#> Datsun 710        0.5060         0.3680      0.000
```

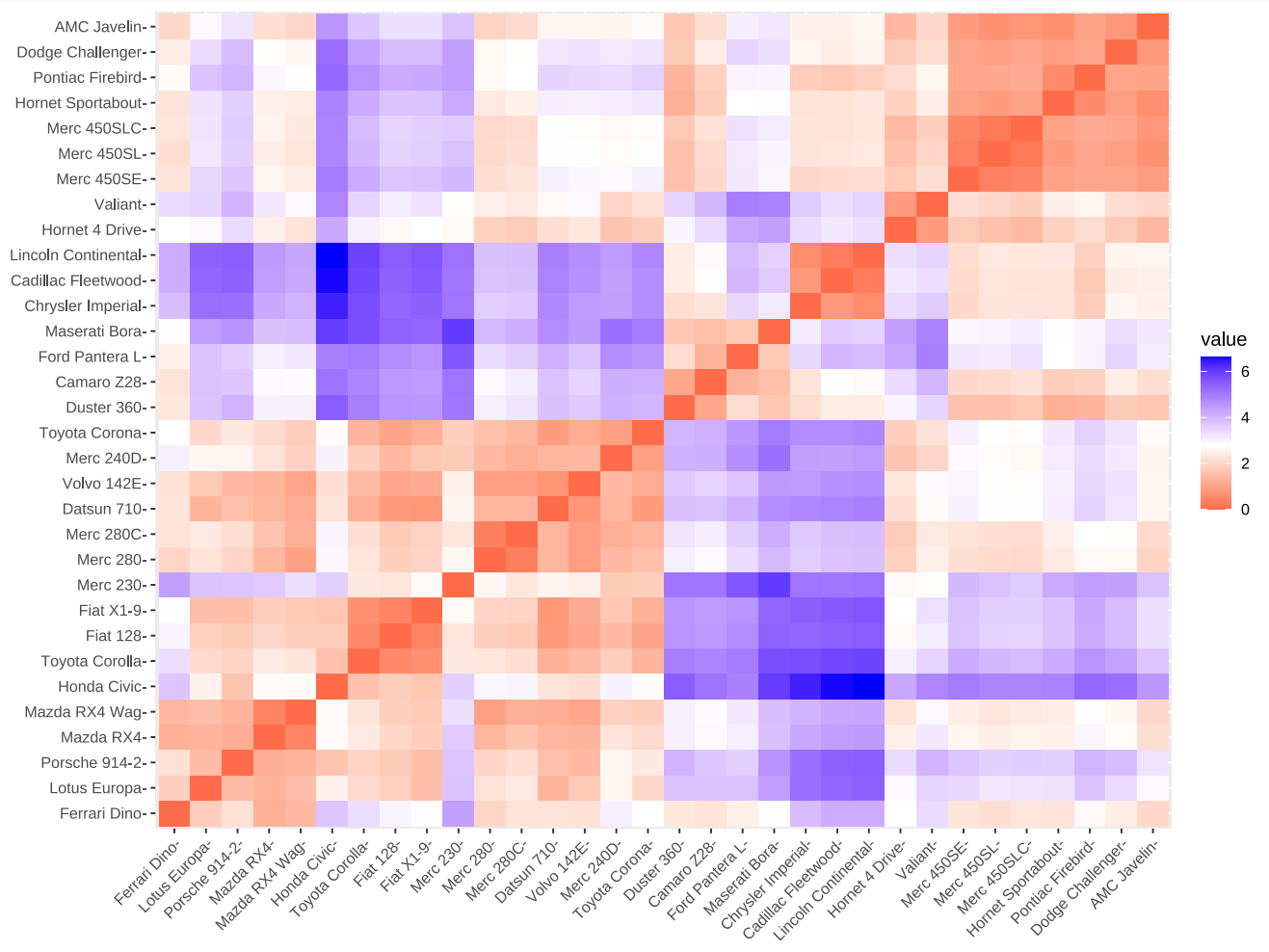
计算混合数据 Gower 距离

```
data(flower)
str(flower)
#> 'data.frame':    18 obs. of  8 variables:
#>  $ V1: Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 1 2 2 ...
#>  $ V2: Factor w/ 2 levels "0","1": 2 1 2 1 2 2 1 1 2 2 ...
#>  $ V3: Factor w/ 2 levels "0","1": 2 1 1 2 1 1 1 2 1 1 ...
#>  $ V4: Factor w/ 5 levels "1","2","3","4",..: 4 2 3 4 5 4
#>  $ V5: Ord.factor w/ 3 levels "1"<"2"<"3": 3 1 3 2 2 3 3 2
#>  $ V6: Ord.factor w/ 18 levels "1"<"2"<"3"<"4"<..: 15 3 1
#>  $ V7: num  25 150 150 125 20 50 40 100 25 100 ...
#>  $ V8: num  15 50 50 50 15 40 20 15 15 60 ...
```

```r
dist_gower = daisy(flower)
as.matrix(dist_gower)[1:3, 1:3]
#>       1     2     3
#> 1 0.000 0.888 0.527
#> 2 0.888 0.000 0.515
#> 3 0.527 0.515 0.000
```

距离矩阵可视化

```
fviz_dist(dist2)
```



注：红色表示高相似度（低相异度），蓝色表示低相似度。

二. K-means

- 将给定数据集划分为 K 个类（群集），其中 K 为预先指定的类数。
- 分类依据是使同一类中的样本尽可能相似（类内高相似度），而来自不同类的样本尽可能不相似（类间高相异度）。
- 每个类用其**中心**表示，即分配给该类的所有样本的平均值样本。

1. K-means **基本思想**

- K-means 聚类的基本思想，就是让**总类内离差平方和**达到最小，标准算法**类内离差平方和**定义为类内各样本到该类中心的欧氏距离的平方和：

$$W(C_k) = \sum_{x \in C_k} (x_i - \mu_k)^2$$

- 每个样本 x_i 被归入某个类，使得如下的**总类内离差平方和**最小：

$$tot.withinss = \sum_{k=1}^{K} W(C_k) = \sum_{k=1}^{K} \sum_{x \in C_k} (x_i - \mu_k)^2$$

2. K-means 算法步骤

(1) 指定要划分的类数K

(2) 随机选择K个样本，作为初始的类中心

(3) 计算每个样本到各个类中心的欧氏距离，将样本归入距离最近的类

(4) 对每个类，计算其新的"中心"（各变量取平均值得到）

(5) 重复(3)(4)步以最小化总类内离差平方和，直到类的划分不再变化或达到最大迭代次数

3. 最佳 K 值的确定

- K 值的选取直接影响到聚类效果，常用的方法有：
  - Elbow 法：可视化**总离差平方和**，找到最佳”拐点”
  - Silhouette 法：计算每个观测的**平均 Silhouette 值**，找到最大值点
  - Gap 统计量法（慢）：计算 Gap 统计量，找到最大值点
- 可用 `factoextra::fviz_nbclust()` 可视化实现，另外 NbClust 包提供了更多种度量指标。

4. K-means 算法的优缺点

- K-means 算法的优点是简单、速度快，可以处理大数据量；
- K-means 算法的缺点：
  - 需要预先设定聚类数 K
  - 聚类结果对初始聚类中心的选取敏感（可多次运行选**总类内平方和**小的）
  - 聚类结果对异常值敏感（可改用更稳健的 K-medoids 算法）
  - 若对数据重排顺序，聚类结果可能会变

5. 其它类似算法

- K-medoids 也称为 PAM 算法，与 K-means 算法类似，只是选择聚类中心时不是用平均值（虚拟样本点），而是用真正的样本点。
- 模糊 C-均值聚类（FCM），每个类的中心是用所有样本点的加权平均来计算，权重为样本点属于该类的隶属度。

6. K-means 案例

- 现有餐饮用户的消费行为特征数据，共 940 个观测，4 个变量
- 其中 R 表示最近一次消费，F 表示消费频率，M 为消费金额。根据这些数据将客户分类成不同客户群，并评价其价值。

clust.kmeans 学习器，调用自带的 kmeans() 函数，基本格式为：

