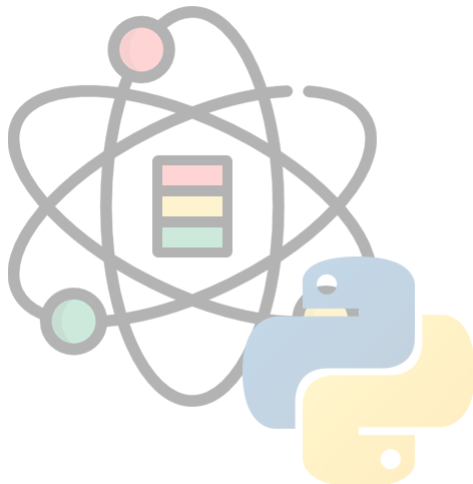
Python 数据科学导论
Data Science Introduction with Python

聚类算法
Clustering Algorithms

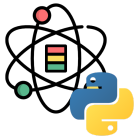
范叶亮



目录

K-means

K-means



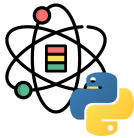
K-means 是一种简单的迭代性的聚类算法。对于数据集 $D = \{x_1, x_2, \ldots, x_n\}$，其中 $x_i \in \mathbb{R}^d$，需要指定利用 K-means 算法对数据划分成 k 个簇。对于数据集 D 的每个点 x_i 仅属于一个簇 S_i，则 K-means 算法的目标函数可以表示为：

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

其中 μ_i 是簇 S_i 的均值向量。从的目标函数不难看出，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

$$dist : M \times M \to \mathbb{R} \tag{2}$$

K-means



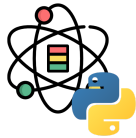
对于集合 M 中的 x, y, z，下列条件均成立：

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

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

$$dist\left(x, y\right) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{3}$$

K-means



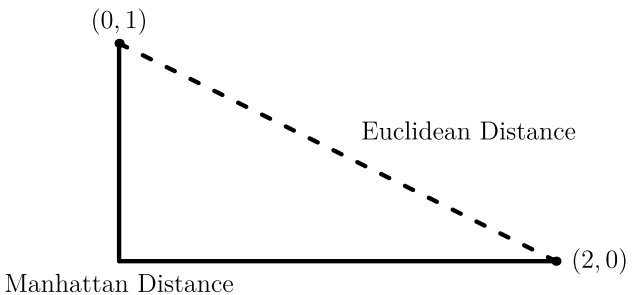
当 $p=1$ 时，称之为曼哈顿距离（Manhattan distance）或出租车距离：

$$dist_{man}\left(x,y\right)=\sum_{i=1}^{n}\left|x_i-y_i\right| \tag{4}$$

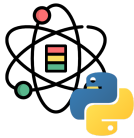
当 $p=2$ 时，称之为欧式距离（Euclidean distance）：

$$dist_{ed}\left(x,y\right)=\sqrt{\sum_{i=1}^{n}\left(x_i-y_i\right)^2} \tag{5}$$

