机器学习
Machine Learning

鲍军鹏

2024年3月（V1.5）

西安交通大学计算机学院
Email: baojp@xjtu.edu.cn

第四章 聚类方法

- ★ 什么是聚类
- ★ 层次聚类方法
- ★ 划分聚类方法
- ★ 基于密度的聚类方法
- ★ 基于子空间的聚类方法
- ★ 人工神经网络聚类方法

什么是聚类

★聚类（Clustering）

- 人以类聚，物以群分
- 把观察数据S映射到未知有限离散空间X上

$$L: S \rightarrow X,\ X=\{x_1, x_2, \ldots, x_k\}$$

- 即，把每个数据个体划分给一个簇（Cluster）

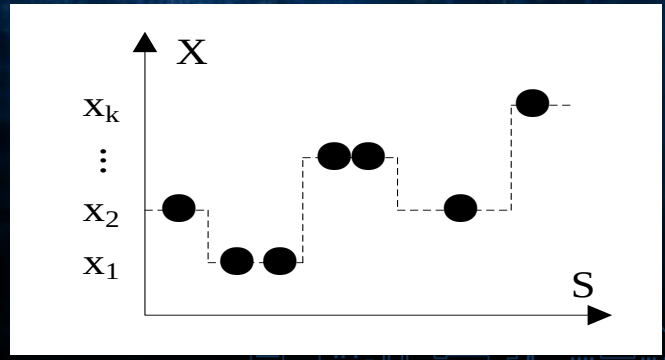
$$L(d)=x_i,\ d \in S,\ x_i \in X$$

要求：簇内相似而簇间不相似

- 聚类器L就是我们要学习的目标

$$\langle D \rangle \Rightarrow L,\ D \subset S$$

- 聚类算法都是无监督学习



什么是聚类

★形式化定义

- 对于观测空间S上的数据集D（D⊂S），
- 求D上的一个划分 $X = \{x_i \mid x_i \subset D, \bigcup_i x_i = D, i = 1,2,...,n\}$
- 使得D中的任意一对数据满足：

$$sim(d_k, d_l) > sim(d_p, d_q)$$

$$d_k, d_l \in x, k \neq l, \quad d_p \in y, d_q \in z, p \neq q, y \neq z, \quad x, y, z \subset D$$

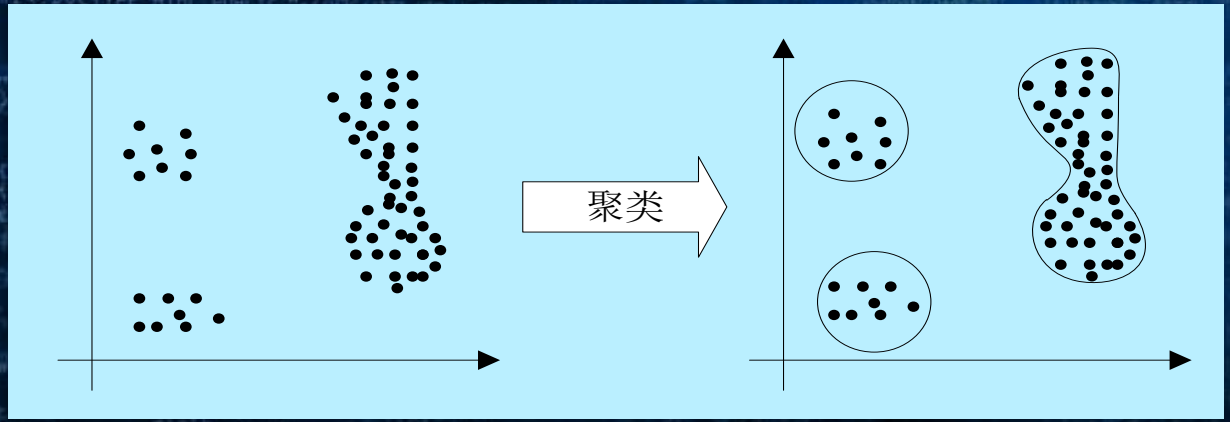
其中，若 $x_i \cap x_j = \Phi, \quad x_i, x_j \subset D$

- 即任意两个簇之间没有共享数据点，亦即一个数据点只能属于一个簇，则称之为硬聚类。
- 否则称之为软聚类。

什么是聚类

★一般而言

- 我们要求同簇内数据的相似度尽可能大，不同簇间数据的相似度尽可能小。



什么是聚类

★数据挖掘对聚类算法的典型要求有

- 可伸缩性：算法能够处理海量的数据对象
- 处理不同类型属性的能力
- 发现具有任意形状的聚类的能力
- 输入参数对领域知识的弱依赖性
- 处理噪声数据或离群数据的能力
- 结果对于输入记录顺序的无关性
- 处理高维度数据的能力
- 结果的可解释性和可用性
- 基于约束的聚类分析能力

什么是聚类

★聚类算法的关键核心

- 度量数据相似性

★相似度

- 两个不同数据点（对象）之间的相似度可以根据问题需要进行不同定义。
- 按照距离定义
  - 距离越大则相似度越小，距离越小则相似度越大
- 按照密度定义
  - 密度越大，则相似度越大，密度越小则相似度越小
- 按照概念定义
  - 具有相同（或者相近）概念的数据（对象）相似度大，反之则小。

常用距离公式

$$d(x,y)=\left(\sum_{k=1}^{m}|x_k-y_k|^q\right)^{\frac{1}{q}},\quad q\geq 1$$

- ★ 曼哈顿距离（ Manhattan Distance ）
  - q=1
- ★ 欧几里德距离（Euclidean Distance）
  - q=2
- ★ 明科夫斯基距离（Minkowski Distance）
  - q>2

第四章 聚类方法

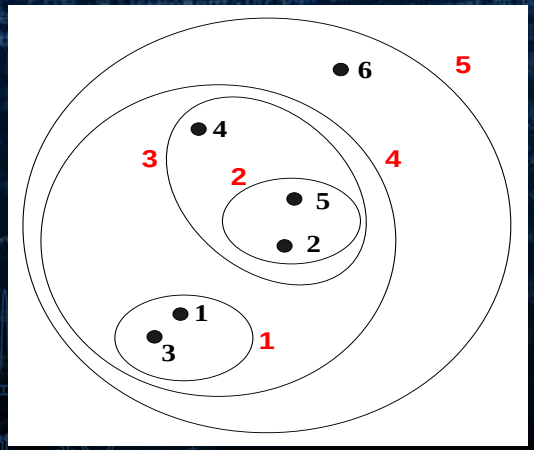
- ★ 什么是聚类
- ★ 层次聚类方法
- ★ 划分聚类方法
- ★ 基于密度的聚类方法
- ★ 基于子空间的聚类方法
- ★ 人工神经网络聚类方法

