

聚类算法

Clustering Algorithms

Mr. Black

目录

- K-means
- 层次聚类
- 基于密度的聚类

K-means

K-means

K-means是一种简单的迭代性的聚类算法。对于数据集 $D = \{x_1, x_2, \dots, x_n\}$ ，其中 $x_i \in \mathbb{R}^d$ ，需要指定利用K-means算法对数据划分成 k 个簇。对于数据集 D 的每个点 x_i 仅属于一个簇 S_i ，则K-means算法的目标函数可以表示为：

$$\operatorname{argmin}_S \sum_{i=1}^k \sum_{x \in S_i} \|x - \mu_i\|_2^2$$

其中 μ_i 是簇 S_i 的均值向量。从目标函数不难看出，K-means是通过一种“紧密程度”的形式对数据进行划分的，衡量这种“紧密程度”一般我们会用到“距离”的概念。距离可以理解为在集合 M 上的一个度量（Metric），即

$$\operatorname{dist} : M \times M \rightarrow \mathbb{R}$$

K-means

对于集合 M 中的 x, y, z , 下列条件均成立:

1. $dist(x, y) \geq 0$ (非负性)
2. $dist(x, y) = 0$, 当且仅当 $x = y$ (同一性)
3. $dist(x, y) = dist(y, x)$ (对称性)
4. $dist(x, z) \leq dist(x, y) + dist(y, z)$ (三角不等式)

对于点 $x = (x_1, x_2, \dots, x_n)$ 和点 $y = (y_1, y_2, \dots, y_n)$, 常用的距离为 p 阶明科夫斯基距离 (Minkowski distance) :

$$dist(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

K-means

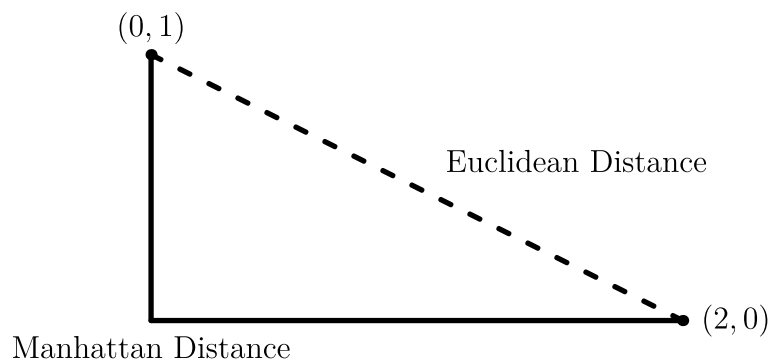
当 $p = 1$ 时，称之为曼哈顿距离（Manhattan distance）或出租车距离：

$$dist_{man}(x, y) = \sum_{i=1}^n |x_i - y_i|$$

当 $p = 2$ 时，称之为欧式距离（Euclidean distance）：

$$dist_{ed}(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

曼哈顿距离和欧式距离直观比较如图所示：



K-means

对于K-means算法，具体的计算过程如下：

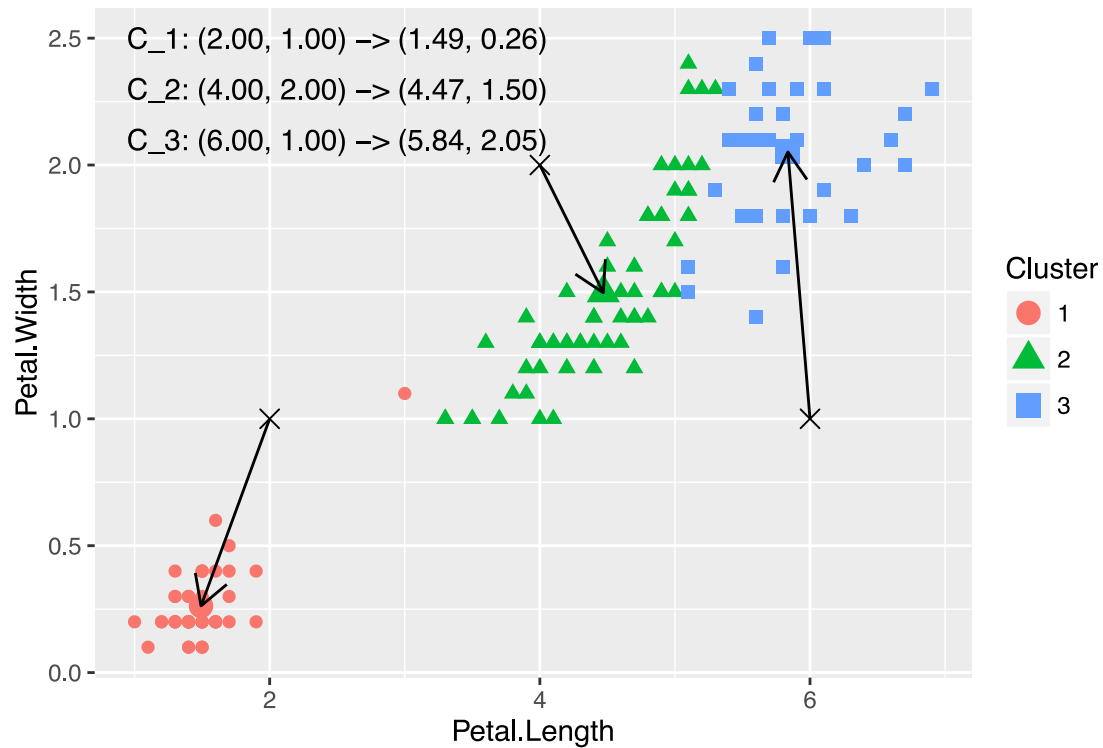
1. 指定簇的个数为 k ，并随机设置 k 个簇的中心，对于簇 S_i 其中心为 μ_i 。
2. 计算数据集 $D = \{x_1, x_2, \dots, x_n\}$ 中的所有点 x_j 到每个簇的中心 μ_i 的距离 $dist(x_j, \mu_i)$ 。
3. 对于点 x_j ，从其到每个簇中心 μ_i 的距离中选择距离最短的簇作为本轮计算中该点所隶属的簇。
4. 对于隶属于同一个簇的样本 D_{S_i} ，计算这些样本点的中心，作为该簇新中心 μ'_i 。
5. 重复执行步骤2到步骤4直至簇中心不再发生变化或超过最大迭代次数。