曼哈顿距离和欧式距离直观比较如图所示：



K-means



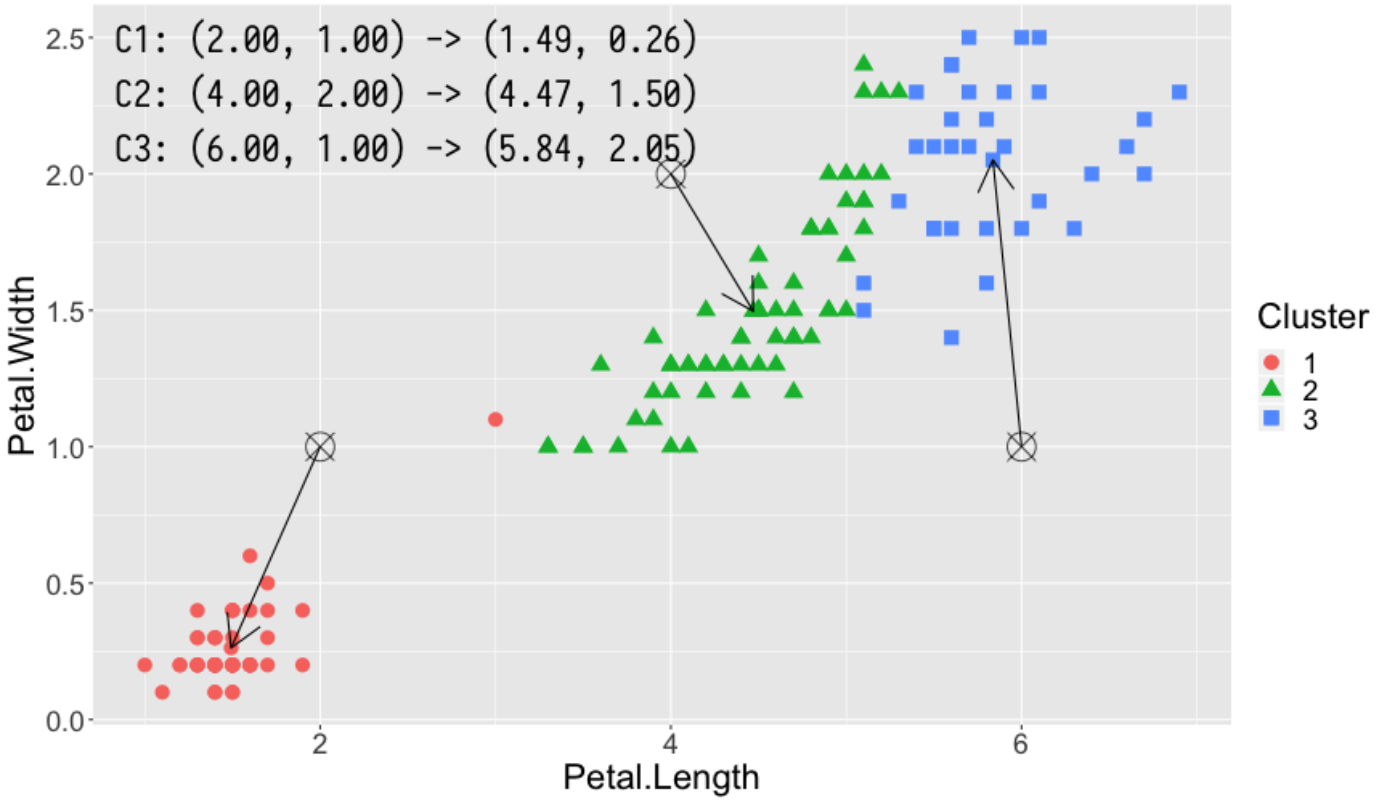
对于 K-means 算法，具体的计算过程如下：

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 $D = \{x_1, x_2, \ldots, x_n\}$ 中的所有点 x_j 到每个簇的中心 μ_i 的距离 $dist\left(x_j, \mu_i\right)$。
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。

