


# 聚 类

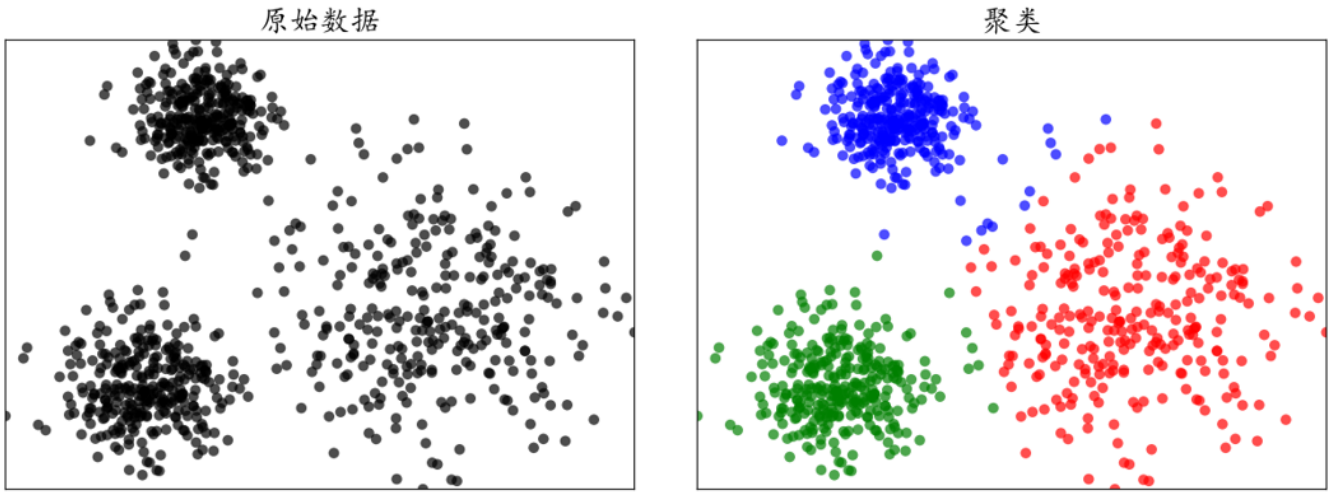
计算机教学部
刘小平
2023.4

# 主要内容

- 聚类
- K均值算法
- Sklearn中的K均值算法示例
- 其他算法*
- 聚类算法性能评价

# 无监督学习之聚类

- 无监督学习
  - 主成分分析、自编码器、最大期望算法、K均值算法、高斯混合模型、密度聚类等
- 聚类
  - 物以类聚



# 什么是聚类

- 聚类算法是一种无监督学习方法
- 聚类算法的主要目的是将数据集中的样本划分为若干个不相交的子集（“簇，cluster”）
- 聚类算法的最直接方法：计算数据点之间的相似性，将相似的数据点划分到同一簇中。
- 聚类算法的应用场景：市场分析、商业经营、图像处理、决策支持、模式识别。

# 初识K均值算法(K—Means)

- 算法基本原理：将$n$个输入样本划分为K个类别，并使得这些类别满足：
  - 1）同一类别中的样本相似度较高；
  - 2）不同类别中的样本相似度较低。

  - 最常见的衡量相似度指标：欧氏距离
  - 根据相似度，划分到同一类的样本子集称为**簇（Cluster）**。

# K均值算法步骤

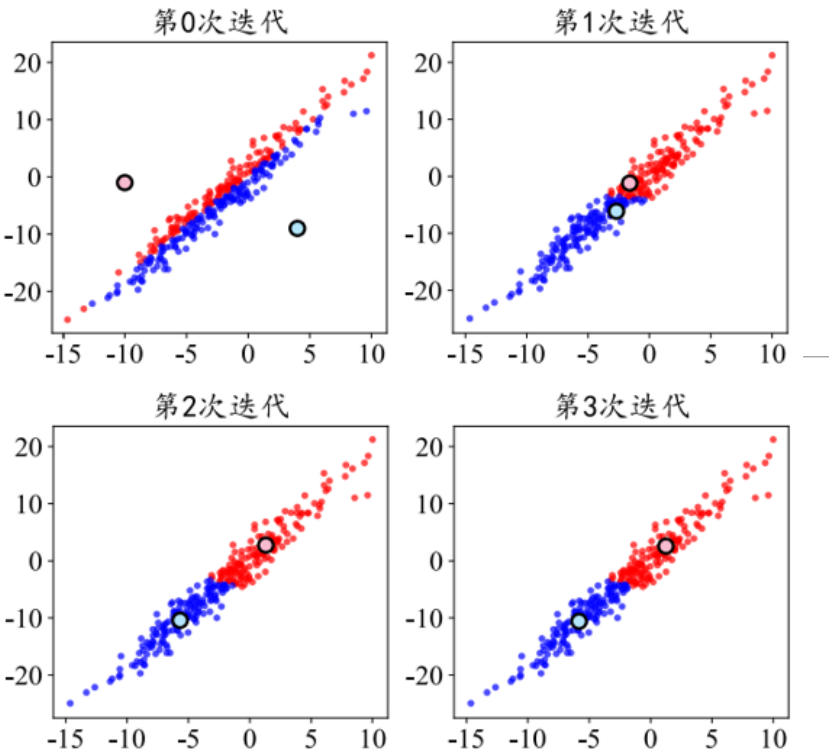
（1）适当选择K个初始的簇中心，设t=0；

（2）在第t次迭代中，对任意一个样本，分别计算其到K个中心的距离，并将该样本归到距离最短的中心所在的簇；

（3）利用均值等方法更新K个簇中心；

（4）判断终止条件，如果满足，算法结束；如果不成立，t:=t+1，返回第（2）步。

**K均值算法的迭代过程：**



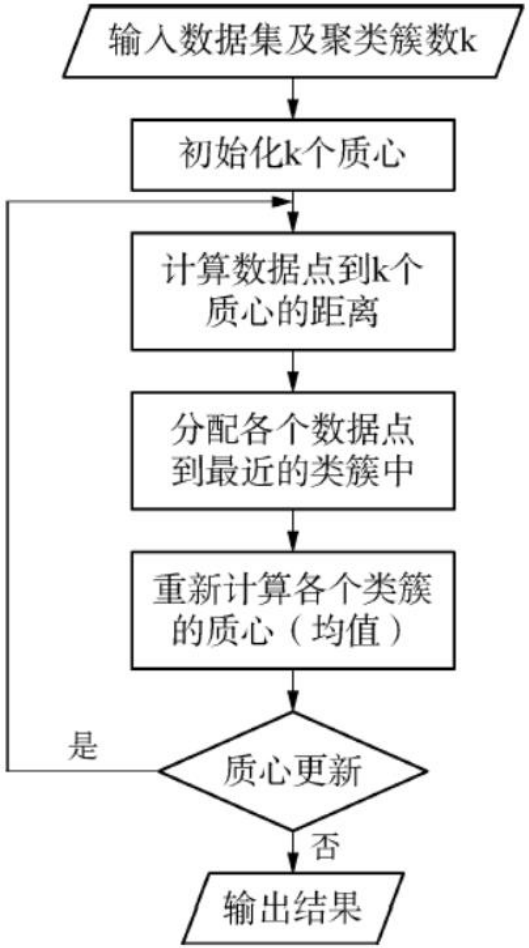




图 5-4-1　K-Means 算法流程图

# K均值算法损失函数

$$J(\boldsymbol{\mu}) = \frac{1}{2}\sum_{j=1}^{K}\sum_{i=1}^{n_j}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_j\right)^2$$

其中，$n_j$表示第$j$个簇中样本的数量，$\boldsymbol{\mu}_j$表示第$j$个簇中心。K是事先设定的，因此该目标函数的变量为$n_j$和$\boldsymbol{\mu}_j$两组。

当$n_j$固定时，对$\boldsymbol{\mu}_j$求导并令导数等于0：

$$\frac{\partial}{\partial \boldsymbol{\mu}_j}J(\boldsymbol{\mu}) = -\sum_{i=1}^{n_j}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_j\right) = 0 \implies \boldsymbol{\mu}_j = \frac{1}{n_j}\sum_{i=1}^{n_j}\boldsymbol{x}_i$$