通过上述步骤的计算，K-means算法可以将样本点划分为 k 个簇，并得到每个簇的最终中心 μ_i 。

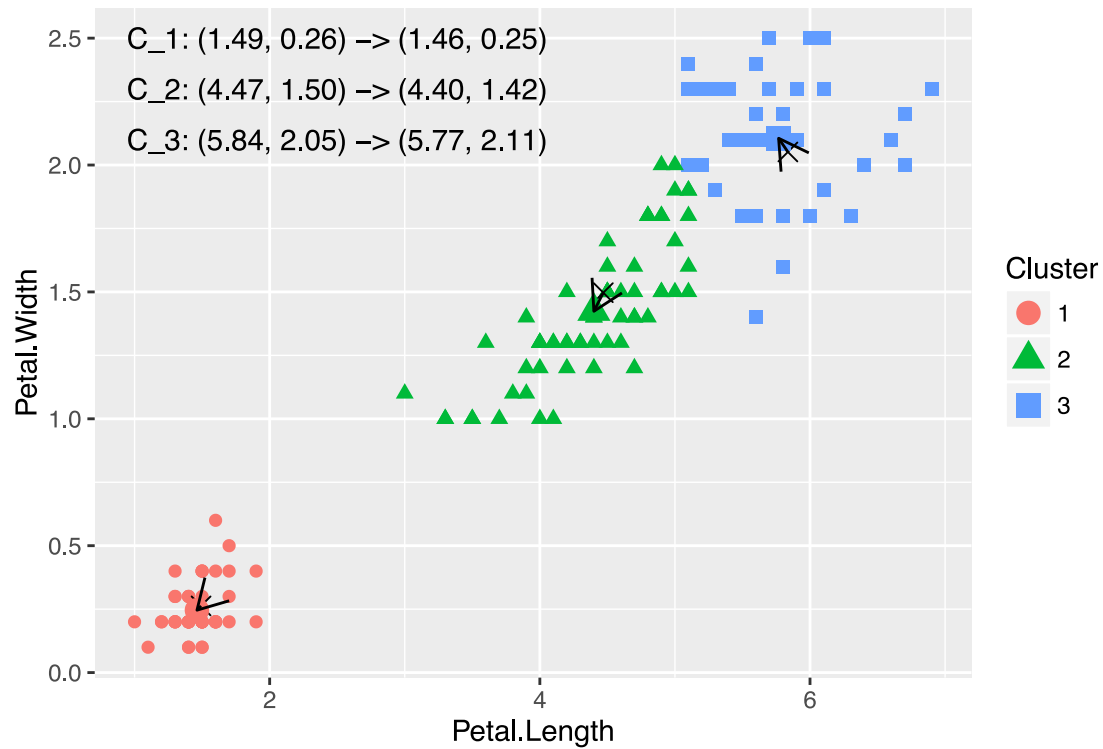
K-means

利用K-means算法，我们对iris数据集进行聚类分析。iris数据集包含了Sepal.Length, Sepal.Width, Petal.Length, Petal.Width以及花的种类共5列数据。为了能够更直观的演示，我们进采用Petal.Length和Petal.Width两列数据。K-means是一种无监督的学习算法，因此我们并没有先验知识知道数据最适合分为几个簇，同时K-means算法又是一个对于聚类中心初始点敏感的算法，因此同样为了便于演示效果，在此我们设置簇的个数 $k = 3$ ，3个簇对应的初始中心点分别为 $\mu_1 = (2, 1)$, $\mu_2 = (4, 2)$, $\mu_3 = (6, 1)$ 。