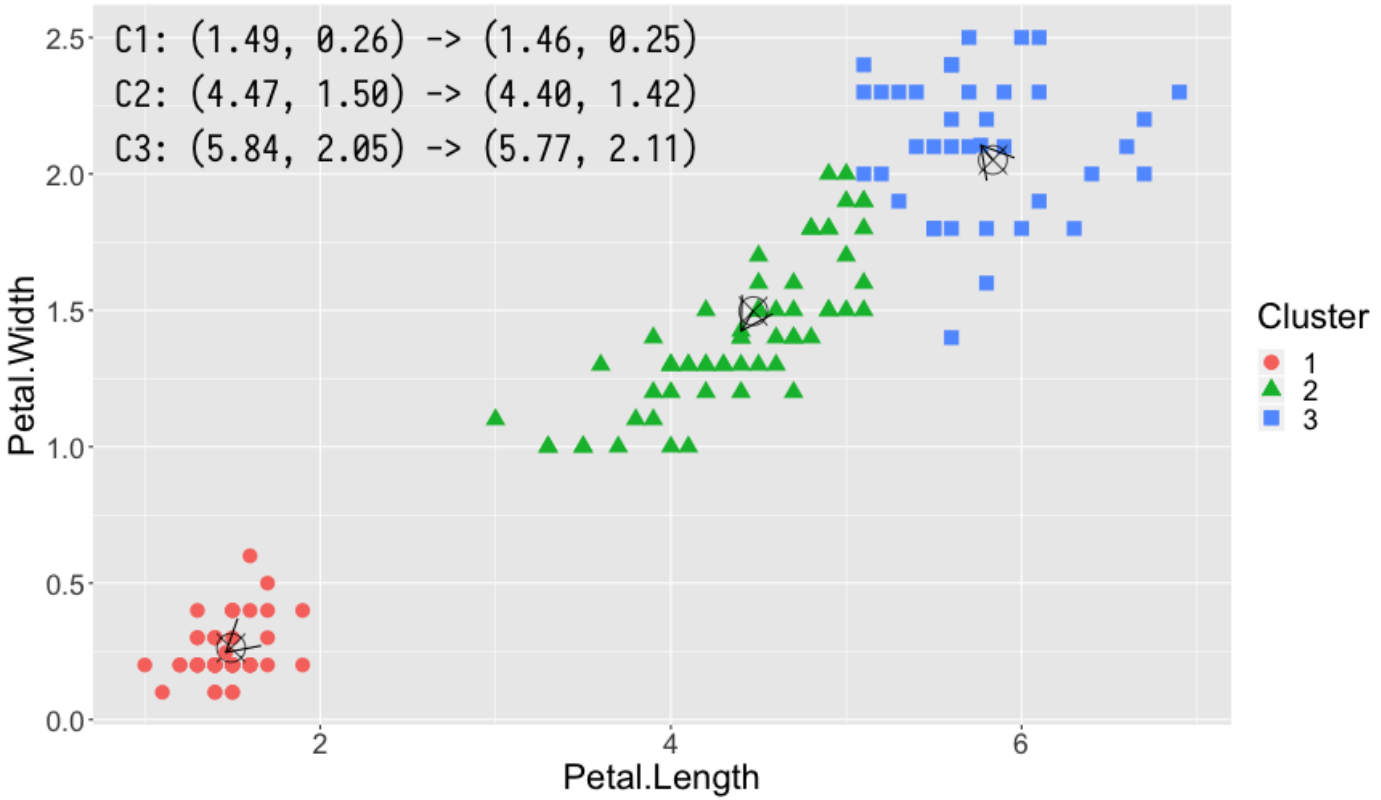
利用 K-means 算法，我们对 iris 数据集进行聚类分析。iris 数据集包含了 Sepal.Length，Sepal.Width，Petal.Length，Petal.Width 以及花的种类共 5 列数据。为了能够更直观的演示，我们仅采用 Petal.Length 和 Petal.Width 两列数据。K-means 是一种无监督的学习算法，因此我们并没有先验知识知道数据最适合分为几个簇，同时 K-means 算法又是一个对于聚类中心初始点敏感的算法，因此同样为了便于演示效果，在此我们设置簇的个数 $k = 3$， 3 个簇对应的初始中心点分别为 $\mu_1 = (2, 1), \mu_2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