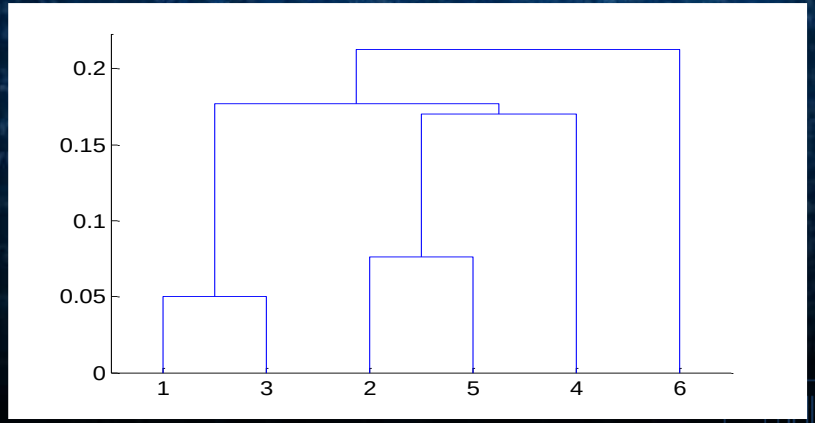
层次聚类方法

★Hierarchical Clustering

★ 基本思想

- 在聚类过程中生成一个聚类树。
- 完整聚类树的最顶端代表把整个数据集划作为一个簇；最底端代表把数据集中每一个数据都当作一个簇；树中父节点对应的簇包含着所有子节点对应的簇。
- 聚类树的不同层次可以表示聚类的不同粒度。





层次聚类方法

- 分为两大类：
  - 自顶向下（top-down）构造聚类树
    - 称之为分裂聚类法（Divisive Clustering）
  - 自下而上（bottom-up）构造聚类树，
    - 称之为凝聚聚类法（Agglomerative Clustering）

- 无论分裂聚类还是凝聚聚类都需要一个参数指明停止聚类的条件。
  - 通常用簇的期望个数k作为分层聚类判断停止的条件。
  - 即，如果当前簇的个数大于等于k（对于凝聚法则是小于等于k），则停止聚类。

常用的层次聚类算法

- ★ Linkage算法
  - 只能聚类凸集数据，有不同的簇间距离度量方法。
- ★ CURE算法
  - 可以聚类任意形状的数据集，但是不能处理具有分类属性的数据。
- ★ CHAMELEON算法
  - 可以聚类任意形状的数据集
- ★ BIRCH算法
  - 最少只扫描一遍数据集，时间复杂度为O(N)。
  - 所以非常适合于大规模数据的聚类。
  - 但是难以聚类非凸数据集。

Linkage算法

★ 基本步骤：

给定簇的期望个数k

1. 计算相似性矩阵
2. 将每个数据点看作一个簇
3. 重复
4. 合并两个最相近的簇
5. 更新相似性矩阵
6. 直到簇的个数为K个

★ 两个簇的相似性计算是该算法中的关键

Linkage算法

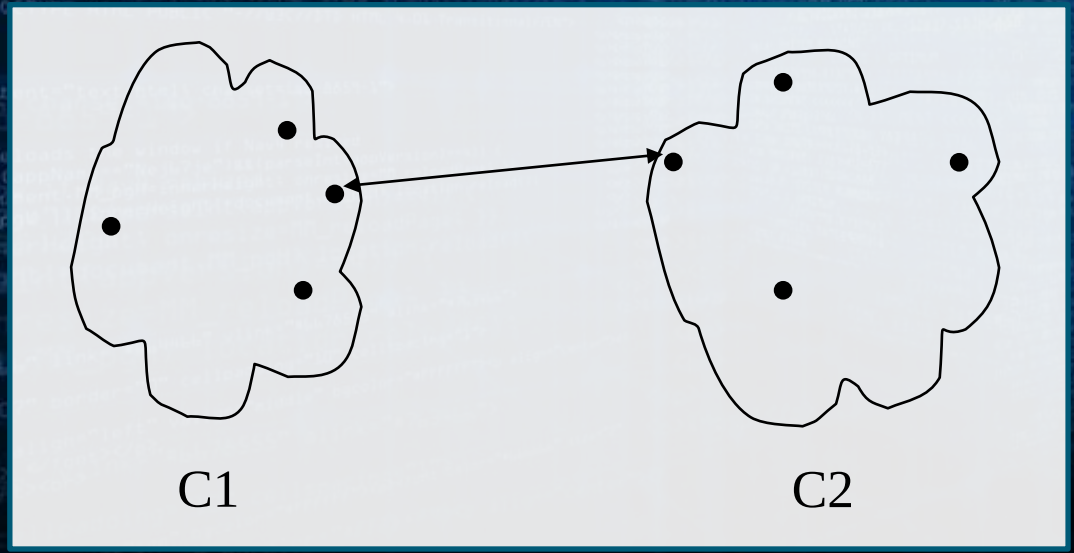
- Linkage算法一般用距离定义相似度。在合并或者分裂簇的时候，主要考虑簇间距离。一次只合并或分裂一个簇。