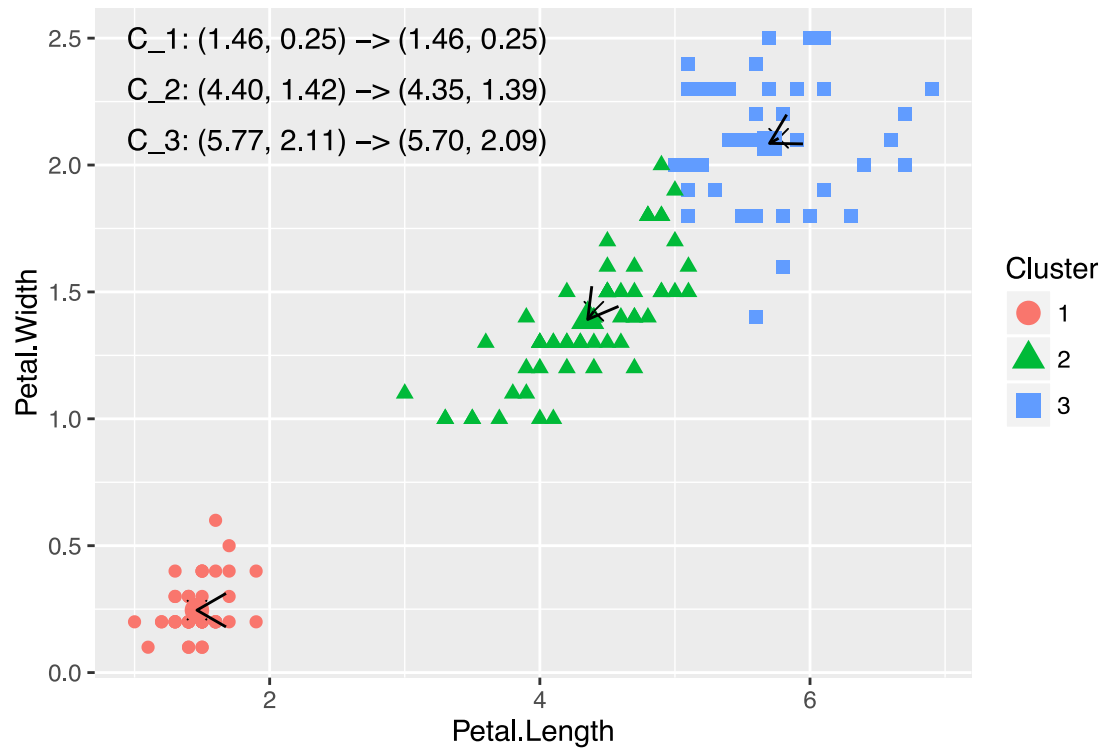
K-means



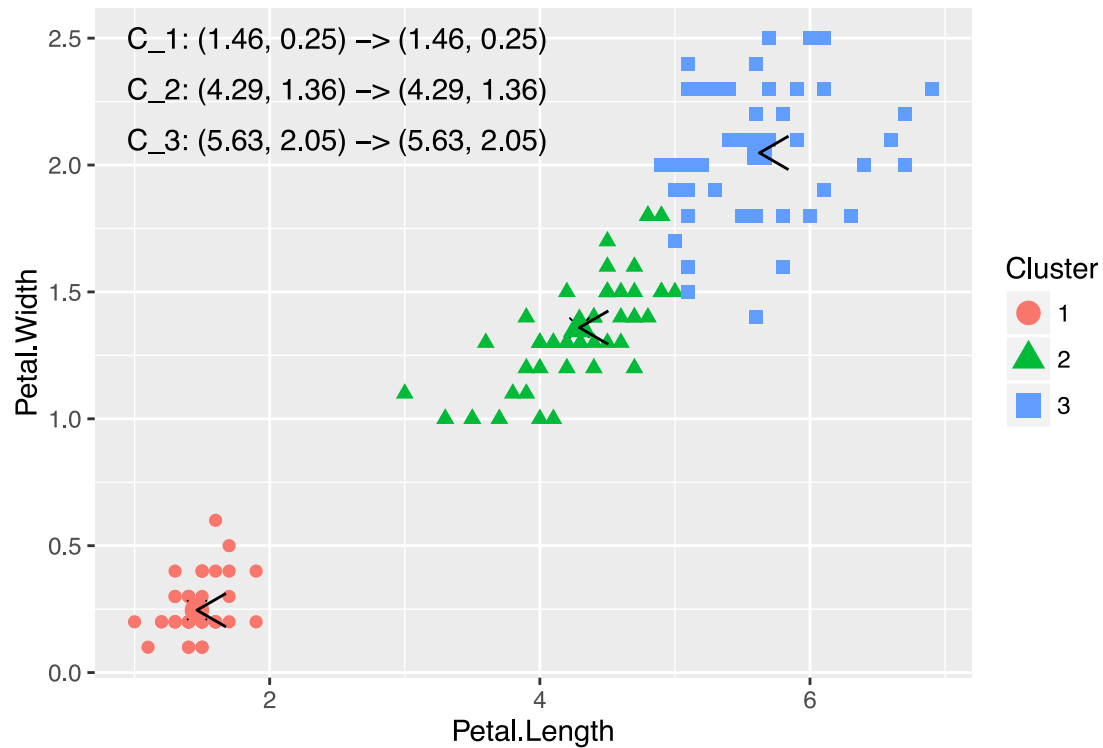
K-means



K-means



K-means



K-means

第1轮，第2轮，第3轮和第7轮（最终轮）计算得出的结果。其中每幅图左上角3组坐标分别表示了3个簇的中心更新前和更新后的位置。图中的x号即为更新前簇的中心，箭头指向的方向即为更新后簇的中心，每轮计算中隶属不同簇的样本点利用颜色和形状加以了区分。