K-means



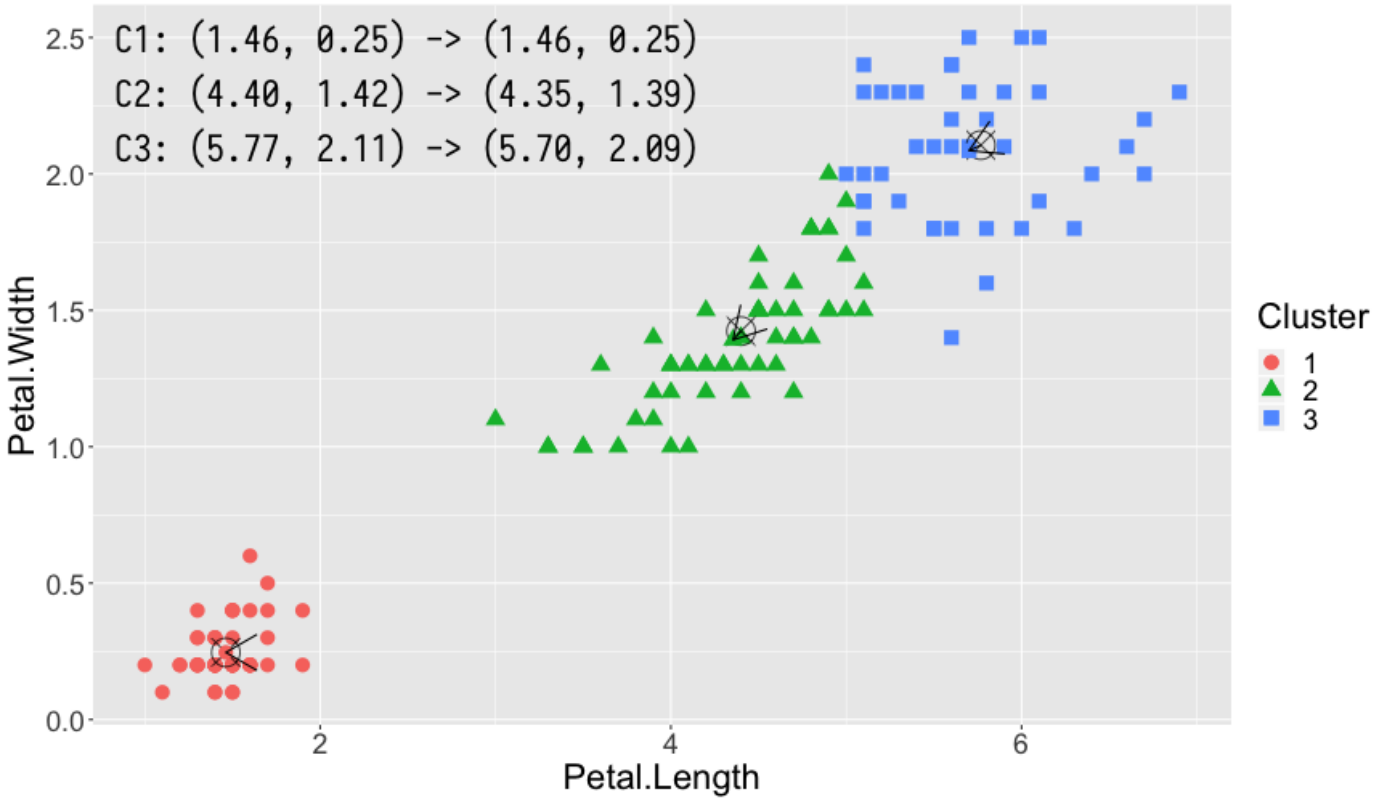
K-means 第 1 轮

K-means



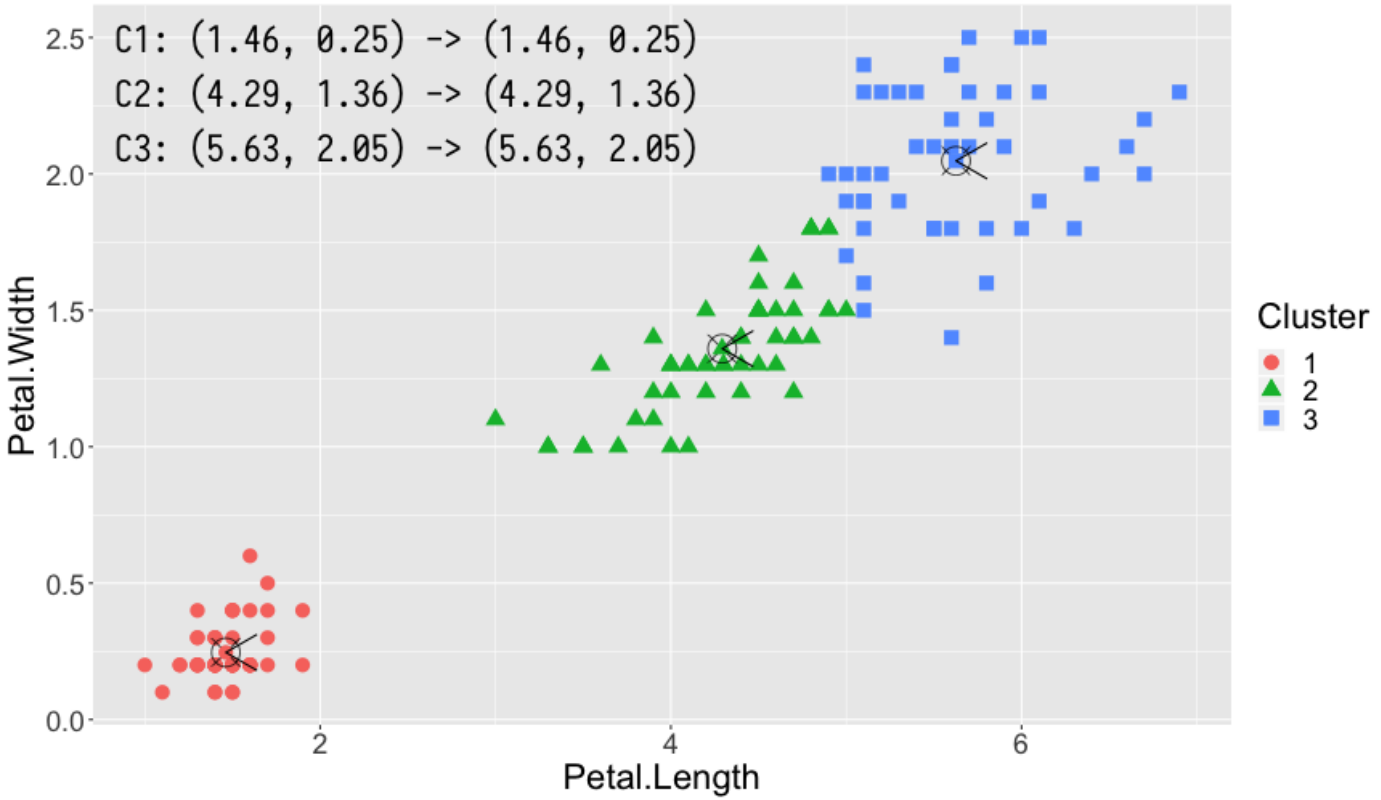
K-means 第 2 轮

K-means



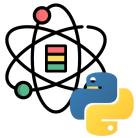
K-means 第 3 轮

K-means



K-means 第 7 轮

K-means

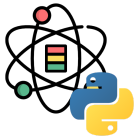


第 1 轮，第 2 轮，第 3 轮和第 7 轮（最终轮）计算得出的结果。其中每幅图左上角 3 组坐标分别表示了 3 个簇的中心更新前和更新后的位置。图中的 ⊠ 号即为更新前簇的中心，箭头指向的方向即为更新后簇的中心，每轮计算中隶属不同簇的样本点利用颜色和形状加以了区分。

K-means 算法在一般数据集上可以的到较好的聚类效果，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

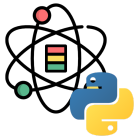


```
sklearn.cluster.KMeans(
    n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0,