$$d(C_1, C_2) = L\{d(x, y) \mid x \in C_1, y \in C_2\}$$

- 簇间距离有三种度量方法：
  - 单链（Single Link）
  - 均链（Average Link）
  - 全链（Complete Link）

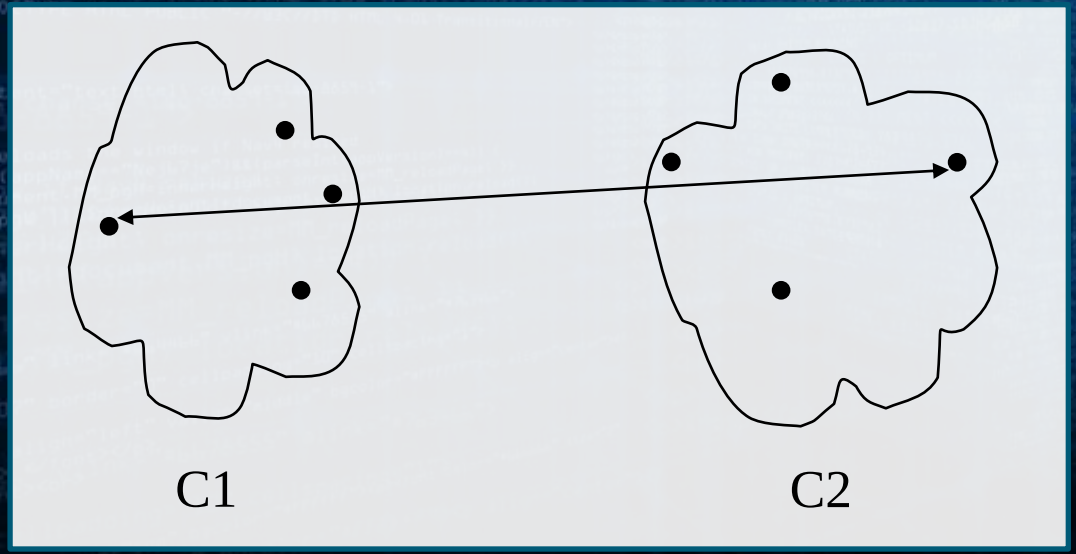
Slink算法

- ★ 单链（Single Link）
  - 算子L取极小化算子，即两个簇的距离等于两簇最近两个点间的距离



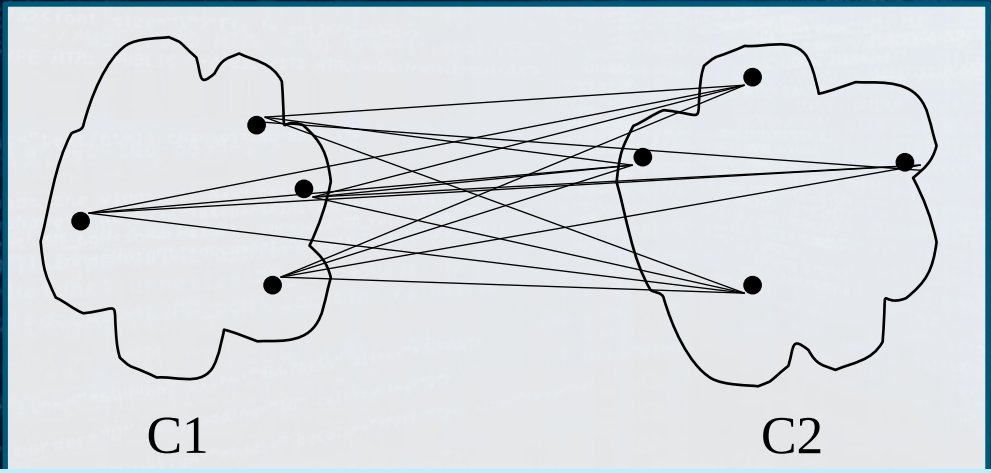
Clink算法

- 全链（Complete Link）
  - 算子L取极大化算子，即两个簇的距离等于两簇最远两个点间的距离



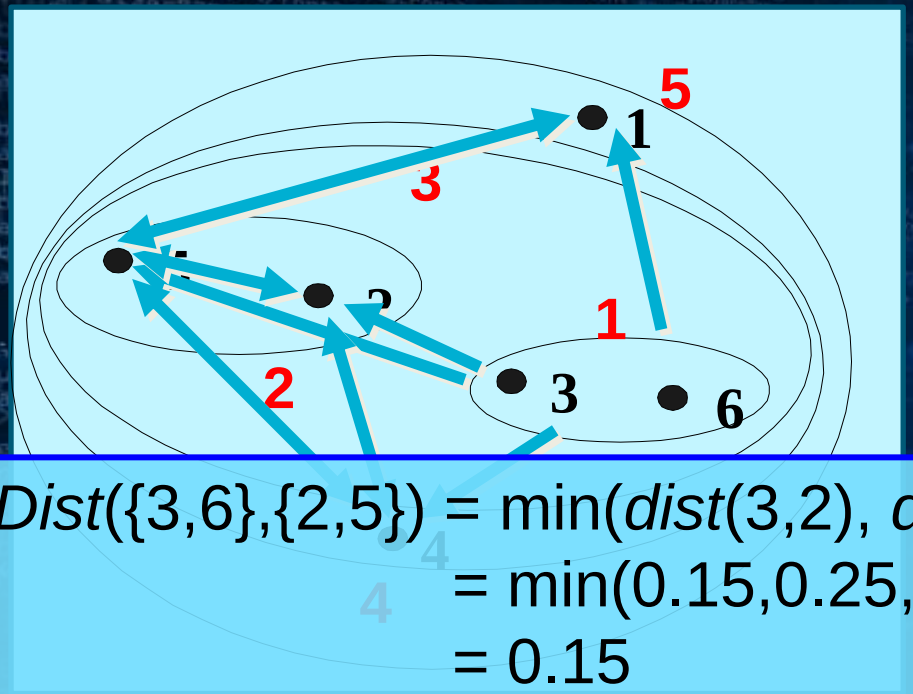
Alink算法

- ★ 均链（Average Link）
  - 算子L取平均算子，即两个簇的距离等于两簇点间距离的平均值



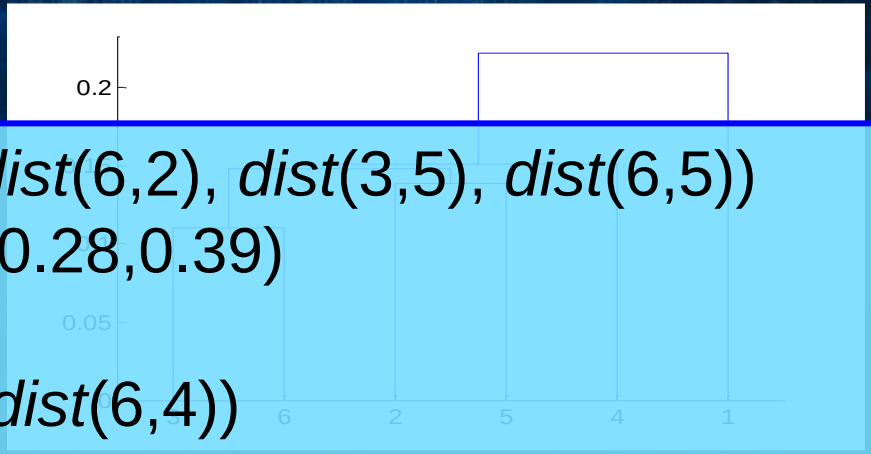
$$proximity(Cluster_i, Cluster_j) = \frac{\sum_{\substack{p_i \in Cluster_i \\ p_j \in Cluster_j}} proximity(p_i, p_j)}{|Cluster_i| * |Cluster_j|}$$

SLINK算法示例



	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

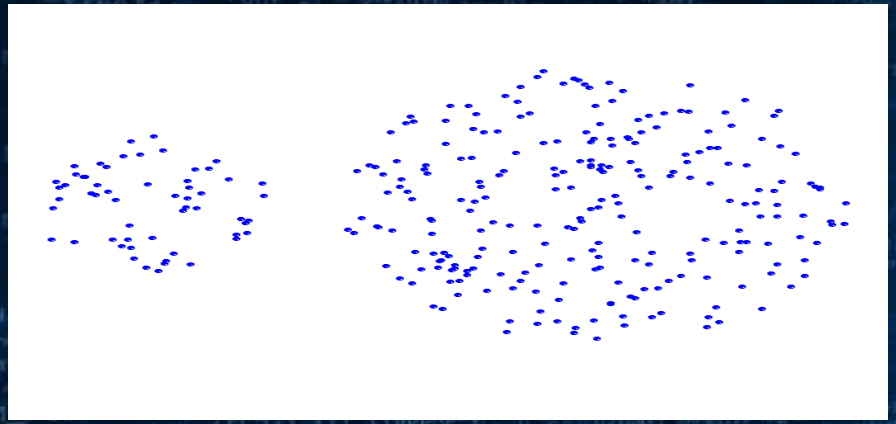
Euclidean distance matrix for 6 points.



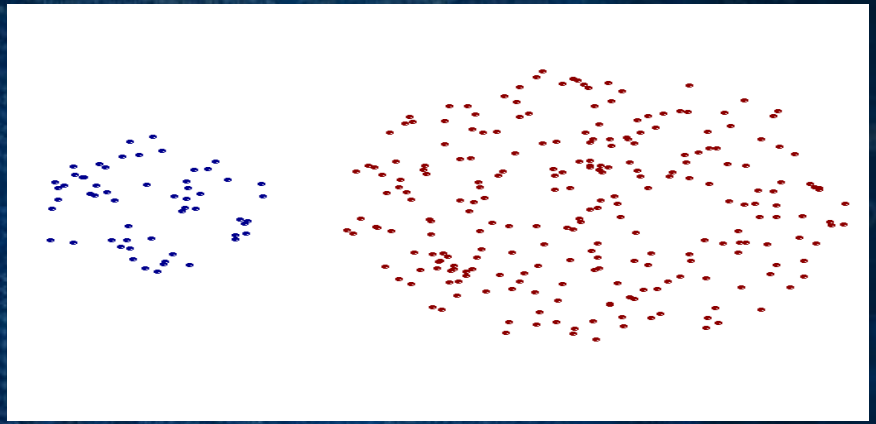
Dist({3,6},{2,5}) = min(*dist*(3,2), *dist*(6,2), *dist*(3,5), *dist*(6,5))
= min(0.15,0.25,0.28,0.39)
= 0.15