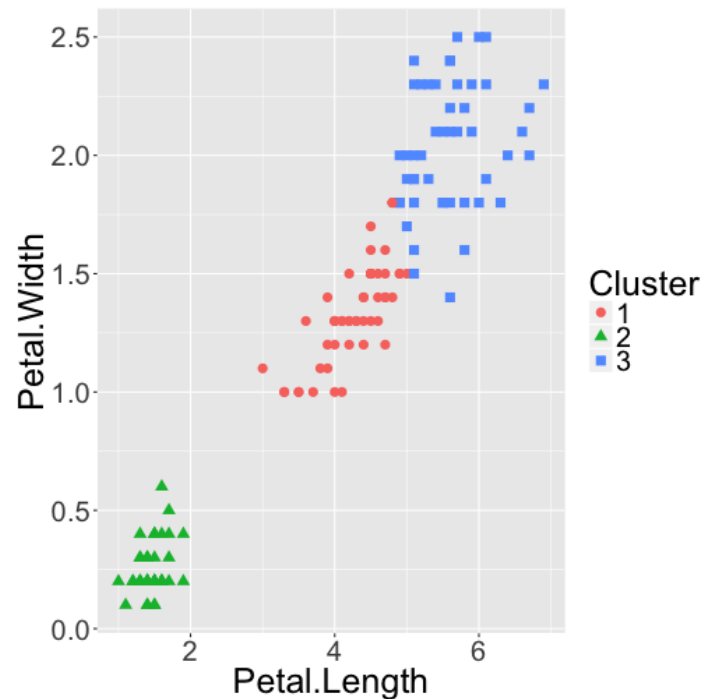
K-means算法在一般数据集上可以得到的较好的聚类效果，但同时也存在若干问题：

1. K-means算法需要预先设置聚类个数 k 。
2. K-means是一个对于簇中心点起始位置敏感的算法，设置不同的簇中心点的起始位置可能得到不同的聚类结果。
3. 噪音数据对K-means算法的聚类结果影响较大。
4. 只能发现球状簇。