K均值算法的终止条件，一般来说，有以下几种：1）目标函数的值的变化小于某个阈值；2）簇中心在一定范围内不再变化；3）达到指定的迭代次数。

由于算法基于欧氏距离，均值和方差大的维度对聚类的结果产生更大的影响，所以必须要对数据进行归一化或统一单位等预处理。

# K均值算法执行过程举例

表 5-4-1　聚类示例数据集

| 编号 | 属性 1 | 属性 2 |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 1 |
| 4 | 6 | 4 |
| 5 | 7 | 5 |
| 6 | 8 | 4 |

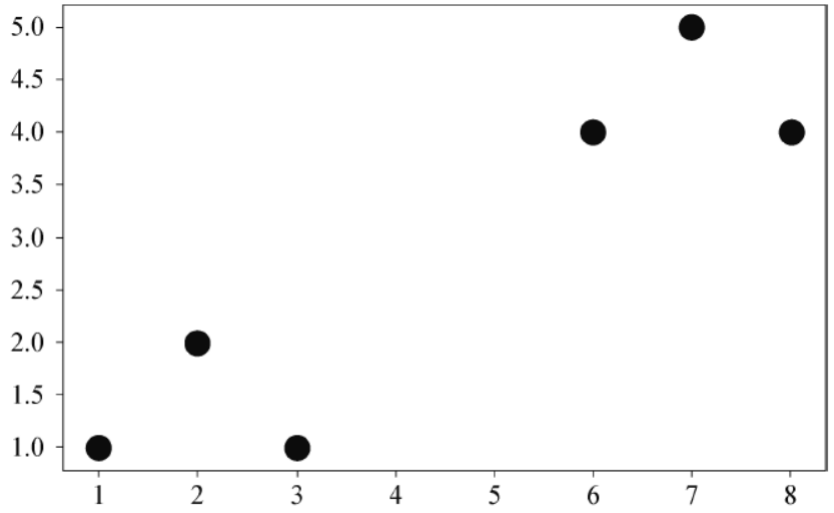



图 5-4-2　聚类示例散点图

# K均值算法执行过程举例

- 选取聚类簇数为2，选定$x_1, x_2$为初始质心。

表 5-4-2　第一次样本聚类

| 样本编号 | 质心(1,1)的距离 | 质心(2,2)的距离 | 目标簇号 |
|---|---|---|---|
| 1:(1,1) | 0 | $\sqrt{2}$ | 1 |
| 2:(2,2) | $\sqrt{2}$ | 0 | 2 |
| 3:(3,1) | 2 | $\sqrt{2}$ | 2 |
| 4:(6,4) | $\sqrt{34}$ | $\sqrt{24}$ | 2 |
| 5:(7,5) | $\sqrt{52}$ | $\sqrt{34}$ | 2 |
| 6:(8,4) | $\sqrt{58}$ | $\sqrt{40}$ | 2 |

计算新的质心：

$$u_1=[1,1]$$

$$u_2=[(2+3+6+7+8)/5,(2+1+4+5+4)/5]=[5.2,3.2]$$

# K均值算法执行过程举例

表 5-4-3　第一次样本聚类结果

| | |
|---|---|
| 原来质心集合 | [[1,1],[2,2]] |
| 聚类结果 | $\{\{x_1\},\{x_2,x_3,x_4,x_5,x_6\}\}$ |
| 新质心集合 | [[1,1],[5.2,3.2]] |

## 由于质心发生了更新进入第二次样本聚类

表 5-4-4　第二次样本聚类

| 样本编号 | 质心(1,1)的距离 | 质心(5.2,3.2)的距离 | 目标簇号 |
|---|---|---|---|
| 1:(1,1) | 0 | 4.74 | 1 |
| 2:(2,2) | 1.41 | 3.42 | 1 |
| 3:(3,1) | 2 | 3.11 | 1 |
| 4:(6,4) | 5.83 | 1.13 | 2 |
| 5:(7,5) | 7.21 | 2.55 | 2 |
| 6:(8,4) | 7.62 | 2.91 | 2 |

计算新的质心：

$$u_1=[(1+2+3)/3,(1+2+1)/3]=[2,1.33]$$

$$u_2=[(6+7+8)/3,(4+5+4)/3]=[7,4.33]$$

# K均值算法执行过程举例

表 5-4-5　第二次样本聚类结果

| | |
|---|---|
| 原来质心集合 | [[1,1],[5.2,3.2]] |
| 聚类结果 | $\{\{x_1,x_2,x_3\},\{x_4,x_5,x_6\}\}$ |
| 新质心集合 | [[2,1.33],[7,4.33]] |

## 由于质心发生了更新进入第三次样本聚类

表 5-4-6　第三次样本聚类

| 样本编号 | 质心(2,1.33)的距离 | 质心(7,4.33)的距离 | 目标簇号 |
|---|---|---|---|
| 1:(1,1) | 1.05 | 6.86 | 1 |
| 2:(2,2) | 0.67 | 5.52 | 1 |
| 3:(3,1) | 1.05 | 5.21 | 1 |
| 4:(6,4) | 4.81 | 1.05 | 2 |
| 5:(7,5) | 6.20 | 0.67 | 2 |
| 6:(8,4) | 6.57 | 1.05 | 2 |

计算新的质心：

$$u_1=[(1+2+3)/3,(1+2+1)/3]=[2,1.33]$$

$$u_2=[(6+7+8)/3,(4+5+4)/3]=[7,4.33]$$

# K均值算法执行过程举例

表 5-4-7　第三次样本聚类结果

| | |
|---|---|
| 原来质心集合 | [[2,1.33],[7,4.33]] |
| 聚类结果 | $\{\{x_1,x_2,x_3\},\{x_4,x_5,x_6\}\}$ |
| 新质心集合 | [[2,1.33],[7,4.33]] |

**由于质心集合未发生更新，所以算法结束。**

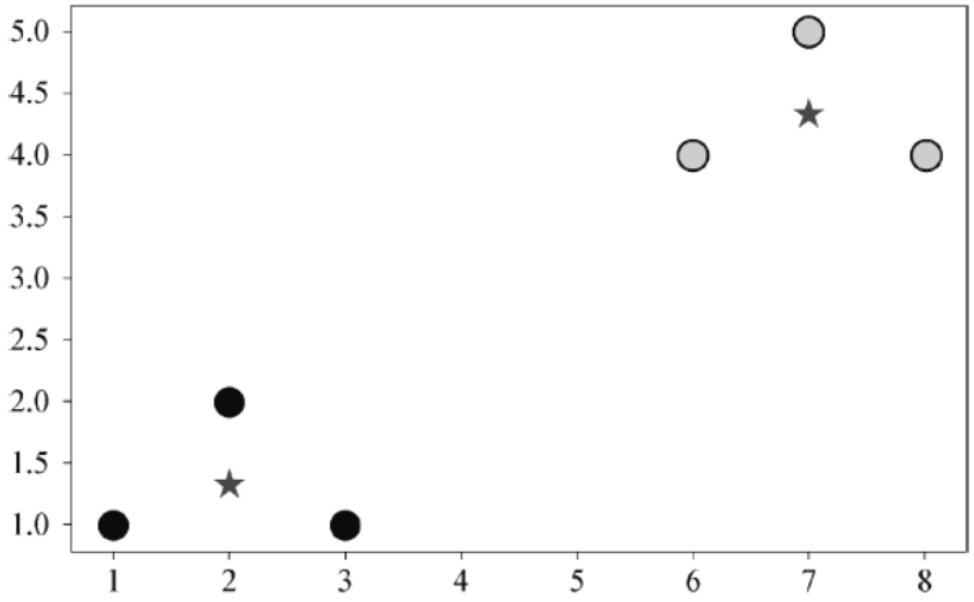
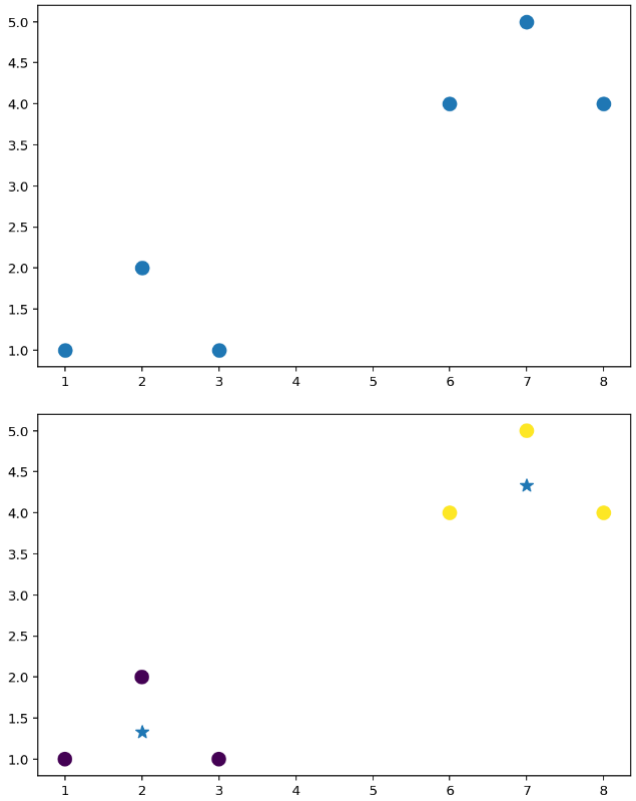