Dist({3,6},{4}) = min(*dist*(3,4), *dist*(6,4))
= min(0.15,0.22)
= 0.15

SLINK的优点



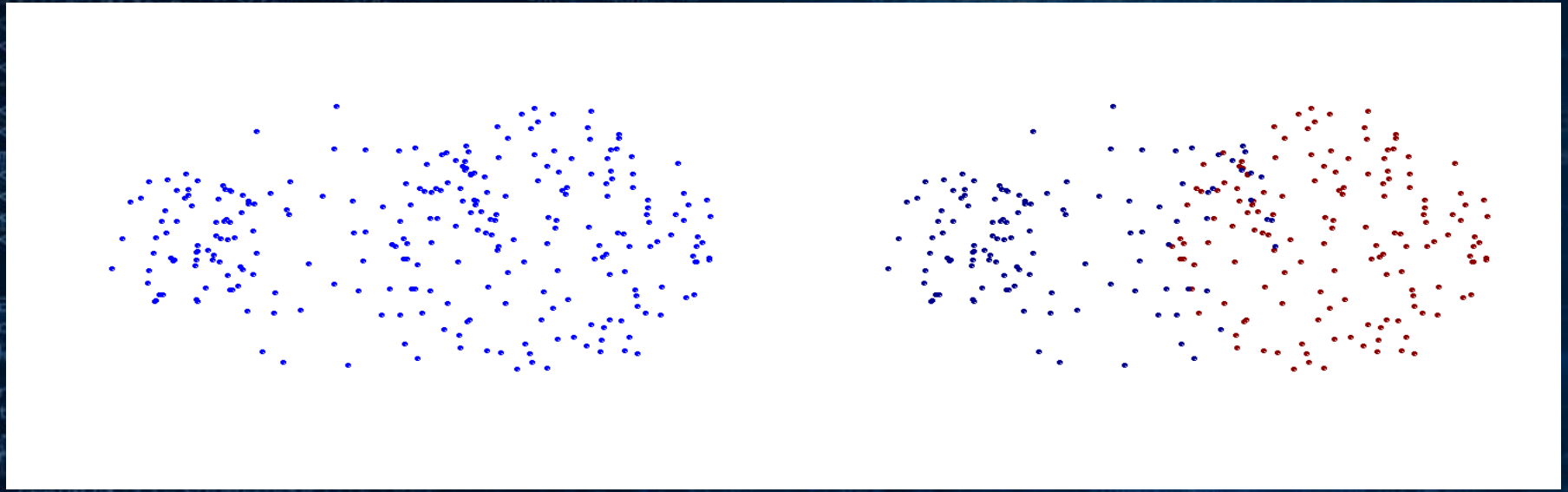
原始数据点



聚为两簇

- 易于处理簇大小不同的情况

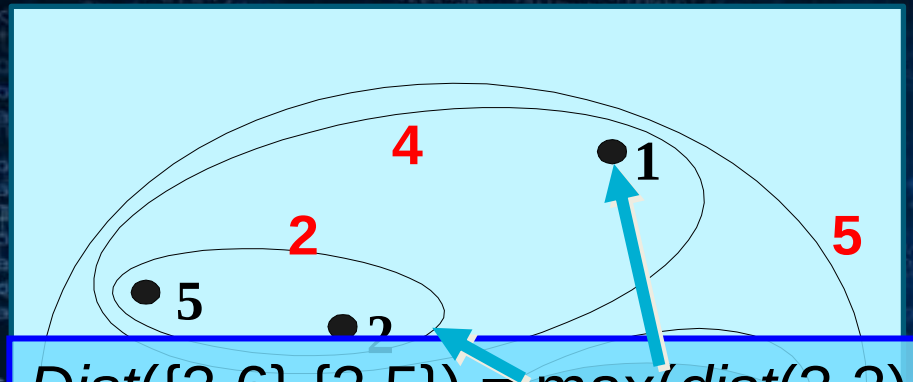
SLINK的缺点



原始数据点　　聚为两簇

- 对噪音和异常点很敏感

全链聚类算法示例



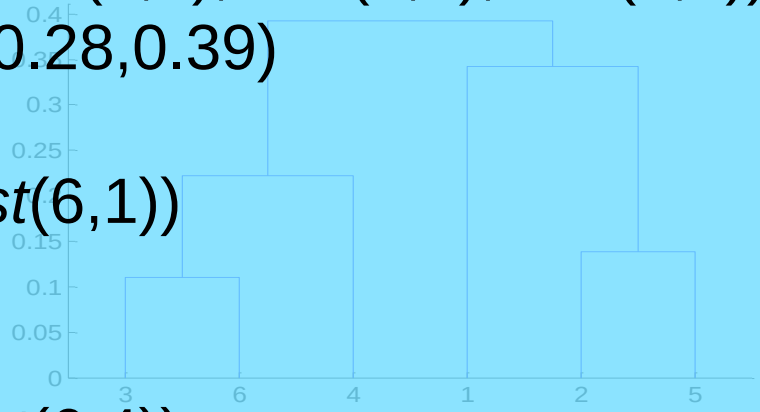
	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Euclidean distance matrix for 6 points.

Dist({3,6},{2,5}) = max(*dist*(3,2), *dist*(6,2), *dist*(3,5), *dist*(6,5))
= max(0.15,0.25,0.28,0.39)
= 0.39

Dist({3,6},{1}) = max(*dist*(3,1), *dist*(6,1))
= max(0.22,0.23)
= 0.23

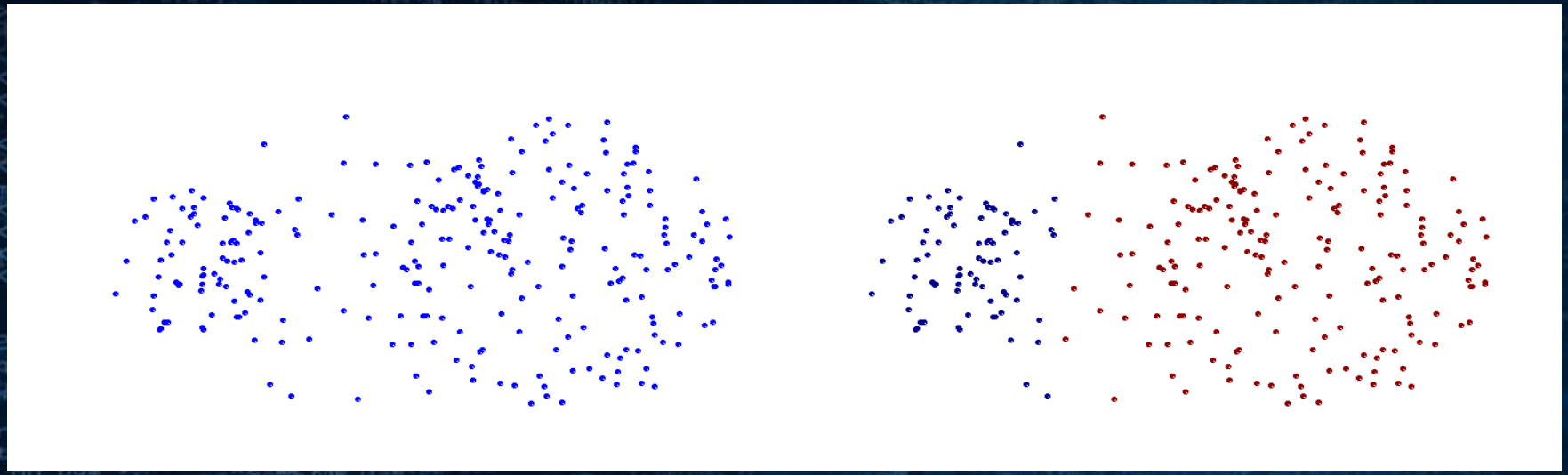
Dist({3,6},{4}) = max(*dist*(3,4), *dist*(6,4))
= max(0.15,0.22) = 0.22