K-means

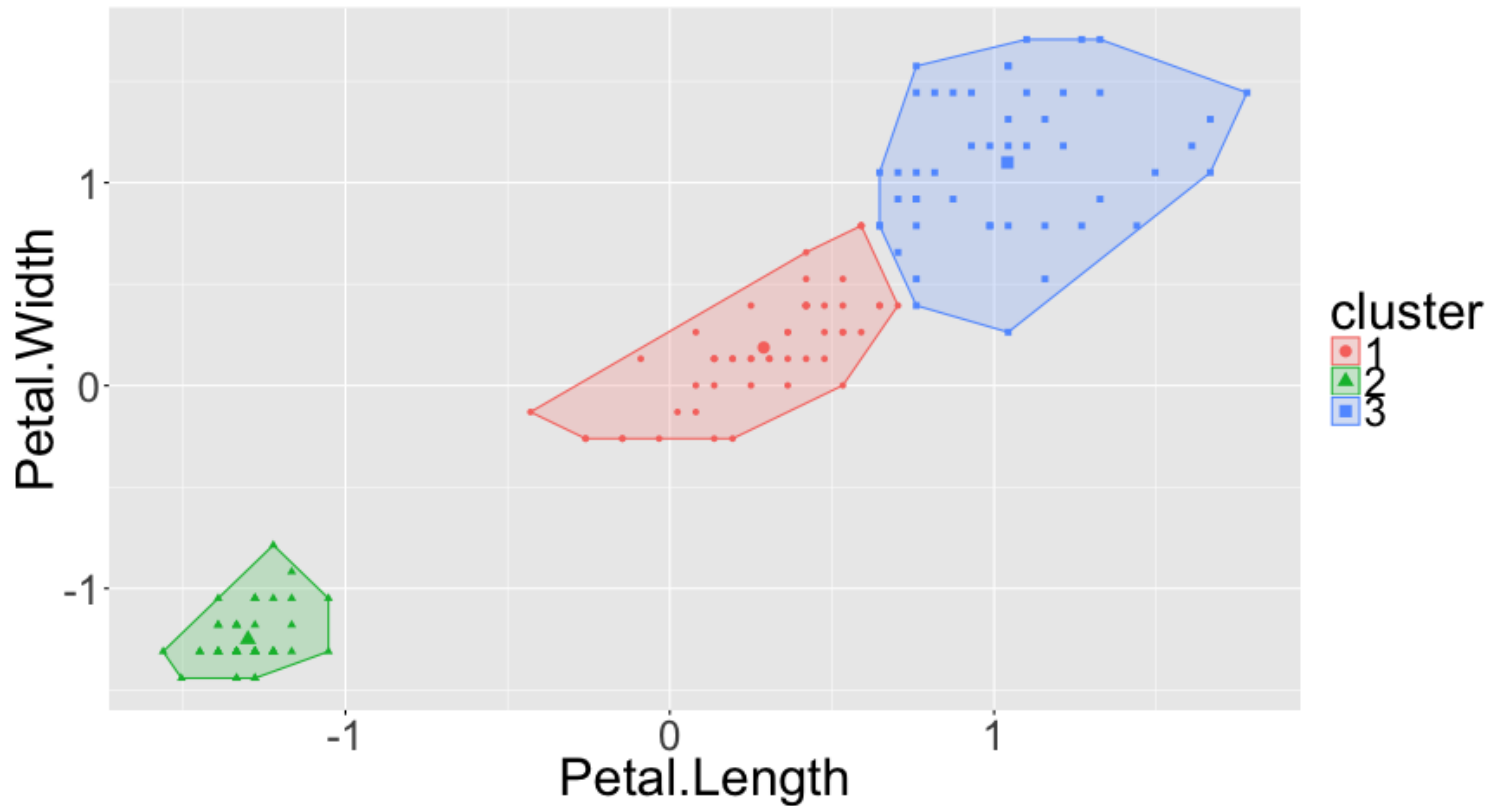
```
kmeans(x, centers, iter.max = 10, nstart = 1,  
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",  
                     "MacQueen"), trace=FALSE)
```

```
iris_ <- iris[, 3:4]  
iris_kmeans <- kmeans(iris_, 3,  
                      nstart=25)  
iris_res <- dplyr::bind_cols(  
  iris_,  
  Cluster=as.factor(  
    iris_kmeans$cluster))  
p <- ggplot(iris_res,  
            aes(Petal.Length,  
                Petal.Width)) +  
  geom_point(aes(  
    color = Cluster,  
    shape = Cluster))  
print(p)
```



K-means

```
factoextra::fviz_cluster(iris_kmeans, data=iris_, geom=c('point'))
```



层次聚类

层次聚类

层次聚类（hierarchical clustering）不同于K-means那种基于划分的聚类，通过对数据集在不同层次上进行划分，直至达到某种条件。层次聚类根据分层的方法不同，可以分为凝聚（agglomerative）层次聚类和分裂（divisive）层次聚类。

AGNES（Agglomerative NESting）算法是一种凝聚层次聚类算法，其基本思想如下：

1. 将数据集中每个样本作为一个簇。
2. 在每一轮计算中，找出两个距离最近的簇进行合并，生成一个新的簇。
3. 重复步骤2，直至达到预设的聚类簇的个数。

层次聚类

因此，对于AGNES算法而言，最关键的是如何计算两个簇之间的距离，对于簇 C_i 和簇 C_j ，常用的距离计算方法有：

- 最小距离，即两个簇内部样本点之间距离的最小值：

$$dist_{min} = \min\{dist(x, y) | x \in C_i, y \in C_j\}$$

- 最大距离，即两个簇内部样本点之间距离的最大值：

$$dist_{max} = \max\{dist(x, y) | x \in C_i, y \in C_j\}$$

- 平均距离，即两个簇内部样本点之间距离的均值：

$$dist_{avg} = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} dist(x, y)$$

- 重心距离，即两个簇重心之间的距离：

$$dist_{med} = dist(Median_{C_i}, Median_{C_j})$$

层次聚类

DIANA (Divisive Analysis) 算法指一种分裂层次聚类算法，其基本思想如下：

1. 将数据集中全部样本作为一个簇。
2. 在每一轮计算中，对于“最大”的簇 C ，找到 C 中与其他点的平均相异度最大的点 p_0 ，将其放在一个新的簇 C_{new} 中，剩余的点此时所组成的簇为 C_{old} 。
3. 在簇 C_{old} 中找到一个距离簇 C_{new} 最近，且距离小于到簇 C_{old} 的点 p_i ，并将其加入到簇 C_{new} 中。
4. 重复步骤3，直至无法找到符合条件的点 p_i ，此时得到两个新簇 C_{old} 和 C_{new} 。
5. 重复步骤2和步骤3，直至达到预设的聚类簇的个数。