    random_state=None, copy_x=True, n_jobs=None, algorithm='auto')
```

常用的参数如下表所示:

参数	描述
n_clusters	聚类个数
init	初始化方法，'k-means++'，'random' 或指定位置
n_init	利用 centroid seeds 运行算法的次数
max_iter	最大迭代次数
algorithm	K-means 使用的算法，'auto'，'full'，'elkan'

K-means

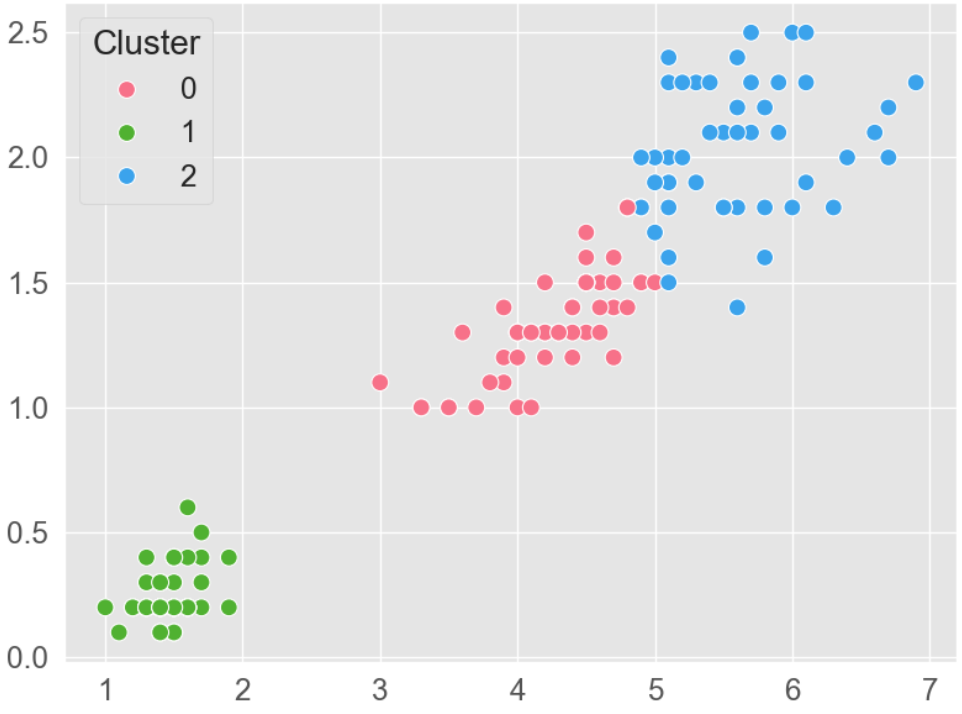


```
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
y_pred = KMeans(n_clusters=3, random_state=112358)\
  .fit_predict(X[:, 2:])

plot_df = pd.DataFrame(
  {'x': X[:, 2], 'y': X[:, 3], 'h': y_pred})
sns.scatterplot(
  data=plot_df, x='x', y='y', s=100, hue='h',
  palette=sns.color_palette('husl', 3))
plt.legend().set_title('Cluster')
plt.show()
```