图 5-4-3　聚类后的散点图

# K—Means聚类示例1

```
# （1）导入库
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

# （2）生成样本数据
samples = np.array([[1,1],[2,2],[3,1],[6,4],[7,5],[8,4]])

# （3）把样本数据显示在二维坐标上
plt.figure(figsize=(8,5),dpi=144)
plt.scatter(samples[:,0],samples[:,1],s=100)

# （4）使用KMeans模型拟合
est=KMeans(n_clusters=2)
est.fit(samples)

# （5）将聚类结果利用散点图显示出来
labels=est.labels_
centers=est.cluster_centers_
fig=plt.figure(figsize=(8,5),dpi=144)
plt.scatter(samples[:,0],samples[:,1],s=100,c=labels.astype(np.float))
plt.scatter(centers[:,0],centers[:,1],s=100,marker="*")
plt.show()
```

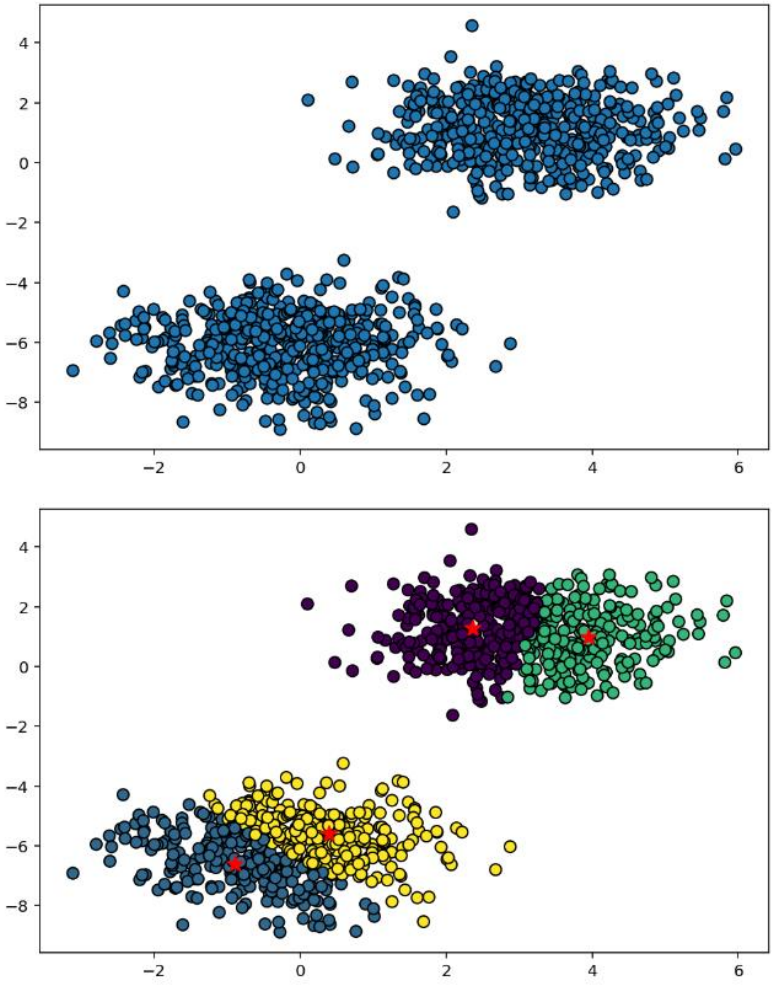


# Sklearn中的K-Means实现

- 构造函数：KMeans(n_clusters=8,…)
  - 基本不需要调参，一般来说，只需要指定聚类簇的数量。
- K-Means类主要属性
  - cluster_centers_：所有质心
  - labels_：每一个数据点的标签值
- K-Means类的主要方法：
  - fit(X[,y,sample_weight)：用来计算数据样本X的K-Means聚类簇
  - fit_predict(X[,y,sample_weight])：返回每个数据对应的标签，并将标签值对应到相应的簇。

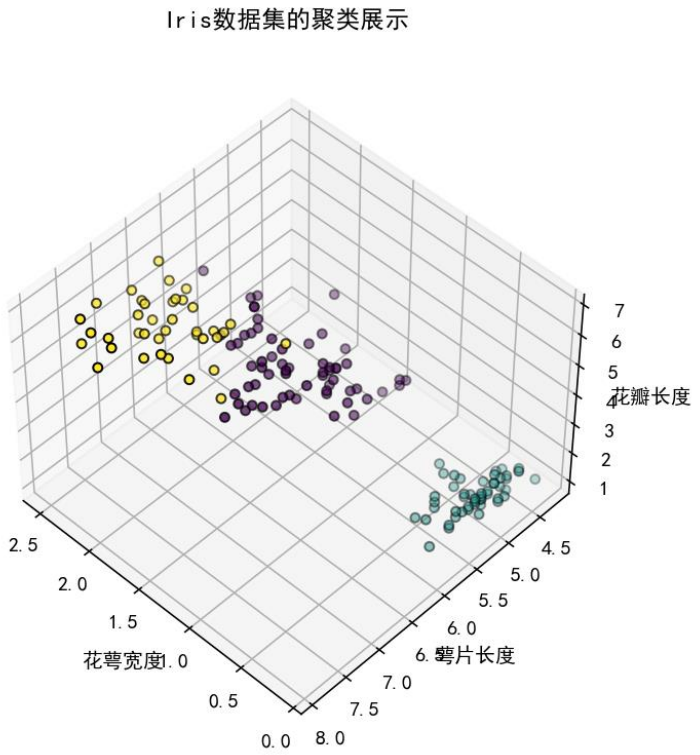
# K—Means聚类示例2

```
#（1）导入库
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

#（2）利用scikit—learn中的make_blobs函数生成样本数据集
X,Y=make_blobs(n_samples=1000,centers=2)

#（3）利用散点图的形式将样本数据展示出来
plt.figure(figsize=(8,5),dpi=144)
plt.scatter(X[:,0],X[:,1],s=50,edgecolor='k')
plt.show()

#（4）使用KMeans模型拟合，聚类数设为4。
kmean=KMeans(4)
kmean.fit(X)

#（5）将聚类结果利用散点图显示出来
labels=kmean.labels_
centers=kmean.cluster_centers_
fig=plt.figure(figsize=(8,5),dpi=144)
plt.scatter(X[:,0],X[:,1],c=labels.astype(int),s=50,edgecolor='k')#显示聚类结果
plt.scatter(centers[:,0],centers[:,1],c='r',s=100,marker="*")#显示质心
plt.show()
```