在DIANA算法中，衡量一个簇 C 的大小，一般利用簇的直径，即簇中任意两个样本之间距离的最大值；衡量簇 C 中一个点 p 的平均相异度，一般利用该点到簇中其他点距离的平均值。

层次聚类

以R中cluster扩展包中的animals数据集为例， animals数据集记录了20中不同昆虫和动物的6种属性值，数据示例如表所示，列分别为：温血， 会飞， 脊椎动物， 濒危， 群居动物， 有毛发：

样本 \ 特征	war	fly	ver	end	gro	hai
ant	1	1	1	1	2	1
bee	1	2	1	1	2	2
cat	2	1	2	1	1	2
...
spi	1	1	1	NA	1	2
wha	2	1	2	2	2	1

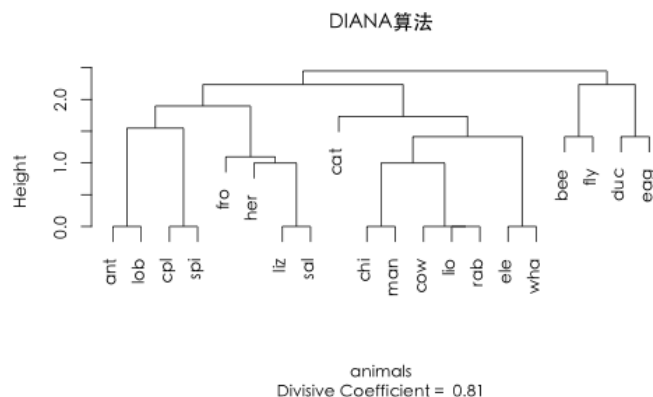
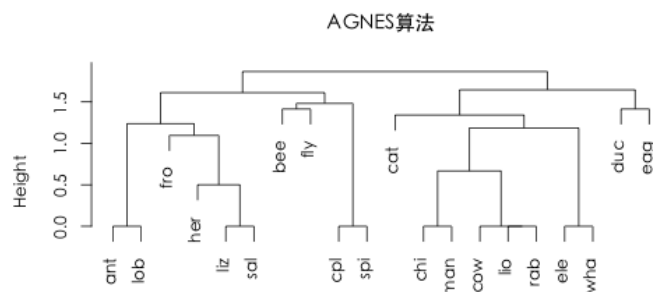
层次聚类

利用AGNES和DIANA算法对animals数据集进行层次聚类分析，可以得到层次聚类分析的树状图（dendrogram），如图所示：

```
require(cluster)
data('animals')

animals_agnes <- agnes(animals)
plot(animals_agnes,
     which.plot=2,
     main='AGNES算法')

animals_diana <- diana(animals)
plot(animals_diana,
     which.plot=2,
     main='DIANA算法')
```



基于密度的聚类

基于密度的聚类

基于密度的聚类（density-based clustering）是一种通过样本的稠密程度划分聚类簇的方法。不同于基于距离的K-means和层次聚类方法往往只能生成球状的聚类簇，基于密度的聚类可以发现任意形状的聚类簇。

DBSCAN（density-based spatial clustering of applications with noise）是一种基于密度的聚类算法。DBSCAN算法最重要的两个参数为 ϵ 和 $MinPts$ ，两个参数分别确定了领域半径和定义了核心点的阈值，通过这两个参数可以刻画样本分布的稠密程度。对于数据集 $D = \{x_1, x_2, \dots, x_n\}$ ，引入如下概念和记号：

- ϵ 邻域（ ϵ neighborhood）

$$N_{\epsilon}(x) = \{y \in X | dist(x, y) \leq \epsilon\}$$

对于 $x \in D$ ，称 $N_{\epsilon}(x)$ 为 x 的 ϵ 邻域。

基于密度的聚类

- 密度 (density)

$$\rho(x) = |N_\epsilon(x)|$$

对于 $x \in D$, 称 $\rho(x)$ 为 x 的密度。

- 核心点 (core point)

对于 $x \in D$, 若 $\rho(x) \geq MinPts$, 则称 x 为一个核心点。假设 D 中所有核心点构成的集合为 D_{core} , 记 $D_{n-core} = D \setminus D_{core}$ 为所有非核心点的集合。

- 边界点 (border point)

对于 $x \in D_{n-core}$, 且 $\exists y \in D$, 满足

$$y \in N_\epsilon(x) \cap D_{core}$$

即点 x 所在的 ϵ 邻域中存在核心点, 则称 x 为 D 的边界点, 记所有的边界点的集合为 D_{border} 。

基于密度的聚类

- 噪音点 (noise point)