嵌套类

聚类树

全链聚类的优点

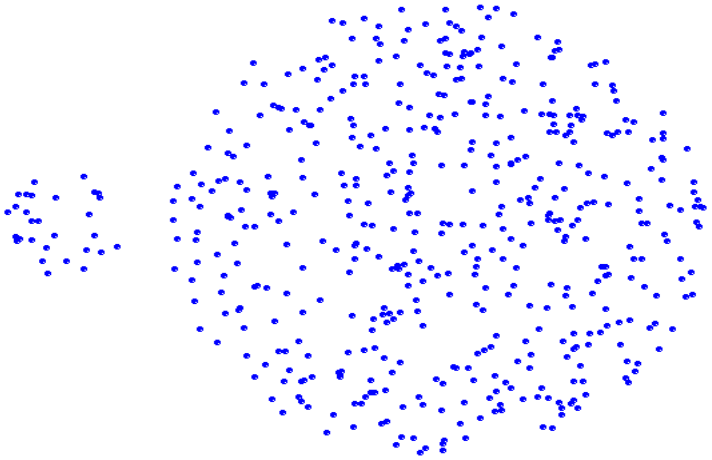


原始数据点

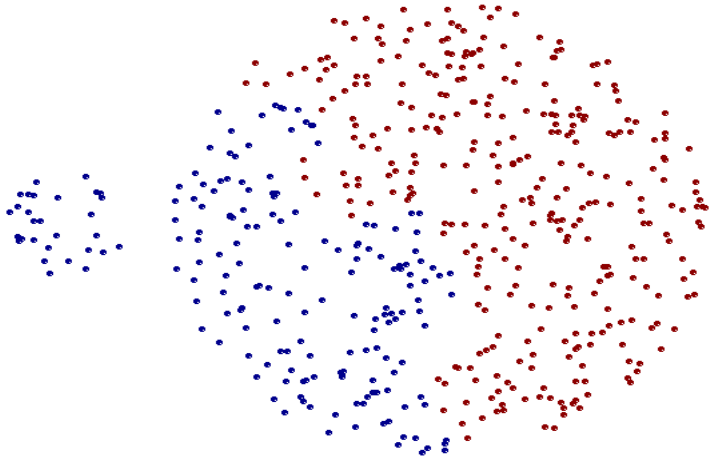
聚为两簇

- 受噪音和异常点的影响小

全链聚类的缺点



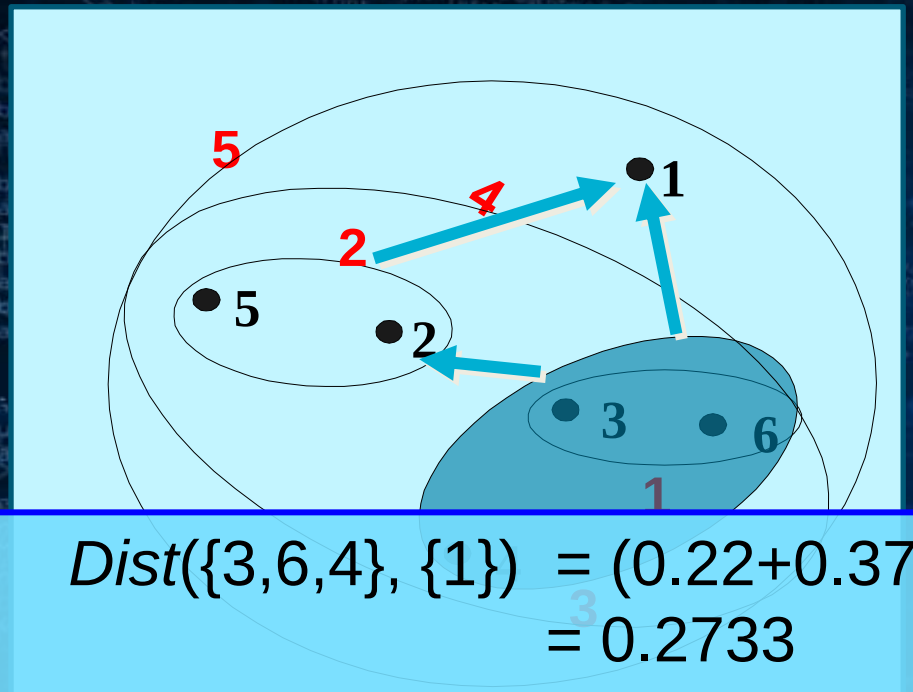
原始数据点



聚为两簇

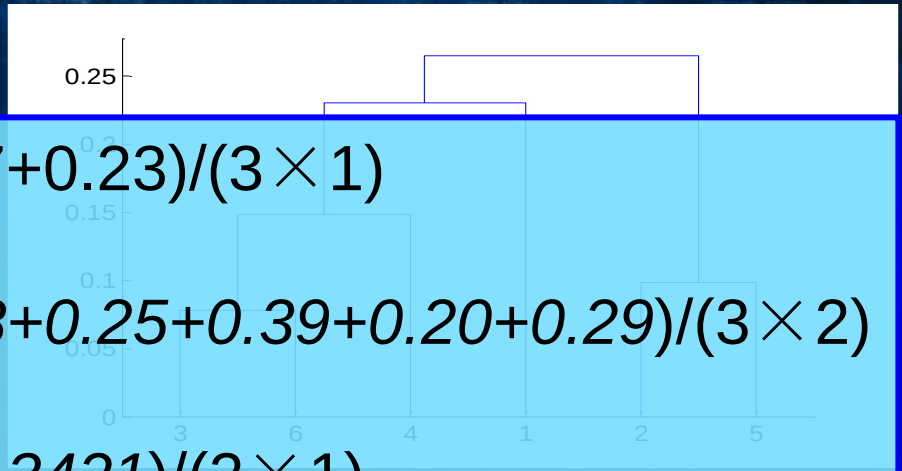
- 倾向于将大簇划分为多个簇

均链聚类算法示例



	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Euclidean distance matrix for 6 points.



嵌套类

聚类树

$Dist(\{3,6,4\}, \{1\}) = (0.22+0.37+0.23)/(3\times1)$
$= 0.2733$

$Dist(\{3,6,4\}, \{2,5\}) = (0.15+0.28+0.25+0.39+0.20+0.29)/(3\times2)$
$= 0.26$

$Dist(\{2,5\}, \{1\}) = (0.2357+0.3421)/(2\times1)$
$= 0.2889$

- ★ 均链是单链与全链的折中方案
- ★ 优点：受噪音和异常点的影响相对较小
- ★ 缺点：聚类结果倾向于球形

Linkage聚类算法复杂度

- 空间复杂度为$O(N^2)$
  - 使用了距离矩阵；N是点的数目

- 时间复杂度为$O(N^2)$
  - 聚类过程为K步，每一步要对大小为N^2的矩阵进行更新和搜索

CURE算法

- 也使用簇间距离定义相似度，然后运用凝聚法进行聚类。
- 簇间距离定义类似于Single Link算法，但并不是在两个簇的所有点中取得最短距离；而是首先寻找一些点作为各自簇的代表，然后取两个簇代表点中的最短距离作为簇间距离。