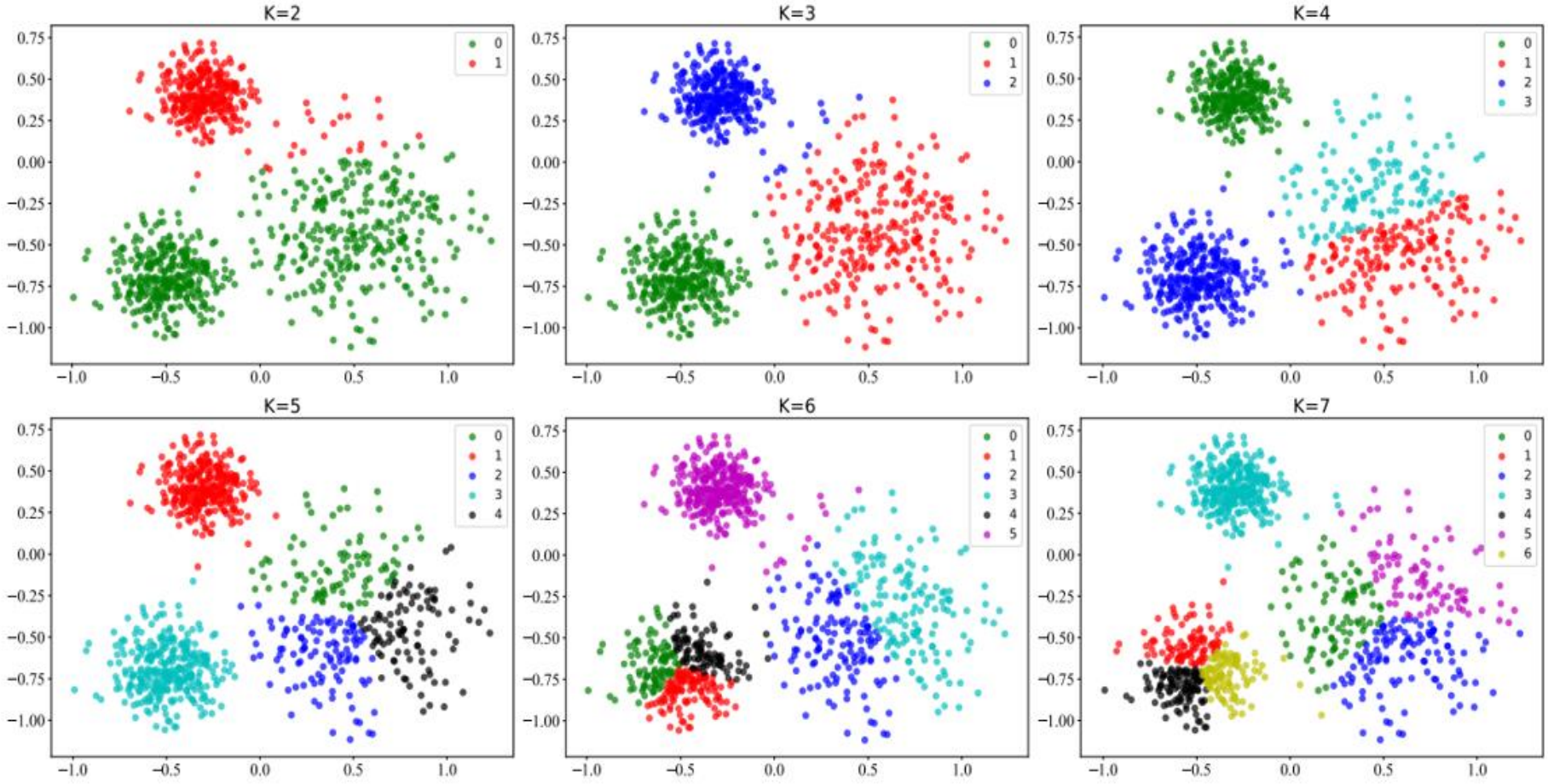


# K—Means聚类示例3

```
#（1）导入库
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn import datasets
import numpy as np
plt.rcParams['font.sans-serif']=['SimHei'] #避免中文出现乱码
#（2）导入Iris数据
iris=datasets.load_iris()#导入iris数据
X=iris.data
#（3）使用KMeans模型拟合，聚类数设为3
est=KMeans(n_clusters=3)
est.fit(X)
#（4）选取其中的三个维度，并显示其聚类结果
labels=est.labels_
fig=plt.figure(figsize=(8,5),dpi=144)
ax=Axes3D(fig,elev=48,azim=134)
ax.scatter(X[:,3],X[:,0],X[:,2],c=labels.astype(np.float),edgecolor='k')
ax.set_xlabel('花萼宽度')
ax.set_ylabel('萼片长度')
ax.set_zlabel('花瓣长度')
ax.set_title('Iris数据集的聚类展示')
ax.dist=12
plt.show()
```

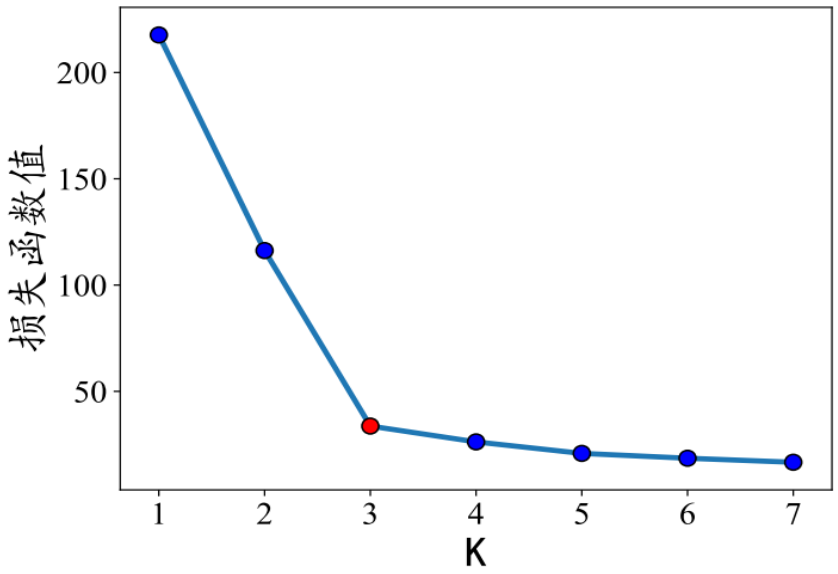


# K均值算法实验



# K均值算法存在的问题1

- 不同的K可能会得到非常不同的结果，如何选择适当的K值？
  - 手肘法

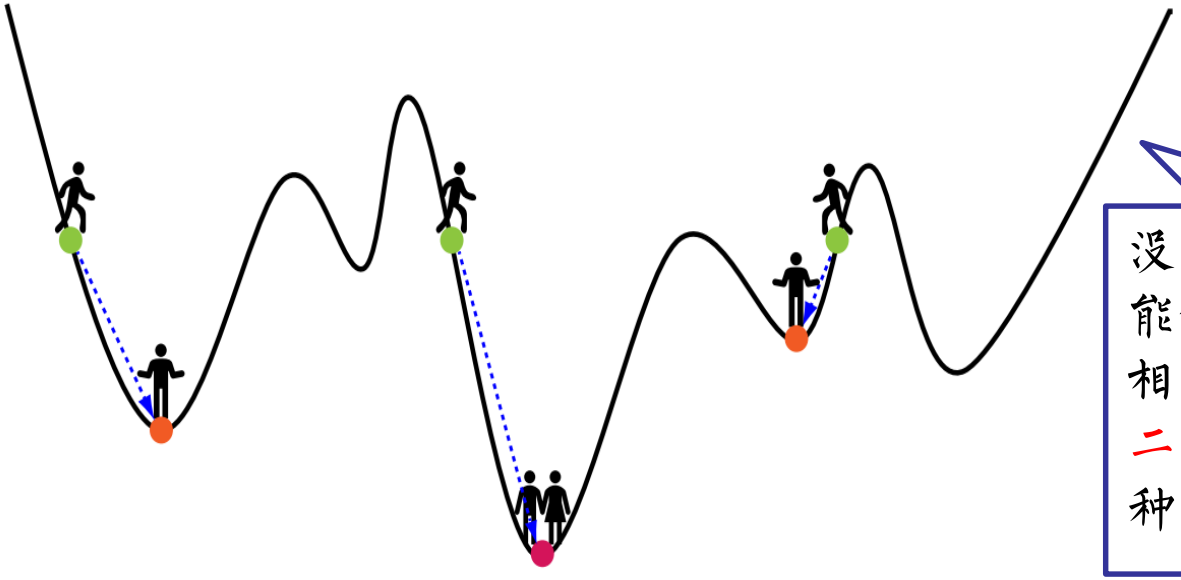



手肘图

  - 这条趋势线非常像人的一条手臂，且手肘位置的K值恰好位于真实簇的个数附近，例如本例中正好等于3（见图中红点）。通过该方法画出手肘图，将肘部对应的K值作为最佳选择，在大部分时候都是非常有效的。