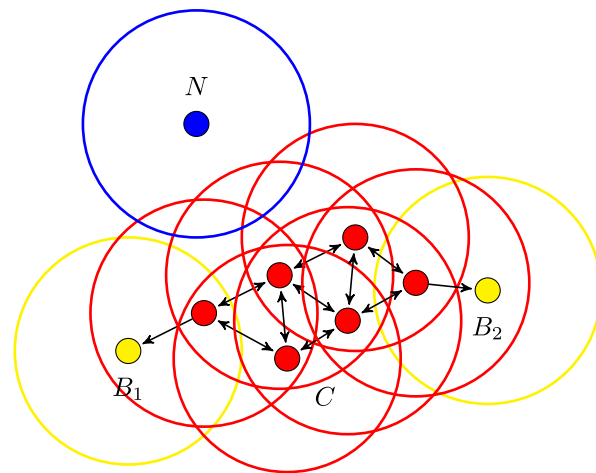
记 $D_{noise} = D \setminus (D_{core} \cup D_{border})$, 对于 $x \in D_{noise}$, 则称 x 为噪音点。

核心点, 边界点和噪音点示例如右图所示:

其中 C 为6个核心点, B_1 和 B_2 为2个边界点, N 为1个噪音点。

- 密度直达 (directly density-reachable)

对于 $x, y \in D$, 若 $x \in D_{core}$, 并且 $y \in N_\epsilon(x)$, 则称 y 由 x 密度值达。



基于密度的聚类

- 密度可达 (density-reachable)

若存在一个序列 $p_1, p_2, \dots, p_m \in D$, 满足 p_{i+1} 由 p_i 密度直达, 则称 p_m 由 p_1 密度可达。

- 密度相连 (density-connected)

对于 $x, y, z \in D$, 若 y 和 z 均由 x 密度可达, 则称 y 和 z 密度相连。

- 簇 (cluster)

对于非空子集 $C \in D$, 如果称 C 为一个簇, 则对于 $x, y \in C$ 满足:

1. 连接性 (connectivity) : 对于 $x, y \in C$, 则 x 和 y 密度相连。
2. 最大性 (maximality) : 对于 $x \in C$, 且 y 由 x 密度可达, 则 $y \in C$ 。

根据如上概念, DBSCAN算法的基本为: 从一个核心点 x 出发, 寻找到 x 密度可达的所有样本点的集合 $X = \{x' \in D | x' \text{ 由 } x \text{ 密度可达}\}$, 则 X 即为一个满足要求的簇。

基于密度的聚类

DBSCAN算法如下：

Algorithm 1 DBSCAN算法

Require: 数据集 d , 参数 $(\epsilon, minpts)$

Ensure: 簇划分 $c = \{c_1, c_2, \dots, c_k\}$

```
1: procedure DBSCAN( $d, \epsilon, minpts$ )
2:   初始化核心对象集合:  $i \leftarrow \emptyset$ 
3:   for  $i = 1$  to  $n$  do
4:     对于样本 $x_i$ , 生成 $\epsilon$ 邻域 $n_\epsilon(x_i)$ 
5:     if  $\rho(x_i) \geq minpts$  then
6:        $i \leftarrow i \cup \{x_i\}$ 
7:     end if
8:   end for
9:   初始化聚类个数:  $k \leftarrow 0$ 
10:  初始化未访问到集合:  $u \leftarrow d$ 
11:  #接下文
12: end procedure
```

```
1: procedure DBSCAN( $d, \epsilon, minpts$ )
2:   #接上文
3:   while  $i \neq \emptyset$  do
4:     当前为访问的样本集合:
        $u_{old} \leftarrow u$ 
5:     随机选取一个核心点 $p \in i$ , 并
       初始化一个队列 $q \leftarrow \{p\}$ 
6:      $u \leftarrow u \setminus \{p\}$ 
7:     while  $q \neq \emptyset$  do
8:        $q \leftarrow q$ 的队首
9:       if  $\rho \geq minpts$  then
10:         $r \leftarrow n_\epsilon(q) \cap u$ 
11:         $q \leftarrow q \cup r$ 
12:         $u \leftarrow u \setminus r$ 
13:      end if
14:    end while
15:     $k \leftarrow k + 1$ 
16:    生成簇 $c_k \leftarrow u_{old} \setminus u$ 
17:     $i \leftarrow i \setminus c_k$ 
18:  end while
19:  return  $c = \{c_1, c_2, \dots, c_k\}$ 
20: end procedure
```