CURE算法

- CURE算法这样寻找一个簇的代表点。
  - 第一个代表点是距离该簇中心最远的点。
  - 然后选取距全部现有代表点最远的点作为下一个代表点。如此重复共选择c个点代表该簇。
  - 接下来所有这些代表点还要向簇中心收缩。收缩系数为a（$a \in [0, 1]$）。
- 理论上，簇的代表点基本上覆盖了簇的形状。
- 代表点向簇中心收缩的用意在于中和外边界点，特别是一些远离簇中心的点，对簇的影响。

CURE算法

- CURE算法用了两个措施来加速聚类过程
- 一、数据采样
  - CURE算法先在大规模数据上进行采样；
  - 然后在采样数据上进行聚类；
  - 最后把未被采样点直接划分给与其最相近的簇（实际是与该簇的代表点最相近）。
  - 此时CURE算法的时间复杂度是$O(M^2)$，M是采样点个数。
- 二、分区
  - 把采样点均匀分布在p个分区内，每个分区包含M/p个数据
  - 在p个分区内分别聚类，直至每个分区得到M/(pq)个簇；
  - 然后把所有分区内的簇合在一起，共pM/(pq)=M/q个簇的基础上，再进行聚类得到最终结果；
  - 最后把未被采样点直接划分给与其最相近的簇。

CHAMELEON算法

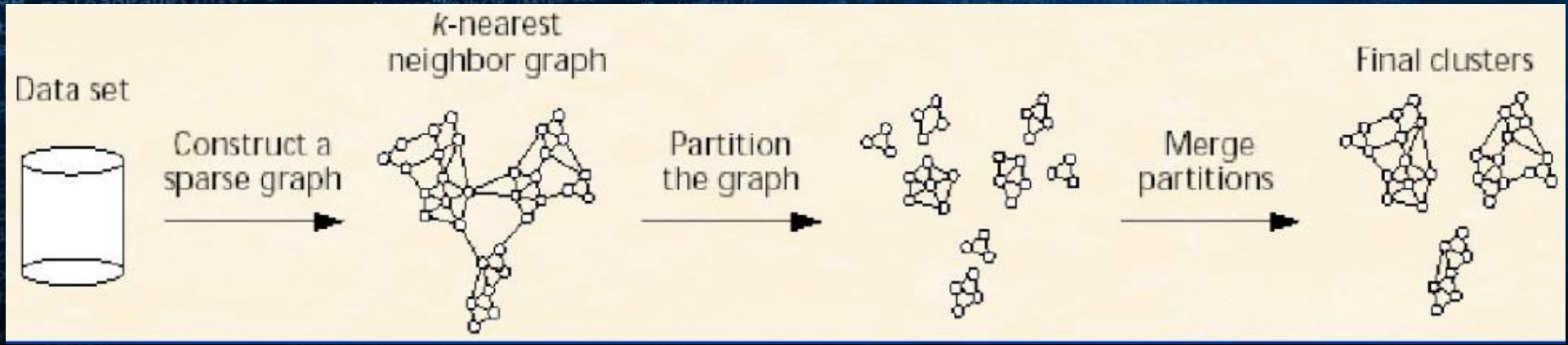
- ★ 以往算法的不足
  - 只处理符合静态模型的簇
  - 忽略互连性

    互连性：簇间距离较近数据对的多少
  - 忽略近似性

    近似性：簇间数据对的相似度（最近距离）
- ★ 变色龙算法同时考虑了互连性和近似性

CHAMELEON算法思路

- ★ 首先由数据集构造一个k-最近邻图Gk；
- ★ 再通过图划分算法将图Gk划分成大量子图，每个子图代表一个初始子簇；
- ★ 用一个凝聚的层次聚类算法反复合并子簇，得到结果簇。



稀疏图

- 节点表示数据项
- 边表示边的一个节点v在另一个节点u的k个最相似点中。
  - 边的权重表示这两个节点间的近似度，距离越大，近似度越小，权重越小。
  - 好处：
    - 距离很远的数据项完全不相连
    - 变得权重代表了潜在的空间密度信息
    - 在密集和稀疏区域的数据项都能建模
    - 表示的稀疏便于使用有效的算法

相对互连性（RI）

★ 相对互连性函数：

$$RI(C_i, C_j) = \frac{|EC(C_i, C_j)|}{\frac{|EC(C_i)| + |EC(C_j)|}{2}}$$

- $\mathrm{EC}(C_i, C_j)$:连接簇C_i和C_j的所有边的权重
- $\mathrm{EC}(C_i)$：把簇C_i划分为两个大致相等部分的最小等分线切断的所有边的权重和。
- 相对互连性能处理簇间形状不同和互连程度不同的问题。

相对近似性（RC）

★ 相对近似性函数：

$$RC(C_i, C_j) = \frac{\bar{S}EC(C_i, C_j)}{\frac{|C_i|}{|C_i| + |C_j|}\bar{S}EC(C_i) + \frac{|C_j|}{|C_i| + |C_j|}\bar{S}EC(C_j)}$$

★ $\bar{S}EC(C_i, C_j)$:连接簇C_i和C_j的边的平均权重

★ $\bar{S}EC(C_i)$：把簇C_i划分为两个大致相等部分的最小等分线切断的所有边的平均权重

CHAMELEON算法聚类过程

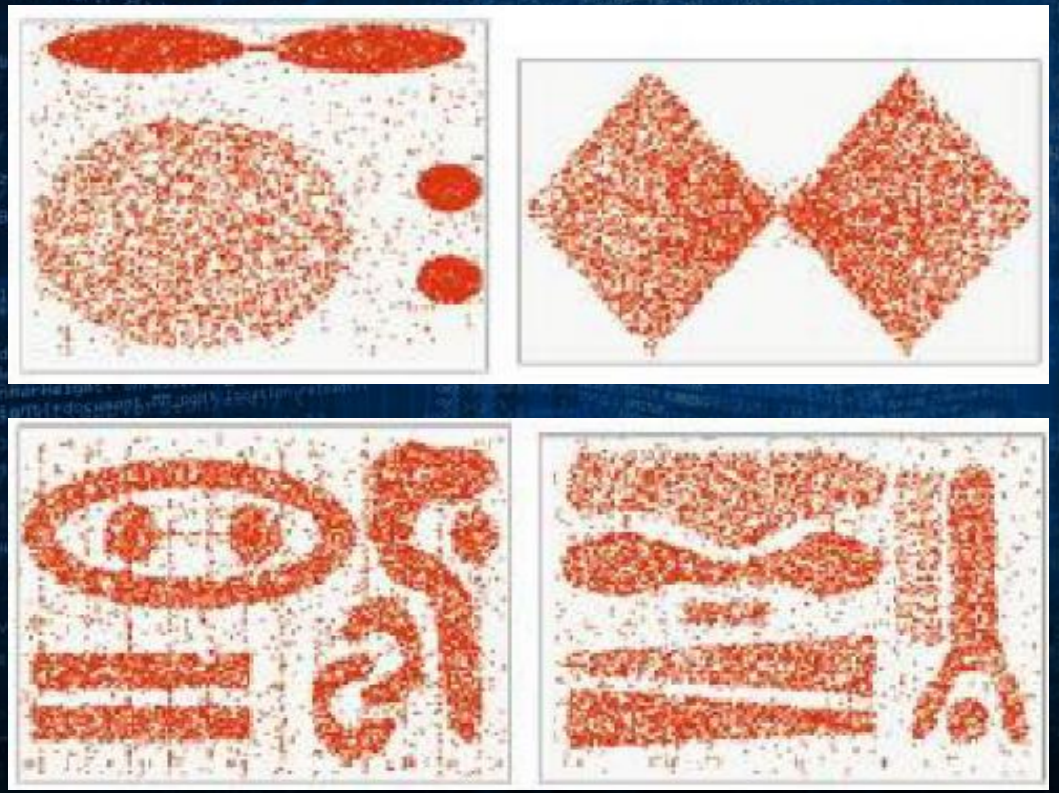
- 第一阶段：构造稀疏Gk图
  - 对数据集中每个数据点找出它的所有k-近邻对象，并分别在它们之间加带权边。
- 第二阶段：划分Gk图
  - 首先把Gk图划分成两个近似的等大小的子图,使分区的边的总权重最小(这个过程称为最小截断)。
  - 然后把每一个子图看成一个初始的子图,重复上述过程直至生成每一个子图中结点数量达到一定的标准。