# K均值算法存在的问题2

- 目标函数不是凸函数，容易陷入局部极小值。



非凸函数

没有办法破解此局，所能做的是使之尽量落入相对较低的局部最小值。二分K均值算法便是一种缓解之策。

- 二分K均值算法主要思想是：先将所有点当做一个簇，且将该簇一分为二。然后选择可以最大程度降低聚类损失函数的簇划分为两个簇。按此法继续下去，直到簇的数目等于设定的K值为止。该算法提升了速度，且实验显示能够找到相对较优的局部最小值。

# K均值算法存在的问题3

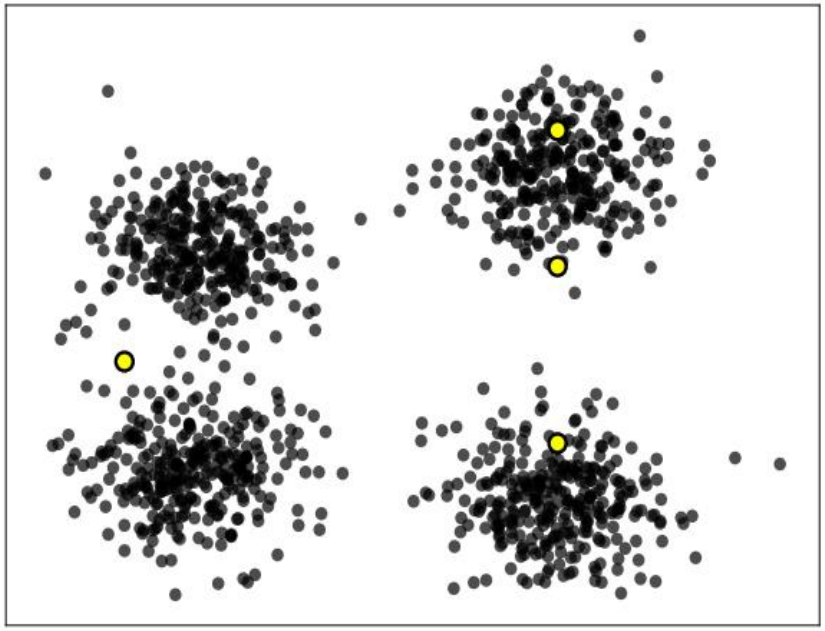
- 样本中含有异常点，容易导致簇中心严重偏离。

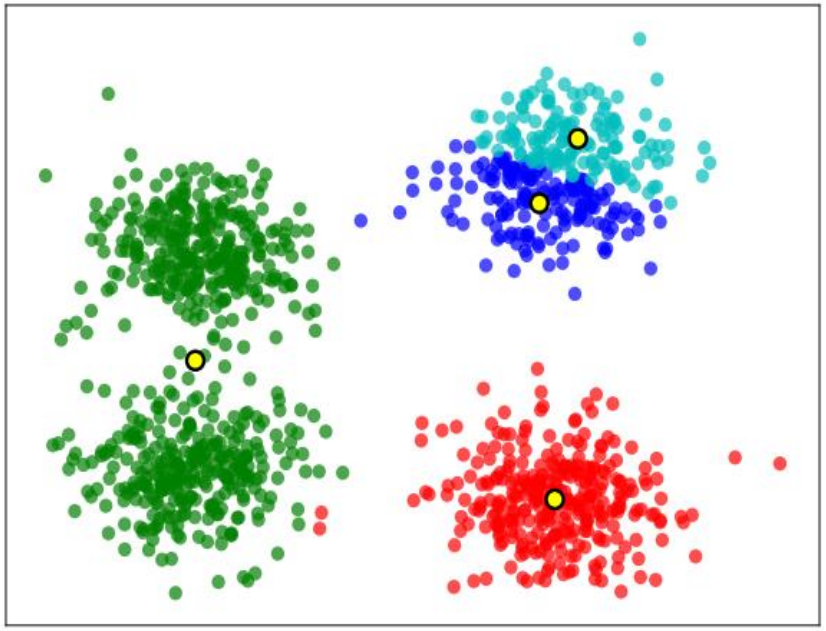

均值和中值

- K中值（K—Mediods）算法
- 它与K均值算法大体相似，只是在更新簇中心时，先在同一簇内计算每一个点到其它点的距离之和，然后再选择距离最小的点作为新的簇中心。
- 在上面的例子中，如果采用K中值算法，可以分别计算得到每个点作为簇中心时的各点距离之和为：105，102，101，102，390。因此，簇中心将位于3，这样就避免了样本中的异常值带来的影响.

# K均值算法存在的问题4

- K均值算法对初始中心非常敏感。





  - K—Means++算法
  - 在选择初始中心时，不是纯粹地随机选择，而是加入一定的策略，即让初始中心之间的相互距离尽可能的远。
  - 自己动手：https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

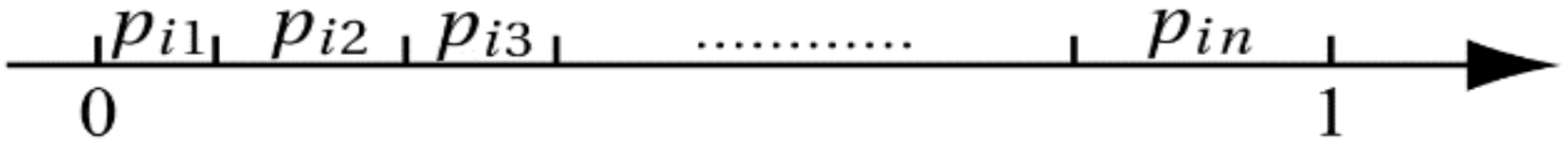
# K—Means++算法1（具体参考实验）

- 与原始算法的区别仅在初始中心的选择。
- K—Means++算法初始中心选择的一般步骤如下：
  - （1）从给定的样本中随机选择一个点作为第一个簇中心，设k=1；如果k=K，算法结束；否则，继续下一步。
  - （2）对于每个样本点，计算它与最近的已选中簇中心的距离；
  - （3）选择一个新的簇中心，原则是**距离越远的点被选中的概率越大**；
  - （4）返回第（2）步。

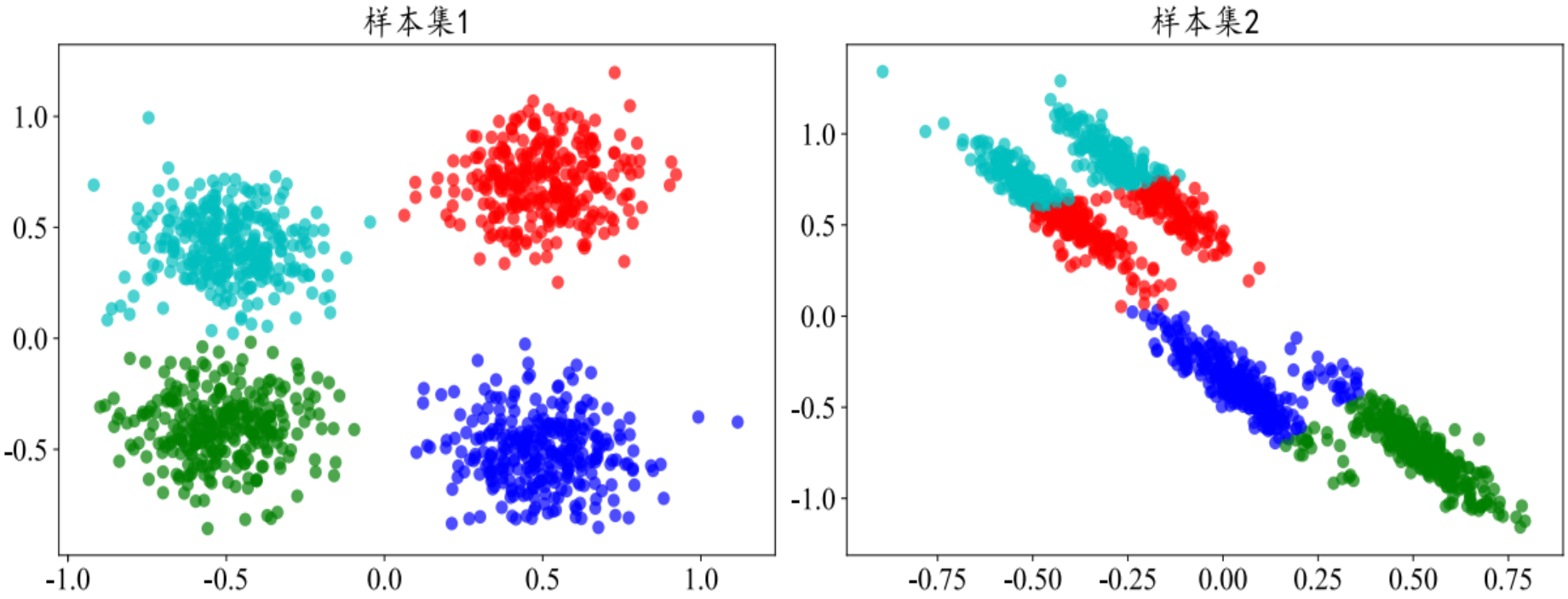
假设当前只有一个簇中心$\boldsymbol{x}_i$（当前有多个簇中心时，做法基本相同），那么每个点距离该中心的距离可以表示为$d_{ij}$，其中$j = 1,2,\cdots,n$。这样，$\boldsymbol{x}_j$被选中的概率可以表示为：