```
kmeans(x, centers, iter.max = 10, nstart = 1)
```

其中，

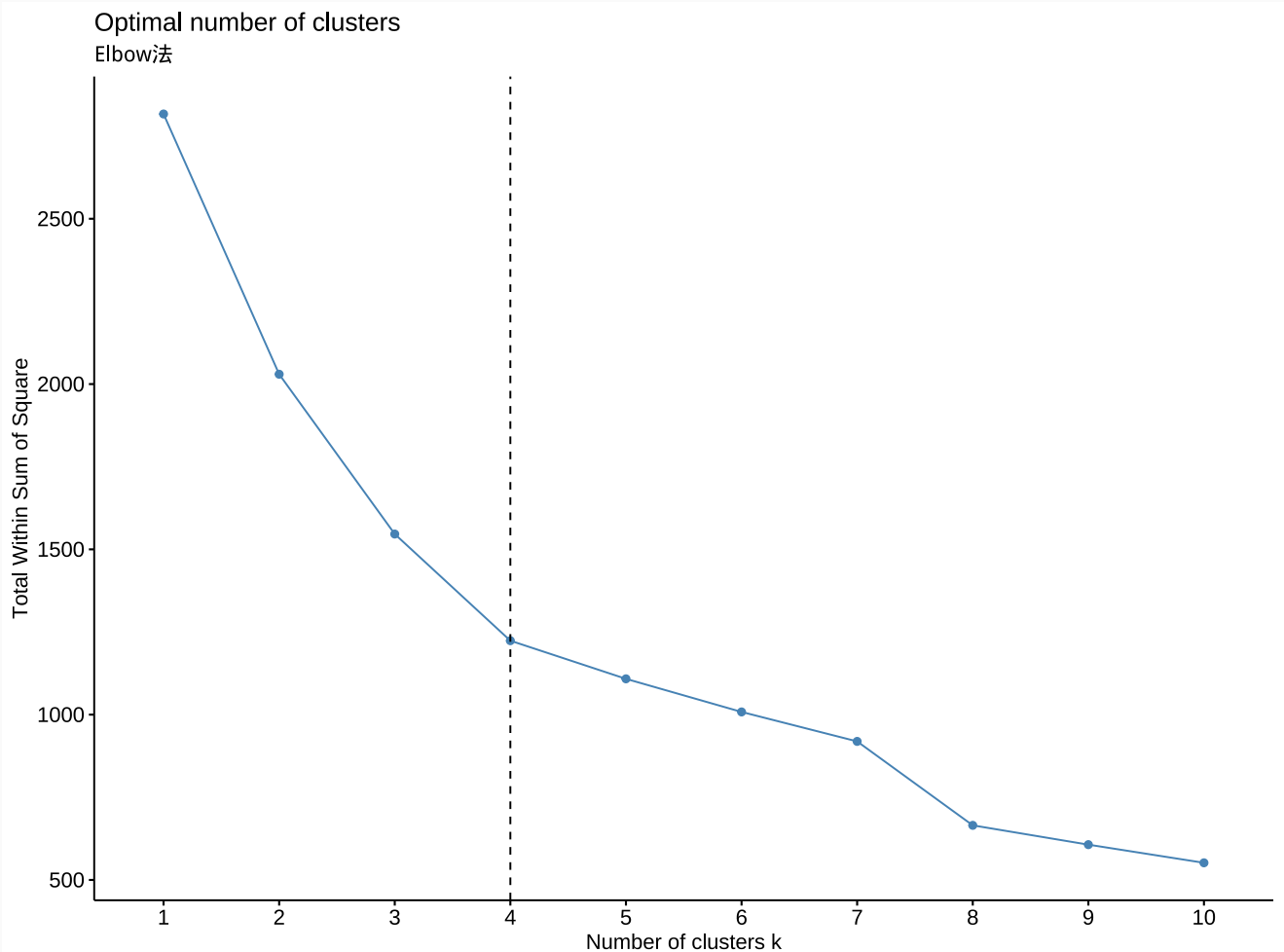
- x 为要聚类的数据，可以是数值矩阵、数据框；
- centers 为要划分的类数 K；
- iter.max 为最大迭代次数；
- nstart 为初始划分时的随机分组数，若 centers 为数值，建议设置为 > 1.

(1) 准备数据

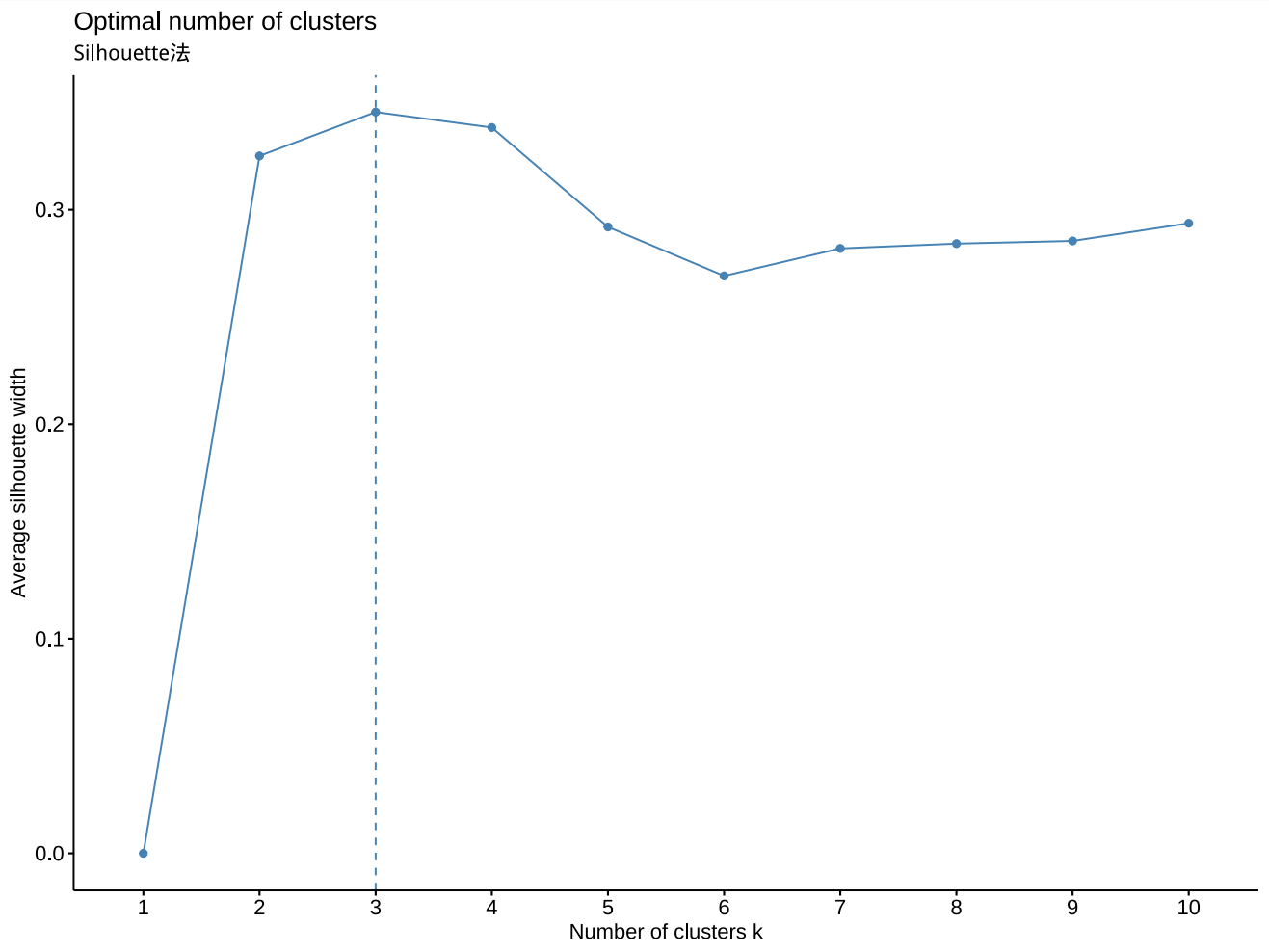
```
library(tidyverse)
library(factoextra)    # 用于聚类绘图和确定最佳 K 值