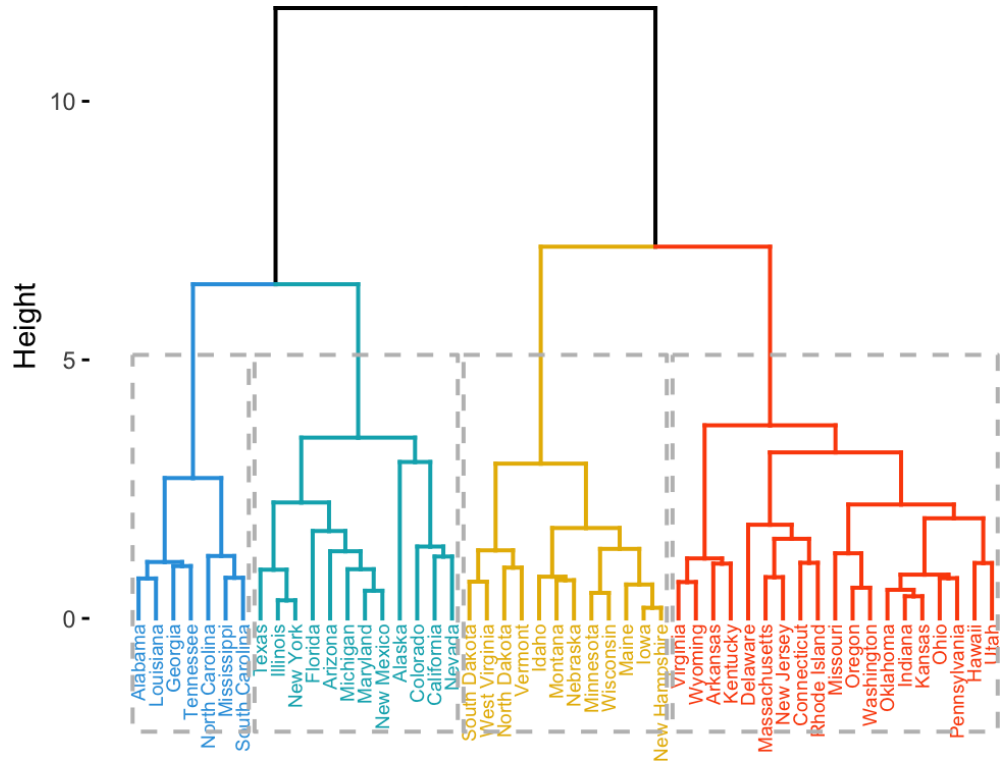
层次聚类

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。

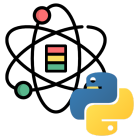
AGNES（Agglomerative Nesting）算法是一种凝聚层次聚类算法，其基本思想如下：

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。



[1] 图片来源：https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/

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：

$$dist_{min} = \min\{dist\,(x,y)|x \in C_i, y \in C_j\} \tag{6}$$

- 最大距离，即两个簇内部样本点之间距离的最大值：

$$dist_{max} = \max\{dist\,(x,y)|x \in C_i, y \in C_j\} \tag{7}$$

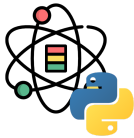
- 平均距离，即两个簇内部样本点之间距离的均值：

$$dist_{avg} = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} dist\,(x,y) \tag{8}$$