$$p_{ij} = \frac{d_{ij}}{\sum_{j=1}^{n} d_{ij}}$$

具体实现：所有概率之和为1，将它们摆放在0至1的数轴上，然后取一个0至1的随机数。该随机数落入哪个区域就选取哪个样本作为新的簇中心。注意，$p_{ii} = 0$，因此是不可能重复选到同一个样本的。



# K—Means++算法2（具体参考实验）




K-Means++的实验结果

K均值算法对于右图这样的样本分布显得束手无策。

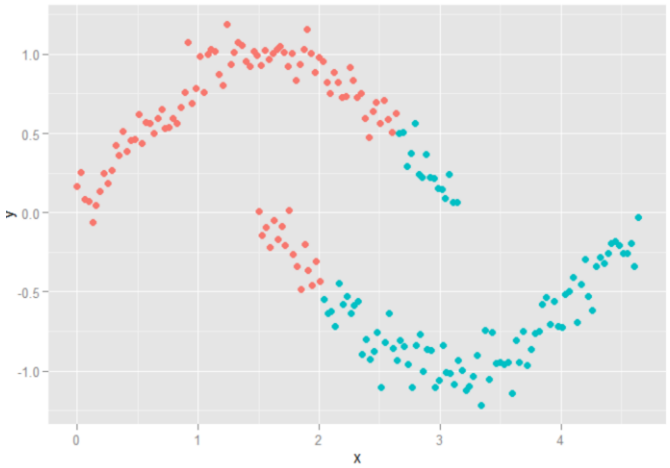
# K均值算法存在的问题5

- 对非球状的样本分布，K均值算法显得束手无策。
  - 原因：采用均方误差作为损失函数，本质上假设了样本集的每个簇分别符合某种高斯分布，是呈现圆形、球形或高维球形的。
  - 解决办法：
    - 核函数，将输入空间的样本非线性映射到高维的特征空间，使得样本点在特征空间中线性可分的概率增加。
    - 谱聚类
    - 基于密度的算法

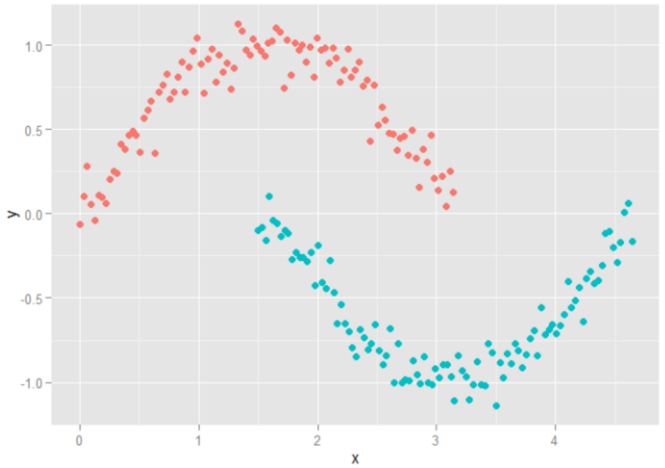
注意：上面讨论的各问题的应对策略只是比较常见和典型的方法。其实，针对每一个问题，都有很多的文献提出了有效方法。读者应以这些问题为出发点，去探索和学习其它的聚类算法。

# 密度聚类

- K均值算法这种基于中心点划分的聚类方法在面对非球状样本分布时容易失效；而密度聚类最大的优点恰好是可以对空间中任意形状的样本分布进行聚类。




K-Means算法聚类效果




密度聚类效果

- 密度聚类：基于某种定义的密度概念来进行聚类，将紧密相连的样本划为一类。经典算法有DBSCAN算法 、OPTICS算法 、DENCLUE算法以及DPCA算法等等。

# DBSCAN算法*（自学）

Density-Based Spatial Clustering of Applications with Noise

https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

- 基本思路：若样本点的密度大于预设的阈值，则将该样本添加到最近的簇中。
  - 簇定义为密度相连的样本的最大集合，它可以发现任意形状的簇。
  - 位于簇边缘附近或远离簇的样本由于密度过低，将被视为噪声，因此算法具有抗噪声的能力。