df = readr::read_csv("datas/consumption.csv") %>%
  mutate(across(-1, ~ (.x - mean(.x)) / sd(.x)))   # 数据标准
head(df, 3)
#> # A tibble: 3 x 4
#>      Id      R      F      M
#>   <dbl>  <dbl>  <dbl>  <dbl>
#> 1     1  0.764 -0.494 -1.16
#> 2     2 -1.02  -0.630  0.623
#> 3     3 -0.950  0.871 -0.341
```

(2) 选择最佳 K 值

```
fviz_nbclust(df[,-1], kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2) +
  labs(subtitle = "Elbow 法")
```



```
fviz_nbclust(df[,-1], kmeans, method = "silhouette") +
  labs(subtitle = "Silhouette 法")
```



- 创建聚类任务：

```
library(mlr3verse)
task = TaskClust$new("RFM", df)
task$set_col_roles("Id", roles = "name")
task
#> <TaskClust:RFM> (940 x 3)
#> * Target: -
#> * Properties: -
#> * Features (3):
#>   - dbl (3): F, M, R
```

- 选择 clust.kmeans 学习器，设置聚类数为 3, 训练模型：

```
learner = lrn("clust.kmeans", centers = 3, nstart = 25)
learner$train(task)
km = learner$model
broom::glance(km)
#> # A tibble: 1 x 4
#>   totss tot.withinss betweenss  iter
#>   <dbl>        <dbl>     <dbl> <int>
#> 1  2817        1546.     1271.     3
```

注： totss 表示总平方和，即 $\sum(x_i - \bar{x})^2$; tot.withinss 表示总类内平方和，即 sum(withinss); betweenss 表示类间平方和，即总平方和减去类内平方和。

- 查看各个样本归入的类别：

```
km_res = bind_cols(df, Class = km$cluster)
head(km_res)
#> # A tibble: 6 x 5
#>      Id      R      F      M Class
#>   <dbl>  <dbl>  <dbl>  <dbl> <int>
#> 1     1  0.764 -0.494 -1.16      3
#> 2     2 -1.02  -0.630  0.623     3
#> 3     3 -0.950  0.871 -0.341     1
#> 4     4 -1.02   0.189 -1.16      3
#> 5     5 -0.205 -0.357  1.19      3
#> 6     6  0.168 -0.494 -1.18      3
```

- 查看聚类中心及相关信息