CHAMELEON算法聚类过程

- 第三阶段：合并子簇
  - 函数定义： $RI(C_i, C_j) \times RC(C_i, C_j)^{\alpha}$
    - α>1，更重视相对近似性
    - α<1，更重视相对互连性
  - 访问每个簇，计算其与邻近簇的RI和RC
  - 选择使函数值最大的簇对合并
  - 重复上两步，直到没有可以合并的簇

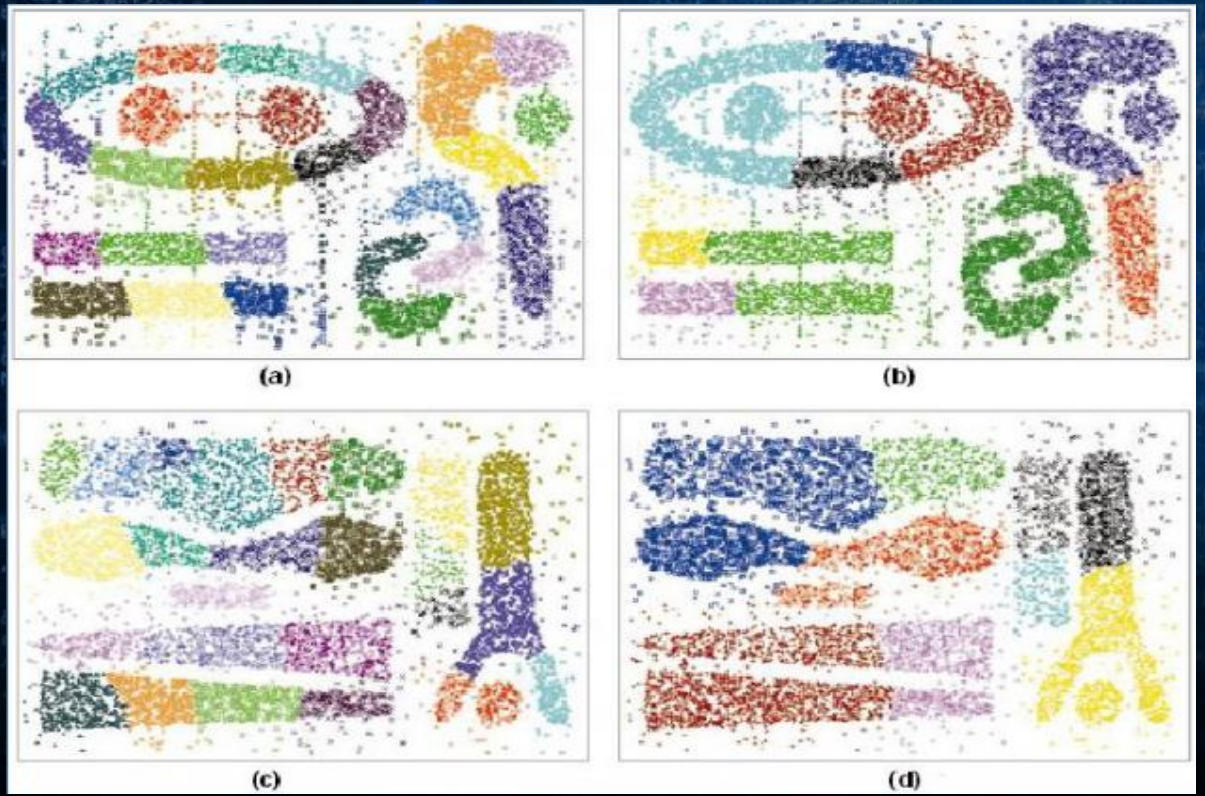
聚类结果示例

实验数据图：



★ CURE算法的聚类结果

25个簇

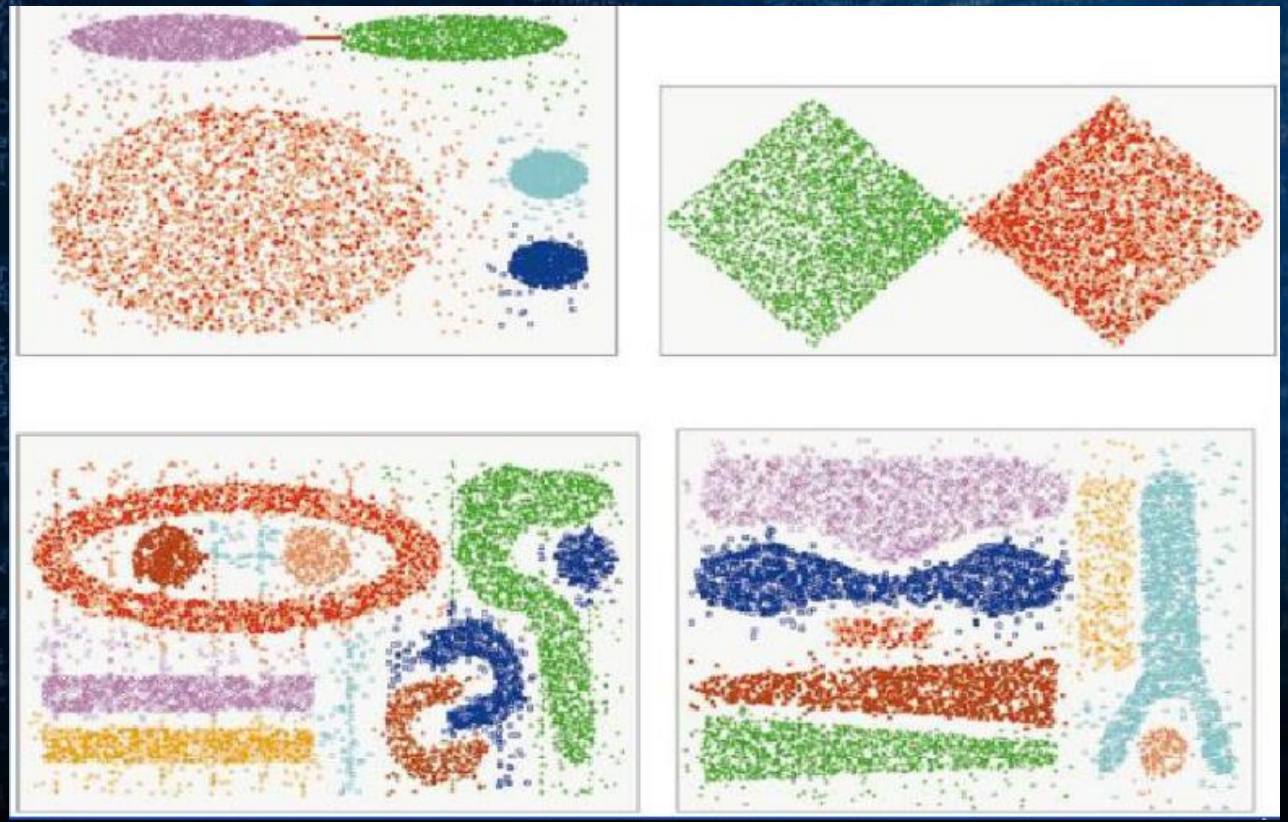


11个簇

25个簇

11个簇

变色龙算法的聚类结果（k=10，α=2）



- ★ 变色龙算法充分考虑了互连性和近似性
- ★ 可以发现高质量的任意形状的簇
- ★ 在最坏情况下，高维数据的处理所需的时间复杂度可能会达到$O(N^2)$。