- 相关概念：

  $\boldsymbol{D}$代表所有样本或对象的集合

  $\boldsymbol{\varepsilon}$表示某对象的邻域距离的阈值

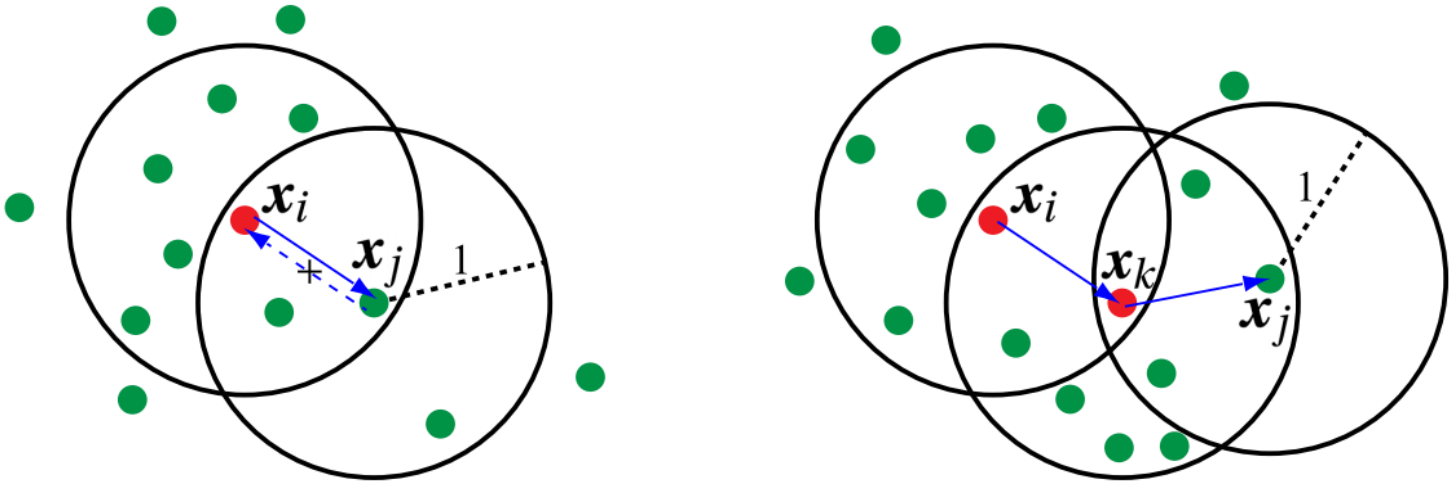
  $\boldsymbol{n_s}$表示某对象的邻域中包含对象个数的阈值

  **$\varepsilon$-邻域：**以集合$D$中的某个对象为中心，半径为$\varepsilon$的区域被称为该对象的$\varepsilon$-邻域

  **核心对象：**如果某个对象的$\varepsilon$-邻域内至少包含$n_s$个对象，则称该对象为核心对象

# DBSCAN算法*（自学）

Density—Based Spatial Clustering of Applications with Noise



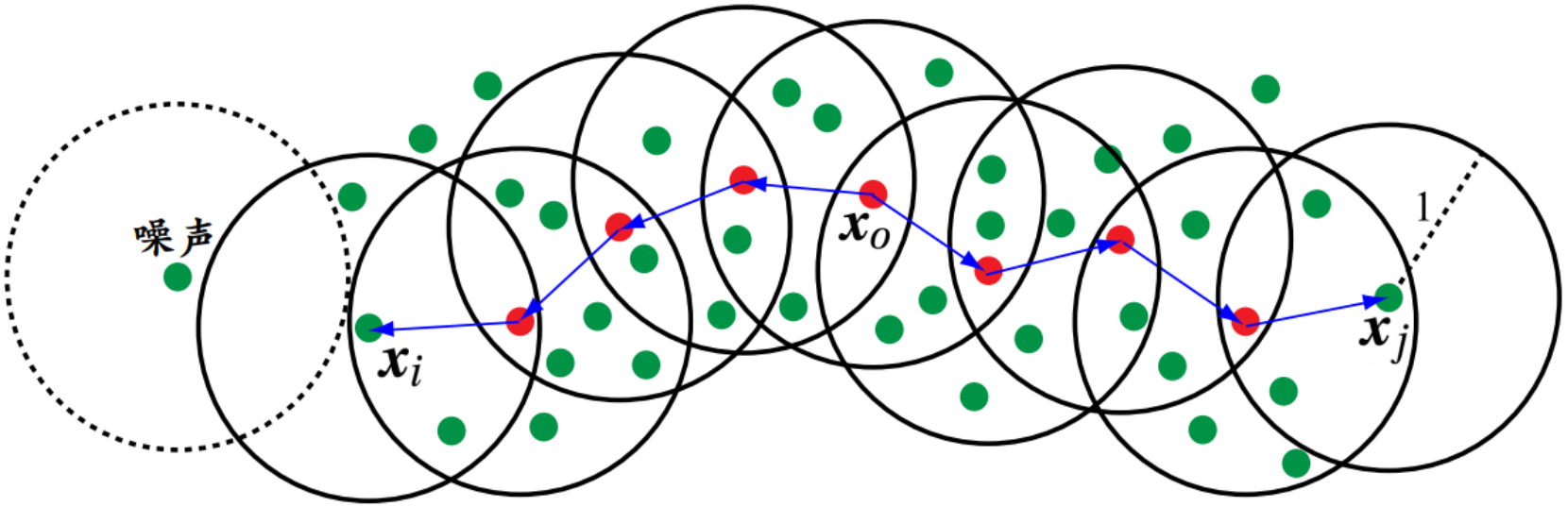

密度直达和密度可达

**密度直达：**如果$x_j$在$x_i$的$\varepsilon$－邻域内，并且$x_i$是核心对象，那么说从$x_i$出发到$x_j$是关于$\varepsilon$和$n_s$密度直达的。如上左图所示，假设$\varepsilon = 1$，$n_s = 5$。那么$x_i$是核心对象，而$x_j$不是核心对象，因此，$x_i$到$x_j$是密度直达的，而$x_j$到$x_i$是密度不直达的。

**密度可达：**在集合$D$中，如果存在一个对象序列$x_1, x_2, \cdots, x_k, x_{k+1}, \cdots, x_l$，且$x_1 = x_i$，$x_l = x_j$。如果对象序列中的前后对象都满足$x_k$到$x_{k+1}$是关于$\varepsilon$和$n_s$密度直达的，那么就称$x_i$到$x_j$是密度可达的。如上右图所示，可以发现，在构成密度可达的这个序列中，除了$x_j$，其它对象都必须是核心对象。

# DBSCAN算法*（自学）

Density—Based Spatial Clustering of Applications with Noise




密度相连和噪声

**密度相连：**如果$x_o$到$x_i$密度可达，且$x_o$到$x_j$密度可达，那么$x_i$与$x_j$是密度相连的。如图5-50所示，可以发现密度相连关系满足对称性，即如果$x_i$与$x_j$密度相连，那么$x_j$与$x_i$也密度相连。

**簇：**最大的密度相连对象的集合。通过不断搜索核心对象的可密度直达对象，所形成的集合。在簇内，可以有一个或者多个核心对象。如果只有一个，那么其它所有非核心对象必然在该核心对象的$\varepsilon$—邻域。如果有多个，那么簇内任意核心对象的$\varepsilon$—邻域中必然有一个其它的核心对象。

**噪声：**不包含在任何簇中的对象称为噪声。

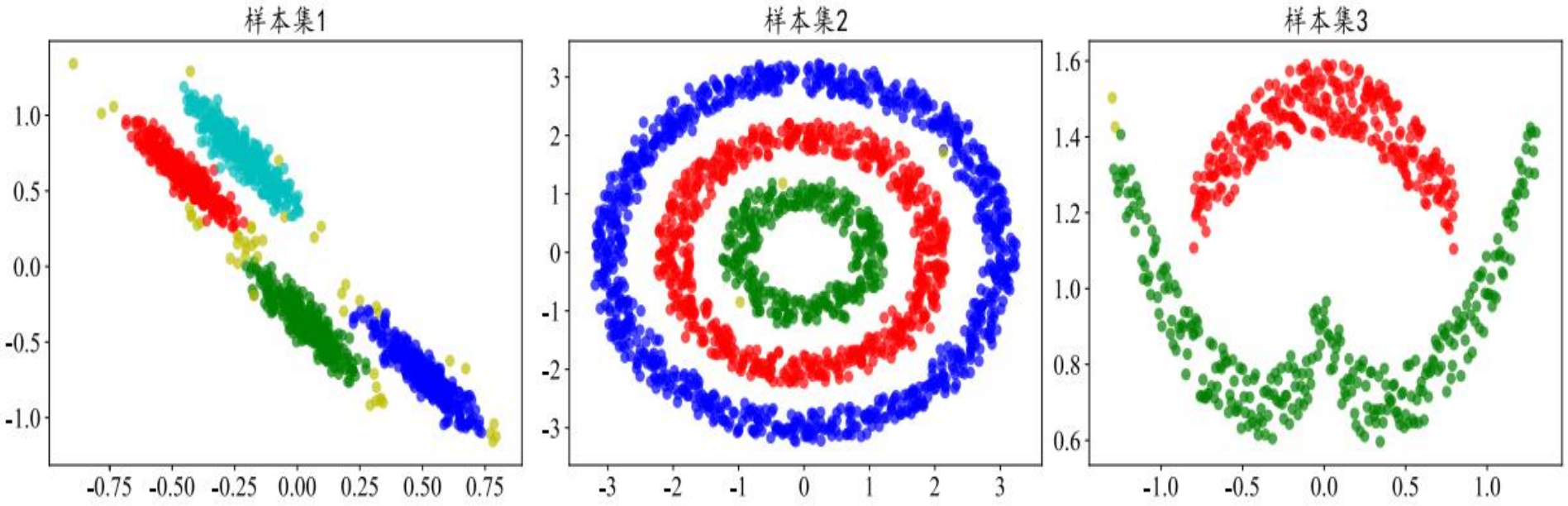
# DBSCAN算法*（自学）

Density—Based Spatial Clustering of Applications with Noise

- 如何找到簇？可简单描述如下：
  - 首先，在没有被归类的核心对象中任意选择一个作为种子；
  - 然后，搜索这个核心对象能够密度可达的所有样本，即找到一个簇。
  - 重复以上操作，直到所有核心对象都被归类。
- DBSCAN算法的优点在于：
  - 1）可以对任意形状的样本聚类；
  - 2）对异常点不敏感，并能发现异常点。
  - 3）初始值对于结果的影响相对不大。
- 缺点：
  - 1）对于分布不是很稠密的样本，聚类效果不好；
  - 2）需要设置两个参数$\varepsilon$和$n_s$，参数直接影响到最终的聚类效果。