```
km_res[,-1] %>%
  group_by(Class) %>%
  summarise(n = n(), across(.fns = mean)) %>%
  mutate(withins = km$withinss)
#> # A tibble: 3 x 6
#>   Class     n      R      F      M withins
#>   <int> <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
#> 1     1   341 -0.160  1.11   0.393    668.
#> 2     2    40  3.46  -0.296  0.449    188.
#> 3     3   559 -0.149 -0.659 -0.272    690.
```

- 可视化聚类结果（用到 PCA 技术）

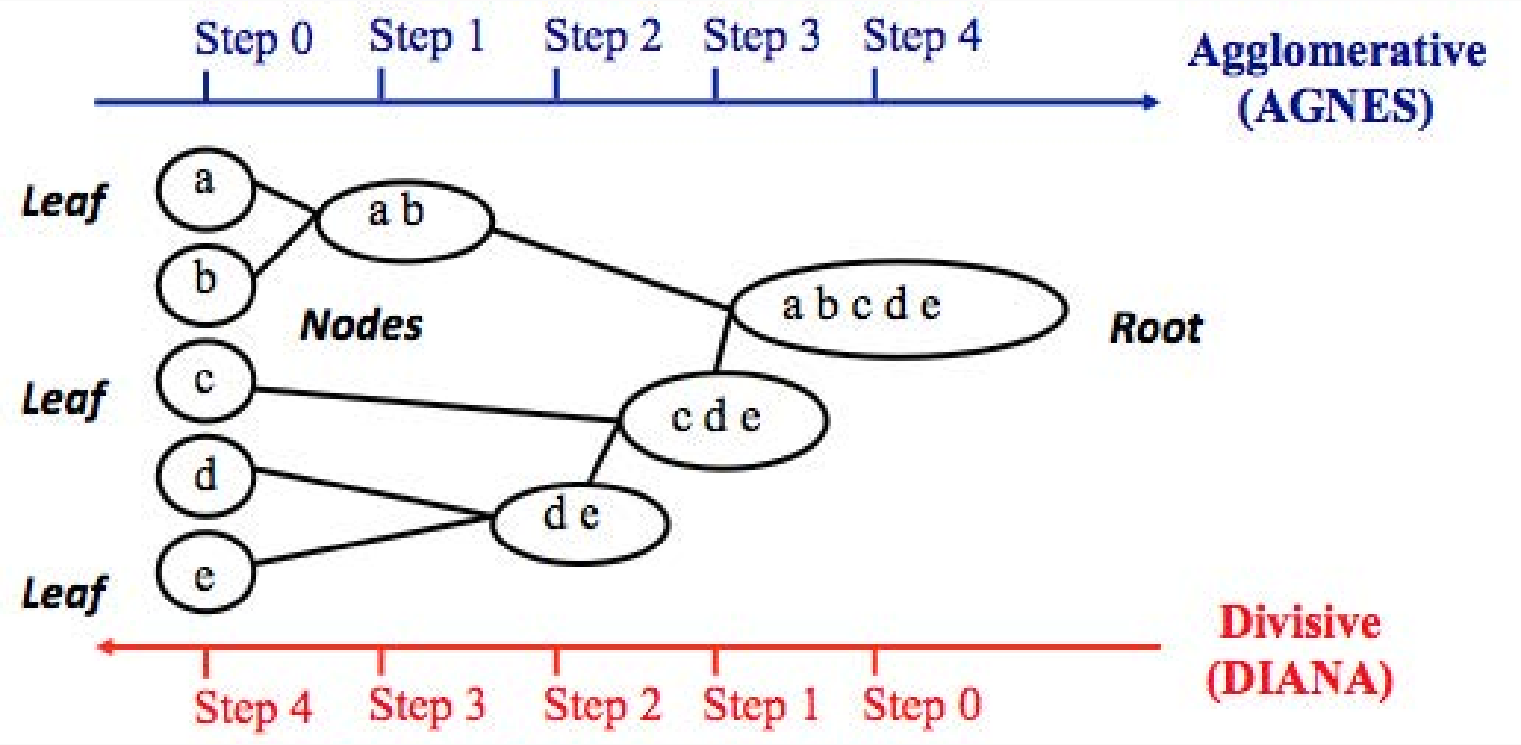
```
# 可视化聚类结果
prediction = learner$predict(task)
# 在任务上执行 PCA，展示聚类效果，添加概率椭圆（假定正态分布
autoplot(prediction, task, type = "pca",
         frame = TRUE, frame.type = "norm")
```


cluster 1 2 3
Cluster 1 2 3
0.2
0.1
0.0
PC2 (33.85%)
-0.1
0.0
0.1
0.2
PC1 (41.5%)

三. 层次聚类

- **层次聚类**，包括聚合聚类和分裂聚类。
- 聚合聚类，是以一种”自底而上”的方式聚类，最开始将每个样本作为单独一类（叶），每步迭代都将最相似的两个类合并为一个新类（节点），直到最终所有样本都合并为一个类（根），从而形成一个**树状图**。
- 分裂聚类，与聚合聚类相反，是以一种”自顶而下”的方式聚类，最开始所有样本作为整体的一个类，每步迭代都将最异质的一类分割为两个类，直到所有样本都单独作为一个类。
- 聚合聚类，擅长识别小的类；分裂聚类擅长识别大的类。通常用的是聚合聚类。