- 重心距离，即两个簇重心之间的距离：

$$dist_{med} = dist\left(Median_{C_i}, Median_{C_j}\right) \tag{9}$$

层次聚类

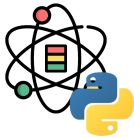


DIANA（Divisive Analysis）算法指一种分裂层次聚类算法，其基本思想如下：

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} 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。

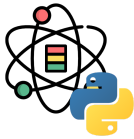
层次聚类



```
sklearn.cluster.AgglomerativeClustering(
    n_clusters=2, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto',
    linkage='ward', distance_threshold=None)
```

参数	描述
n_clusters	聚类个数
affinity	计算连接使用的度量，'euclidean', 'l1', 'l2', 'manhattan', 'cosine', 'precomputed'
connectivity	连接矩阵
compute_full_tree	最大迭代次数
linkage	连接准则，'ward', 'complete', 'average', 'single'
distance_threshold	距离阈值，大于该阈值则不合并

层次聚类



```
import numpy as np

from sklearn.cluster import AgglomerativeClustering

X = np.array([[1, 2], [1, 4], [1, 0],
              [4, 2], [4, 4], [4, 0]])

clustering = AgglomerativeClustering().fit(X)
clustering
```

```
## AgglomerativeClustering()
```

```
clustering.n_clusters_
```

```
## 2
```

```
clustering.labels_
```

```
## array([1, 1, 1, 0, 0, 0])
```

```
clustering.n_leaves_
```

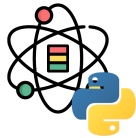
```
## 6
```

```
clustering.n_connected_components_
```

```
## 1
```

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 $D=\{x_1, x_2, \ldots, x_n\}$，引入如下概念和记号：

- ϵ 邻域（ ϵ neighborhood）

$$N_\epsilon(x)=\{y \in X | dist(x, y) \leq \epsilon\} \tag{10}$$

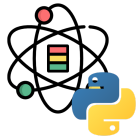
  对于 $x \in D$，称 $N_\epsilon(x)$ 为 x 的 ϵ 邻域。

- 密度（density）

$$\rho(x)=|N_\epsilon(x)| \tag{11}$$

  对于 $x \in D$，称 $\rho(x)$ 为 x 的密度。

基于密度的聚类



- 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

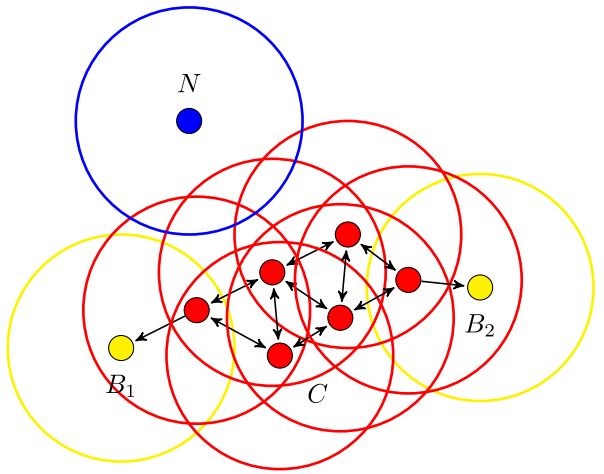
  $$y \in N_{\epsilon}(x) \cap D_{core} \tag{12}$$

  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}。

- 噪音点（noise point）

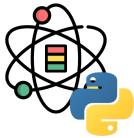
  记 $D_{noise} = D \setminus (D_{core} \cup D_{border})$，对于 $x \in D_{noise}$，则称 x 为噪音点。

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。

基于密度的聚类



- 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 密度直达。

- 密度可达（density-reachable）

  若存在一个序列 $p_1, p_2, \ldots, p_m \in D$，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 $x, y \in D$ 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。

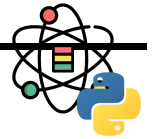
基于密度的聚类

DBSCAN 算法如下：

Algorithm 1 DBSCAN 算法

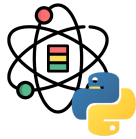
Require: 数据集 D，参数 $(\epsilon, MinPts)$
Ensure: 簇划分 $C = \{C1, C2, \ldots, C_k\}$