# DBSCAN算法实验*（自学）

Density—Based Spatial Clustering of Applications with Noise




DBSCAN的实验结果

# 聚类算法的性能评价

- 常见的评价指标有混淆矩阵、均一性、完整性、V—measure、杰卡德相似系数、皮尔逊相关系数、调整兰德指数、调整互信息、Davies—Boulding指数、Dunn 指数、闵可夫斯基距离、KL散度、余弦相似度、轮廓系数、高斯相似度等等。

- 除Boulding指数和Dunn 指数，其它指标都是需要标注信息的。

- 像Boulding指数和Dunn 指数这种指标，往往存在明显的局限性，比如：对环状分布的样本效果很差。

# 杰卡德相似系数

- 用来度量两个集合之间的相似性：

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

  - 取值在[0,1]之间，越接近1，说明相似度越高。

- 杰卡德距离公式：

$$d_J(A,B)=1-J(A,B)=\frac{|A\cup B|-|A\cap B|}{|A\cup B|}$$

  - 杰卡德距离越大，相似度越低。

假设：标签向量$\boldsymbol{y}$和模型预测得到的向量$\hat{\boldsymbol{y}}$的元素取值都为0或1，那么可以得到四个值：$\boldsymbol{y}$和$\hat{\boldsymbol{y}}$的值都为0的元素个数$n_{00}$，$\boldsymbol{y}$的值为0而$\hat{\boldsymbol{y}}$的值为1的元素个数$n_{01}$，$\boldsymbol{y}$的值为1而$\hat{\boldsymbol{y}}$的值为0的元素个数$n_{10}$，$\boldsymbol{y}$和$\hat{\boldsymbol{y}}$的值都为1的元素个数$n_{11}$。那么：

- 预测值和真实值间的杰卡德相似系数：

$$J(\boldsymbol{y},\hat{\boldsymbol{y}})=\frac{n_{11}}{n_{01}+n_{10}+n_{11}}$$

- 杰卡德距离为：

$$d_J(\boldsymbol{y},\hat{\boldsymbol{y}})=\frac{n_{01}+n_{10}}{n_{01}+n_{10}+n_{11}}$$

# 调整兰德指数

- 兰德指数：

$$RI = \frac{TP + TN}{TP + FP + TN + FN}$$

| | | 真实值 | |
|---|---|---|---|
| | | 正 | 负 |
| 预测值 | 正 | *TP* | *FP* |
| | 负 | *FN* | *TN* |

混淆矩阵

  - TP（True Positive）：指真实值是正的，预测值也是正的样本的个数。
  - FN（False Negative）：指真实值是正的，预测值是负的样本的个数。
  - FP（False Positive）：指真实值是负的，预测值是正的样本的个数。
  - TN（True Negative）：指真实值是负的，预测值也是负的样本的个数。
  - 兰德指数值位于[0,1]之间，如果聚类结果完美，兰德指数为1。

- 调整兰德指数：

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

  - 调整兰德指数的取值在[−1,1]之间，值越大表示预测值和真实值的相似度越高。

# 轮廓系数

样本$\boldsymbol{x}_i$的轮廓系数可定义为：

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

$\boldsymbol{a_i}$为样本$\boldsymbol{x}_i$的簇内不相似度，样本$\boldsymbol{x}_i$到同簇中其它样本的平均距离。

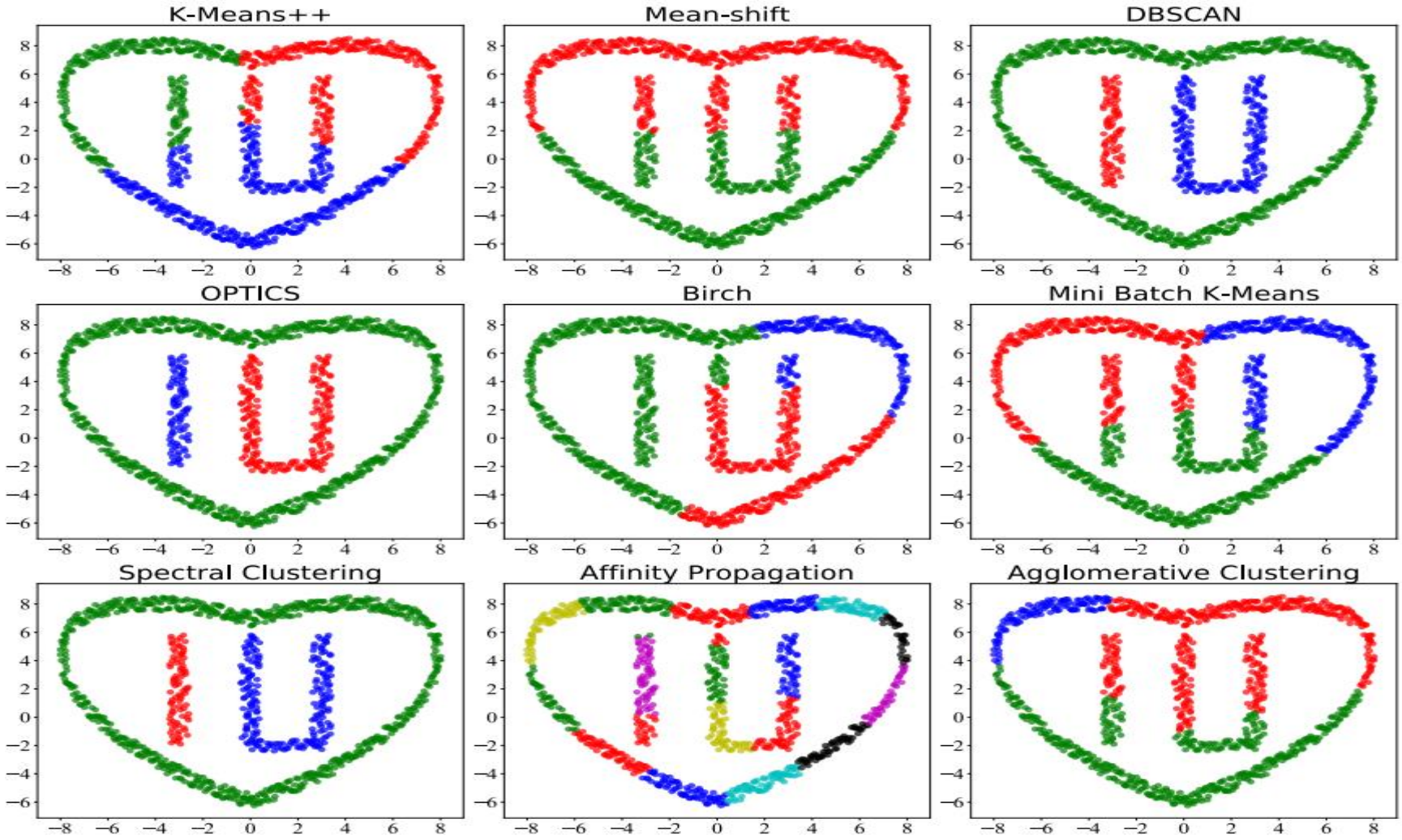
$\boldsymbol{b_i}$为样本$\boldsymbol{x}_i$的簇间不相似度为：$b_i = \min\{b_{i1}, b_{i2}, \cdots, b_{iK}\}$。其中K表示其它簇的数目，$b_{ij}$为样本$\boldsymbol{x}_i$到其它某簇$\Lambda_j$的所有样本的平均距离，如果$b_i$越大，说明样本$\boldsymbol{x}_i$越不属于其它簇。

考虑到$a_i$和$b_i$的小大关系，因此$S_i$可正可负，可以分为如下三种情况：

$$S_i = \begin{cases} 1 - \dfrac{a_i}{b_i}, & a_i < b_i \\ 0, & a_i = b_i \\ \dfrac{b_i}{a_i} - 1, & a_i > b_i \end{cases}$$

$S_i$接近1，说明样本$\boldsymbol{x}_i$归类合理；$S_i$接近$-1$，说明样本$\boldsymbol{x}_i$更应该分类到其它簇；若$S_i$近似为0，说明样本$\boldsymbol{x}_i$在两个簇的边界上。$S_i$值越大说明聚类效果越好。

# 各种算法在“爱心”样本集上的聚类实验




各种聚类算法没有绝对好坏之分，只是在不同场合下，谁表现更加优秀而已。

谢　谢！