```
knitr::include_graphics(path = "figs/hclust.jpg")
```



1. 层次聚类的步骤

（1）准备数据（处理缺失值，标准化处理）

（2）计算两两样本之间的相似度或相异度

（3）根据前面计算的度量信息，用聚合函数将距离近的样本/类别聚合成层次聚类树

（4）确定从何处将层次聚类树剪切为最终聚类结果

2. 聚合算法

- 对标准化后的数据，用 `dist()` 函数计算距离矩阵
- 再用聚合算法计算两个样本/小类之间的距离，距离最小的样本/小类就合并为一个大类，重复该过程继续创建更大的类，直到所有样本都连接到一起构成一棵层次树。
- 常用的聚合算法包括：
    - 最大距离法（complete）：倾向于生成更紧凑的聚类
    - 最小距离法（single）：倾向于生成松散的聚类
    - 平均距离法（average）
    - 中心距离法（centroid）
    - Ward 最小离差法（ward.D, ward.D2）

注：更建议采用 Ward 最小离差法。

3. **树状图**

- 层次聚类会生成一个**树状图**，每个”叶节点”对应一个样本，从下往上看相似的样本在更高的层次聚合为”枝节点”。
- 从竖直方向看，“枝节点”的高度（称为共表距离）表明了两个样本/小类的相异度或距离，“枝节点”越高，其分支的相似性越少。
- 注意，只能用竖直高度（而不能用水平方向）作为度量两个样本/小类的相似性的依据。

注：用 `cophenetic()` 函数可以计算共表距离；共表距离与原始距离的线性相关性能够反映聚类效果的好坏，由此可以测试并选择最好的聚合算法。

4. 剪切树状图得到聚类结果

- 要得到最终的聚类结果，只需要在一定高度对树状图做剪切即可。问题是，层次聚类并不能告诉我们应该聚为几类或者从何处剪切树状图。
- 与 k-Means 类似，也可以 `factoextra::fviz_nbclust()` 可视化寻找最佳 K 值：
  - Elbow 法：可视化**总离差平方和**，找到最佳”拐点”
  - Silhouette 法：计算每个观测的**平均 Silhouette 值**，找到最大值点
  - Gap 统计量法（慢）：计算 Gap 统计量，找到最大值点

5. 层次聚类案例

- 现有 2014 年我国 30 个省份（除西藏）就业质量方面的数据，包含指标：
  - X1：人均教育经费；
  - X2：平均工资；
  - X3：失业率；
  - X4：医保参保率；
  - X5：人均就业培训投入
- 用 R 自带的 `hclust()` 实现，需要先标准化处理；或者用 `cluster` 包，提供了一步到位实现：聚合聚类 `agnes()` 和分裂聚类 `diana()`

(1) 准备数据

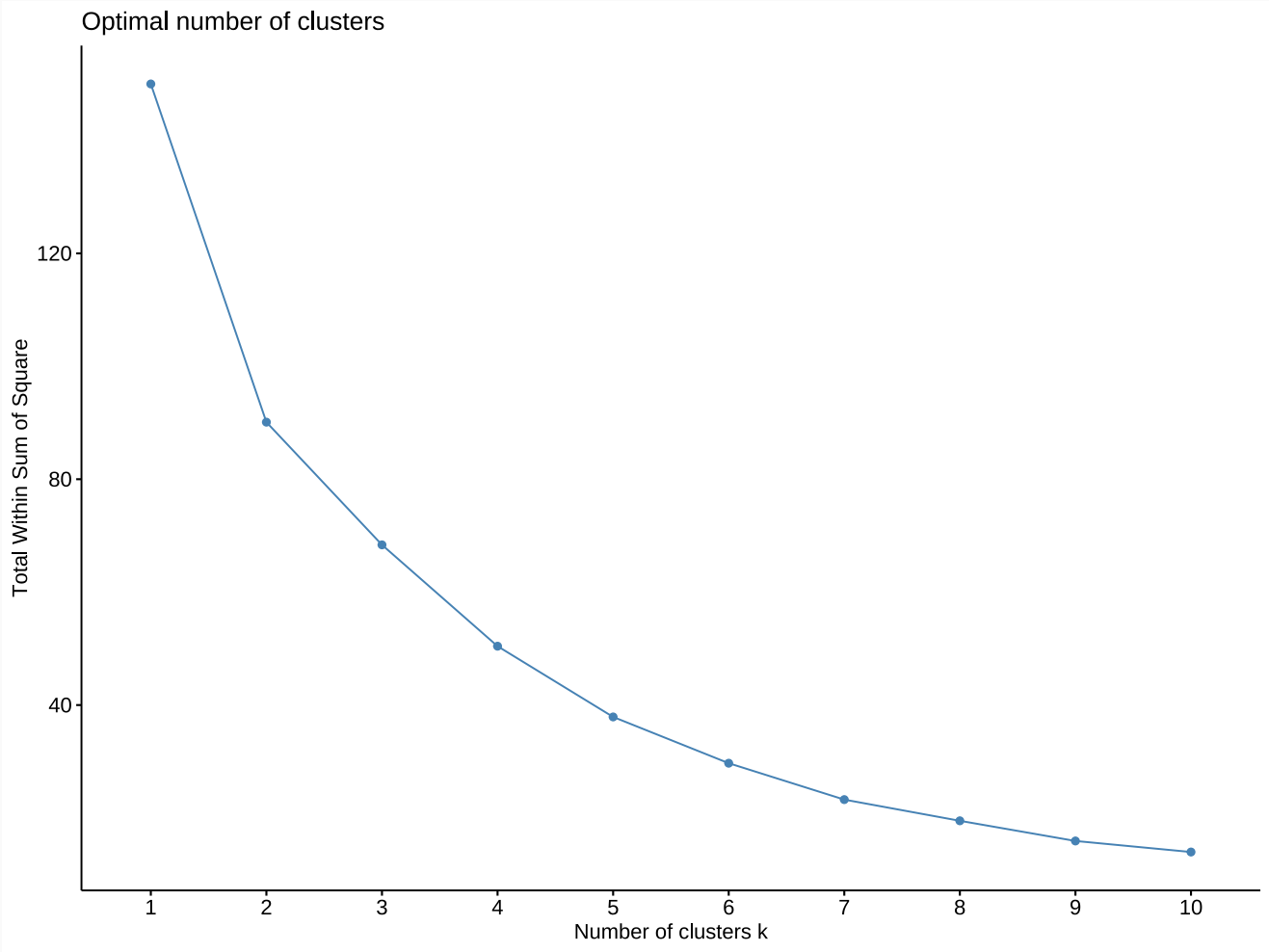
```
load("datas/hcluster.Rda")
df = dat %>%   # 数据标准化
  mutate(across(-1, ~ (.x - mean(.x)) / sd(.x))) %>%
  column_to_rownames("Region")
head(df, 3)
#>             X1    X2     X3     X4     X5
#> 全国 -0.283 0.047  1.163 0.0277 -0.392
#> 北京  3.385 3.622 -3.087 1.8665  1.637
#> 天津  2.140 1.610  0.261 1.6306  0.598
```

- 创建聚类任务

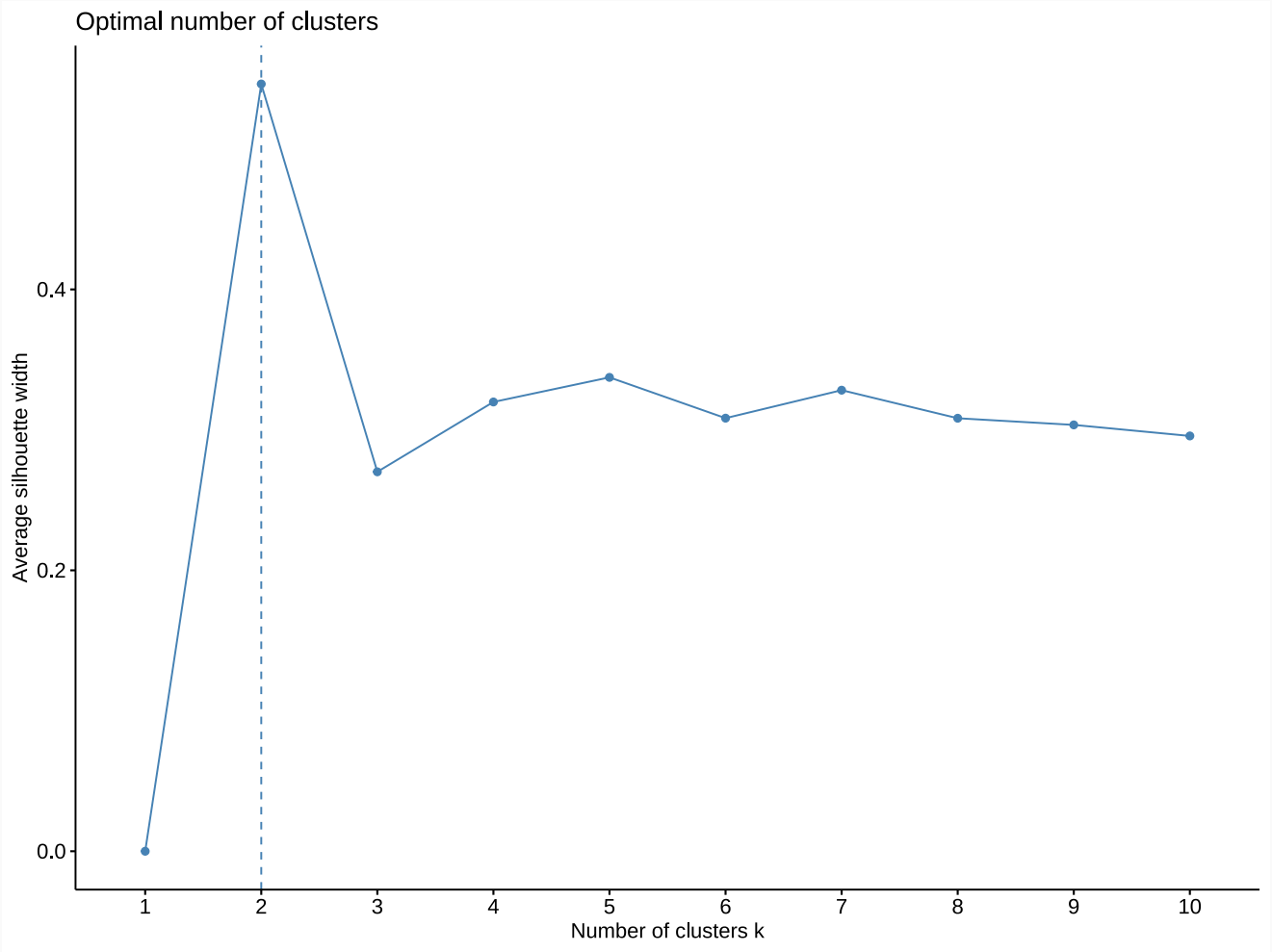
```
task = TaskClust$new("Employment", df)
task
#> <TaskClust:Employment> (31 x 5)
#> * Target: -
#> * Properties: -
#> * Features (5):
#>   - dbl (5): X1, X2, X3, X4, X5
```

(2) 确定最佳的 K