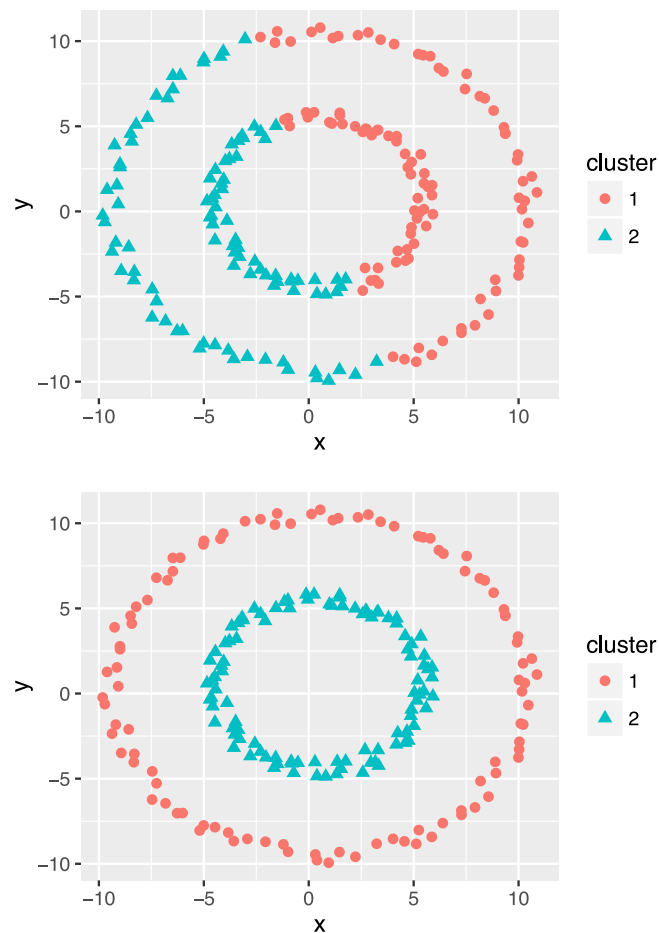
基于密度的聚类

相比K-means算法，DBSCAN算法有如下优势：

1. 不需要事先指定簇的个数 k 。
2. 可以发现任意形状的簇。
3. 对噪音数据不敏感。

尽管相比K-means，DBSCAN算法有很多优势，但是对于不同的数据集，DBSCAN算法的参数 ϵ 和 $MinPts$ 有时很难选取和优化。

一个非球形簇的数据分别利用DBSCAN算法和K-means算法进行聚类分析，对比结果如图所示：



基于密度的聚类

```
dbscan(x, eps, minPts = 5, weights = NULL, borderPoints = TRUE, ...)
```

```
library(dbscan)
```

```
# 构造数据
```

```
set.seed(123)
```

```
a <- seq(0, 2*pi, by = pi/50)
```

```
x_1 <- 10 * cos(a) + runif(length(a), 0, 1)
```

```
y_1 <- 10 * sin(a) + runif(length(a), 0, 1)
```

```
x_2 <- 5 * cos(a) + runif(length(a), 0, 1)
```

```
y_2 <- 5 * sin(a) + runif(length(a), 0, 1)
```

```
points <- data.frame(
```

```
  x = c(x_1, x_2),
```

```
  y = c(y_1, y_2)
```

```
)
```

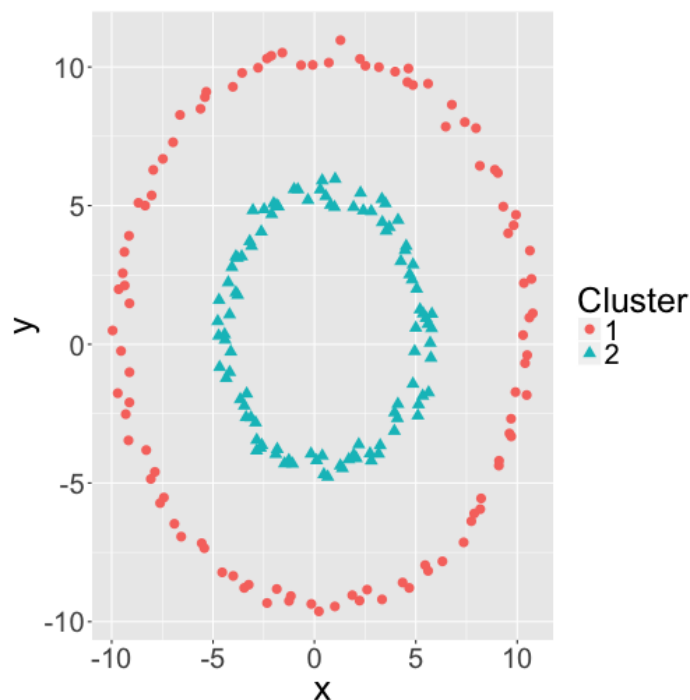
```
# 聚类
```

```
points_dbscan <- dbscan(points, eps=3.5, minPts=10)
```

基于密度的聚类

```
# 构造绘图数据
points_plt <- dplyr::bind_cols(
  points,
  Cluster=as.factor(
    points_dbscan$cluster))

# 绘图
p <- ggplot(points_plt,
  aes(x, y)) +
  geom_point(
    aes(color = Cluster,
      shape = Cluster))
print(p)
```



Thanks



本作品采用 **CC BY-NC-SA 4.0** 进行许可