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



1: **procedure** DBSCAN($D, \epsilon, MinPts$)
2: #接上文
3: **while** **do**
4: 当前为访问的样本集合： $_{old}$
5: 随机选取一个核心点 $p \in$ ，并初始化一个队列 $\{p\}$
6: $\setminus \{p\}$
7: **while** **do**
8: 的队首
9: **if** $\rho \geq MinPts$ **then**
10: $N_\epsilon() \cap$
11: $\cup$
12: $\setminus$
13: **end if**
14: **end while**
15: k $k+1$
16: 生成簇 C_k $_{old}$ $\setminus$
17: $\setminus C_k$
18: **end while**
19: **return** $C = \{C_1, C_2, \ldots, C_k\}$
20: **end procedure**

基于密度的聚类

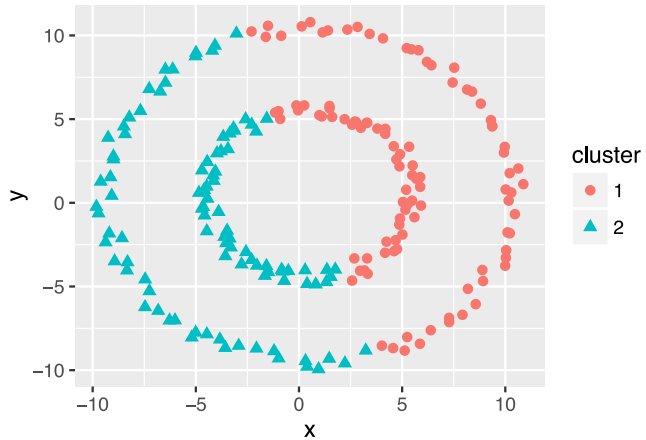


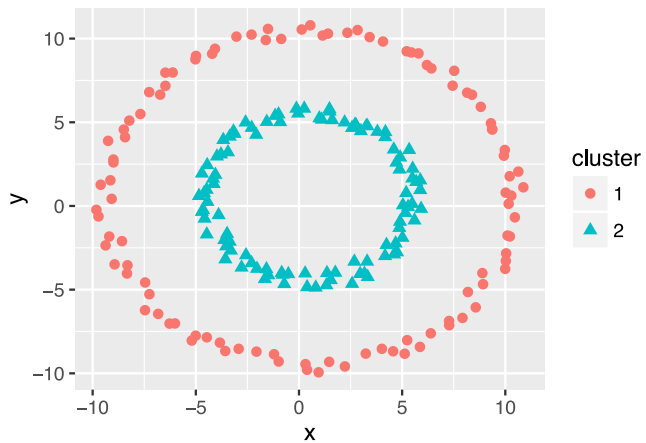
相比 K-means 算法，DBSCAN 算法有如下优势：

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。

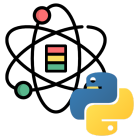
尽管相比 K-means，DBSCAN 算法有很多优势，但是对于不同的数据集，DBSCAN 算法的参数 ϵ 和 $MinPts$ 有时很难选取和优化。

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：





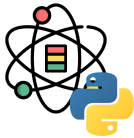
基于密度的聚类



```
sklearn.cluster.DBSCAN(
    eps=0.5, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30,
    p=None, n_jobs=None)
```

参数	描述
eps	用于确定邻域的最小距离
min_samples	用于确定一个点是否为核心点的样本个数
metric	距离度量，sklearn.metrics.pairwise_distances
algorithm	最近邻算法，'auto', 'ball_tree', 'kd_tree', 'brute'
leaf_size	用于 BallTree 或 cKDTree 的叶子个数
p	用于计算 Minkowski 距离的 p 值

基于密度的聚类



```
import numpy as np

from sklearn.cluster import DBSCAN

X = np.array([[1, 2], [2, 2], [2, 3],
              [8, 7], [8, 8], [25, 80]])

clustering = DBSCAN(eps=3, min_samples=2).fit(X)
clustering
```

```
## DBSCAN(eps=3, min_samples=2)
```

```
clustering.core_sample_indices_
```

```
## array([0, 1, 2, 3, 4])
```

```
clustering.labels_
```

```
## array([ 0,  0,  0,  1,  1, -1])
```

感谢倾听