```
fviz_nbclust(df, FUN = hcut, method = "wss")
```



```
fviz_nbclust(df, FUN = hcut, method = "silhouette")
```

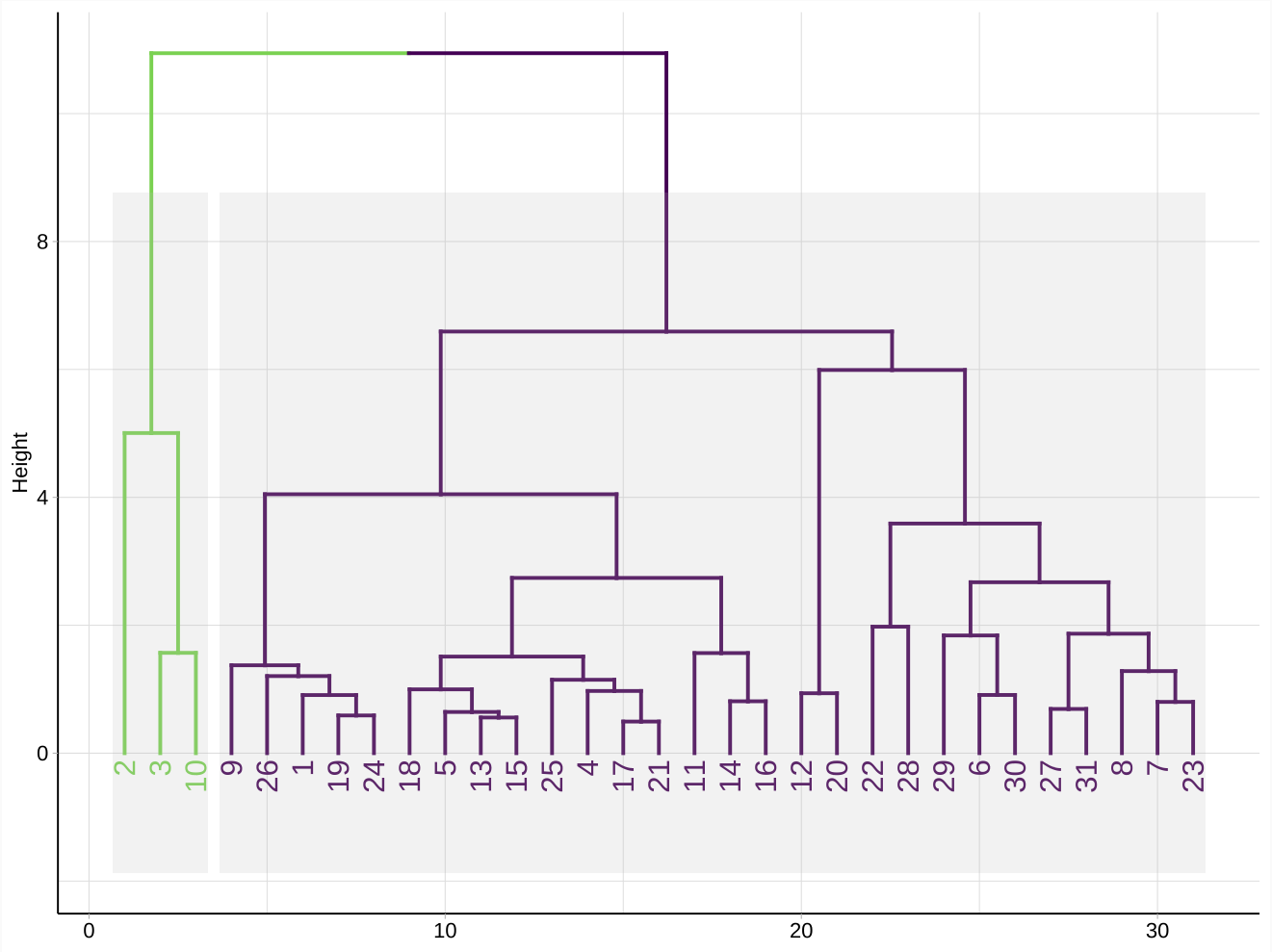


(3) 层次聚类

- 选择 clust.hclust 学习器，聚类数为 2, 聚合算法选择"ward.D2"

```
learner = lrn("clust.hclust", k = 2, method = "ward.D2")
learner$train(task)
```

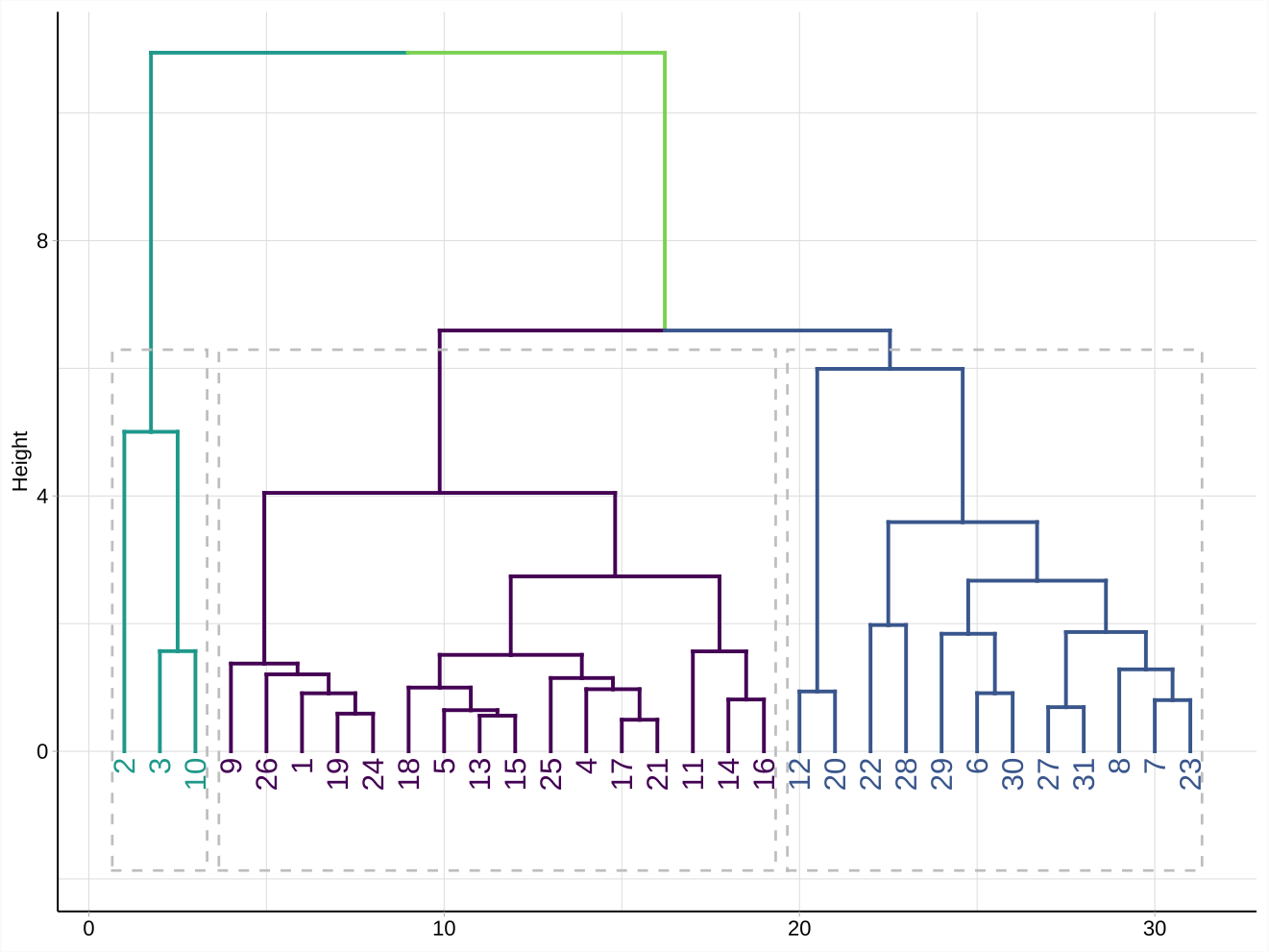
```r
autoplot(learner, k = 2, rect_fill = TRUE, rect = TRUE)
```



注：叶节点为行号，省份名标签暂时无法传入。

- 若剪切为 3 类:

```
autoplot(learner, k = 3, rect = TRUE)
```



四. DBSCAN

前文的聚类方法适合寻找球状或凸状聚类，且对噪声和异常值敏感。

有的数据可能是任意形状的聚类，比如椭圆、线型、S 型等，或者具有许多离群值和噪声，则适合采用基于密度的聚类。

基于密度的聚类以 DBSCAN 为代表，它不需要指定聚类数，可以找到任何形状的聚类，可以识别异常值。

DBSCAN 比较符合人的直观：连续的密度高的点聚为一类，被密度低的点隔开：

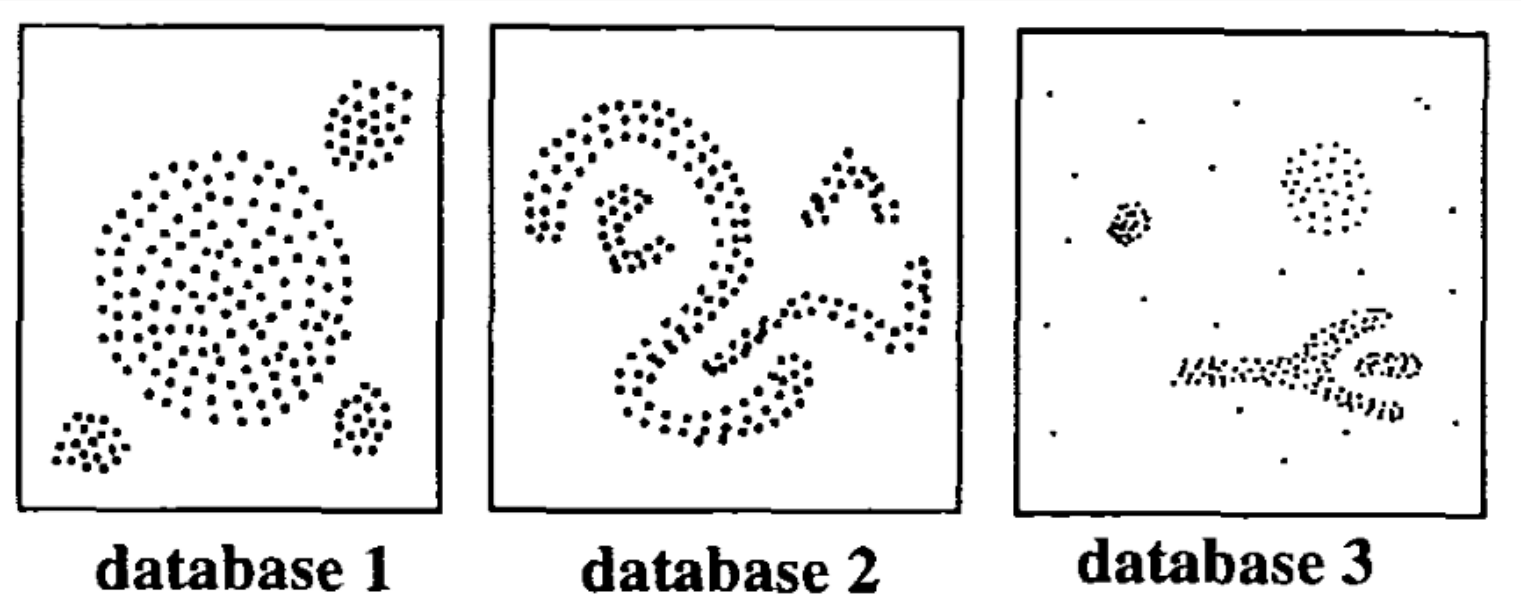


图 1: DBSCAN 法示意图

DBSCAN 的关键思想是，对于聚类中的每一个点，给定半径的邻域必须至少包含某最低数量的点。

1. 算法原理

目标是识别密集区域，这可以用接近某一点的点数来衡量。

DBSCAN 需要两个重要参数：

- 邻域半径 ε：考察的邻域范围
- 最小邻居数 `MinPts`：ε 半径内的最小邻居数

数据集中的任一点可以分为三类：

- 核心点：邻居数大于等于 `MinPts`
- 边界点：邻居数小于 `MinPts`，但它属于某核心点的 ε-邻域
- 噪声点或离群点：既不是核心点也不是边界点

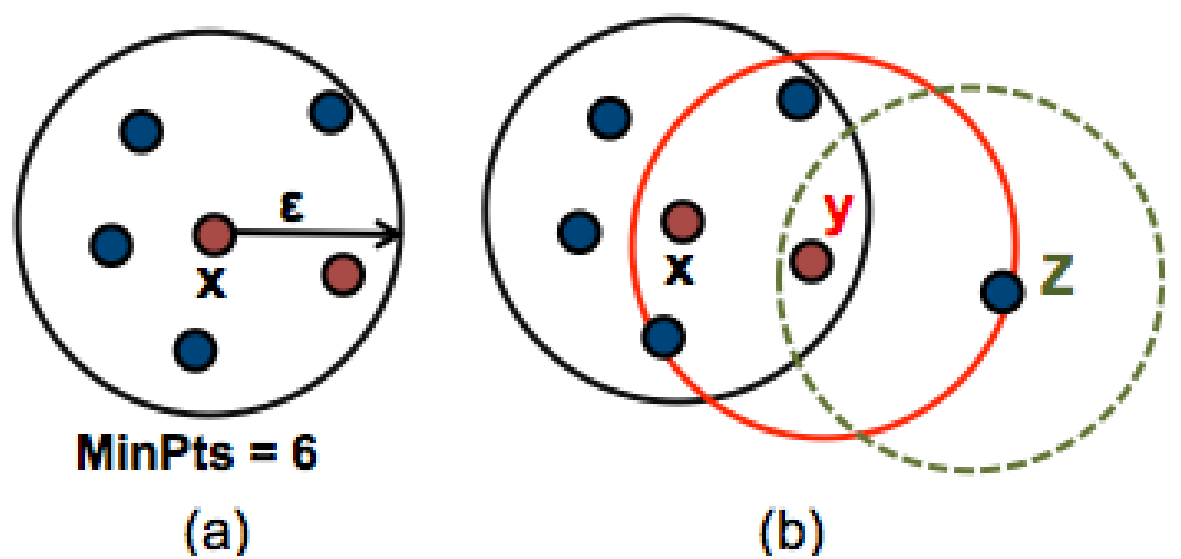


图 2: MinPts=6 的三类点示意图

- x 是核心点，因为其 ε-邻居数为 6
- y 是边界点，因为其 ε 邻居数小于 6，但它属于核心点 x 的 ε-邻域
- z 是噪声点

DBSCAN 算法需要用到三个术语：

- 点"A" 从另一个点"B" **直接密度可达**：若 (i) "A" 在"B "的 ε-邻域，(ii)"B" 是一个核心点。
- 点"A" 从"B" **密度可达**：若有一组核心点从"B" 通向"A"
- 两点"A" 和"B" 密度相连：若有一个核心点"C"，使"A" 和"B" 都能从"C" 密度到达

一个基于密度的聚类定义为一组密度相连的点。

DBSCAN 算法步骤：

- 对每个点 x_i，计算与其他点之间的距离。找出距离起点 x_i 的 ε-邻域内的所有邻居点。每个点，若其邻居数大于或等于 `MinPts`，则标记为核心点或已访问。
- 对每个核心点，若它还没有被分配到一个聚类，就创建一个新的聚类。递归地寻找其所有的密度相连点，并将它们分配到与核心点相同的聚类中。
- 遍历数据集中剩余的未访问的点，那些不属于任何聚类的点标记为离群点或噪声点。

2. DBSCAN 案例

- 使用来自 `factoextra` 包的模拟数据

```
library(factoextra)
data("multishapes")
df = multishapes[,1:2]
```

- 创建聚类任务

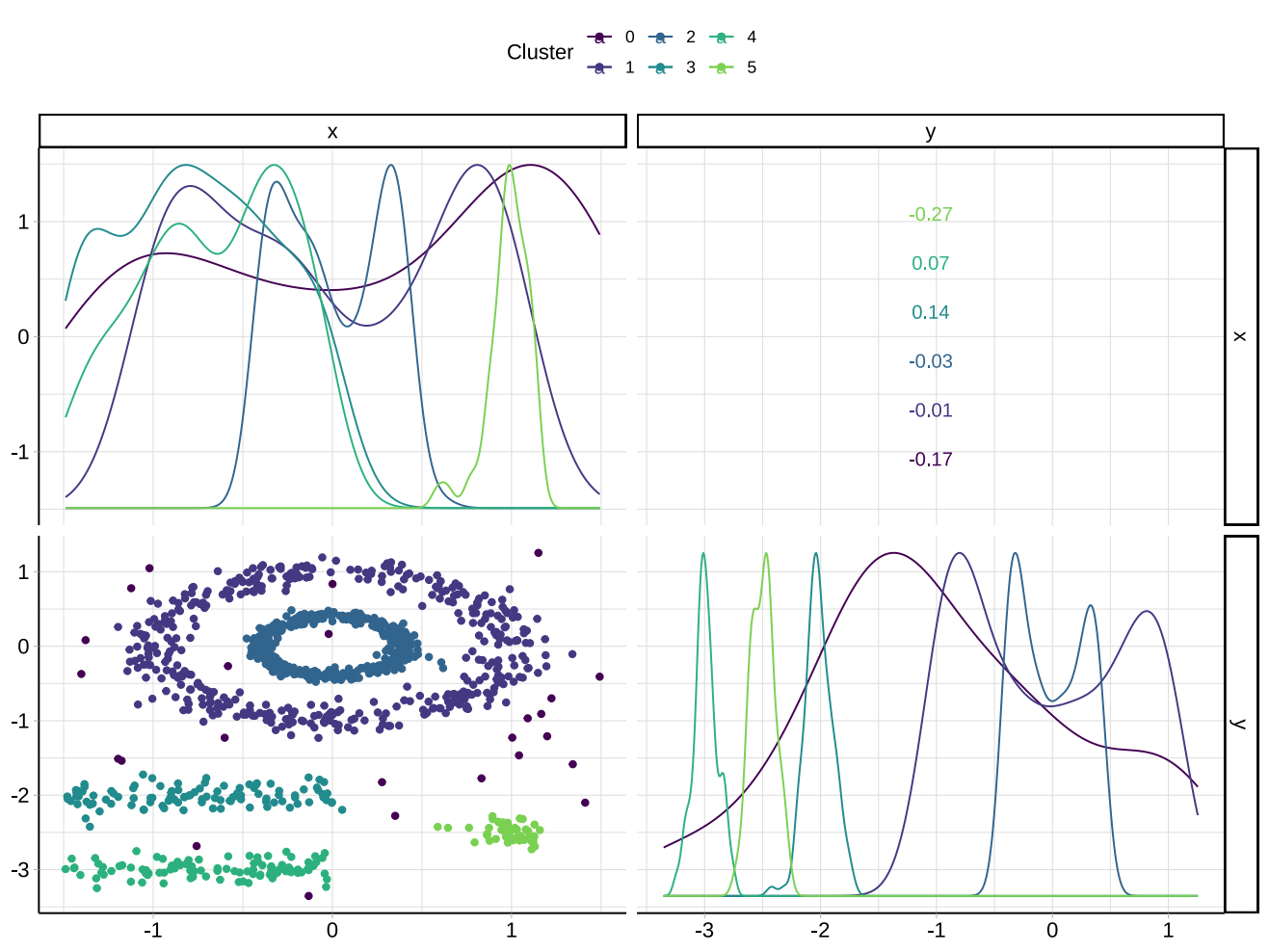
```
library(mlr3verse)
library(dbscan)
task = TaskClust$new("dbscan", df)
task
#> <TaskClust:dbscan> (1100 x 2)
#> * Target: -
#> * Properties: -
#> * Features (2):
#>   - dbl (2): x, y
```

- 选择学习器，训练模型

```
set.seed(123)
learner = lrn("clust.dbscan", eps = 0.15, minPts = 5)
learner$train(task)
```

- 可视化聚类效果

```
prediction = learner$predict(task)
autoplot(prediction, task)
```



主要参考文献

[1] Alboukadel Kassambara. Practical Guide To Cluster Analysis in R: Unsupervised Machine Learning. STHDA, 2017.

[2] Jim Liang(梁劲). Getting Started with Machine Learning, 2019.

[3] 刘顺祥. 从零开始学 Python: 数据分析与挖掘. 清华大学出版社, 2018.

[4] Alboukadel Kassambara. Practical Guide To Cluster Analysis in R, Unsupervised Machine Learning. STHDA.