层次聚类的特点

- ★ 在聚类过程中一次性建好聚类树，没有回溯调整操作。
  - 一个数据点一旦属于每个簇之后就一直属于该簇，不会更改。
  - 一个簇一旦被合并或者分裂之后，也不会再调整其中的数据点。
- ★ 优点：
  - 算法简单，适用性强，数据扫描顺序对聚类结果无影响，不用担心组合数目的不同选择。
- ★ 缺点：
  - 没有全局优化，如果某一步没有很好地合并或者分裂，则必将导致低质量的聚类结果。

第四章 聚类方法

- ★ 什么是聚类
- ★ 层次聚类方法
- ★ 划分聚类方法
- ★ 基于密度的聚类方法
- ★ 基于子空间的聚类方法
- ★ 人工神经网络聚类方法

划分聚类方法

★ Partitional Clustering

★ 基本思想

- 每个簇都与一个簇心相关联
- 每个点被分配给距离最近的簇心
- 簇的数目K必须确定

★ 算法步骤

1. 选择K个点作为初始簇心
2. repeat
3. 将所有点分配给距离最近的簇心
4. 重新计算每个簇的簇心
5. until 簇心不再变化

基本的划分聚类方法

- 划分聚类方法大多数使用距离定义数据相似度。
- 距离度量
  - 选取一个点y作为簇C的代表，然后计算簇外一点x和y两点间的距离就是点x到簇C的距离。
- K平均方法（K-means）
  - 簇的代表点是簇的理论中心（centroid）。
  - 理论中心点不一定是簇内真实存在的数据点。
- K代表点方法（K-medoids）
  - 簇的代表点是簇内最靠近理论中心的数据点（即最有代表性的数据点）。

划分聚类方法

- 在划分聚类过程中，簇心在不断地调整。调整的目的是使数据集上的划分到达全局优化。
- 全局优化一般是指误差最小。
- 误差一般取数据点到簇代表点距离的平方和。

$$Error = \sum_{i=1}^{k} \sum_{x \in C_i} \| x - r_i \|^2 = \sum_{i=1}^{k} \sum_{x \in C_i} \sum_{j=1}^{m} (x_j - r_{ij})^2$$

- 全局最优和误差都可以有其它的定义方式。

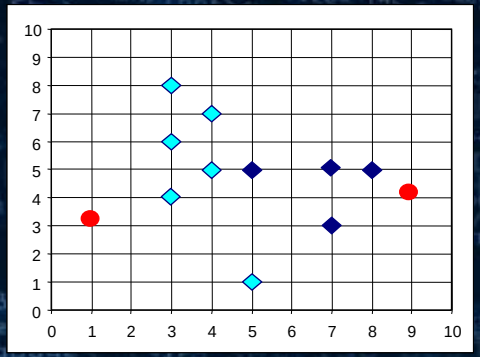
K平均聚类方法（K-means）

- 基本过程：
  - 第一步 从数据集中随机选择K个数据点作为初始簇代表点；
  - 第二步 将数据集中每一个数据点按照距离分配给与其最近的簇；
  - 第三步 重新计算每个簇的中心，获得新的代表点；
  - 第四步 如果所有簇的新代表点均无变化，则算法结束；否则转至第二步。
- 簇理论中心常用算术平均公式计算：

$$r = \frac{1}{|C_i|}\sum_{x \in C_i} x$$

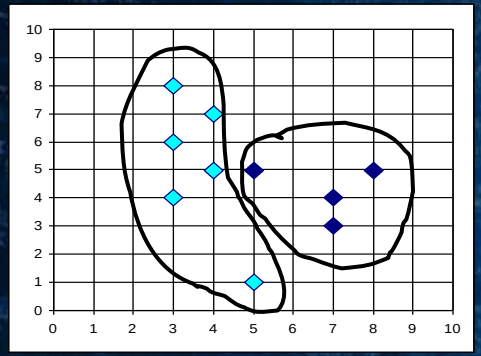
K-means算法示例

★ 以2个簇为例

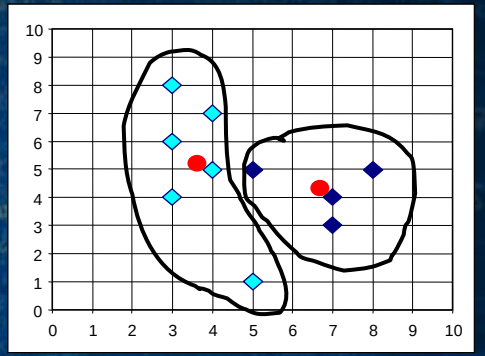


K=2

随机选择K个
初始簇心

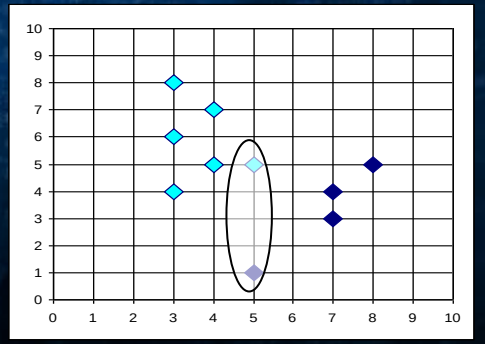


第一轮
分配

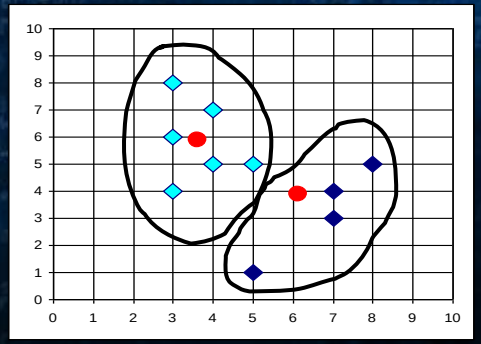


更新簇
心

重新分配



更新簇
心



簇心不
再变化

K-means聚类算法

- 确定初始点
  - 往往采用随机选择方法。但是随机初始点不能很好地反映簇分布，所以聚类结果往往不佳。
  - 其它手段。例如根据密度划分区域并选取初始点。
- 距离度量通常采用欧氏距离、余弦相似度或相关性等方法
- K-means在上述相似度度量方法下都会进行收敛
- 大多数收敛过程发生在开始的几次迭代中

K-means聚类算法

- 簇中心
  - 不再变化，就是采用欧几里德距离定义的误差收敛到了一个极小点。
  - 基本的K平均方法不能保证聚类结果一定收敛到误差全局最小点。
  - 基本的K平均方法是把所有点都分配给簇以后，才重新调整簇中心。
  - 另一种方法是渐进更新簇中心，即每当簇得到一个新数据点则立刻更新簇中心。
- 算法复杂度为O(n*K*I*d)
  - n=数据点的数目，K=簇的个数
    I=迭代次数　　，d=属性数目

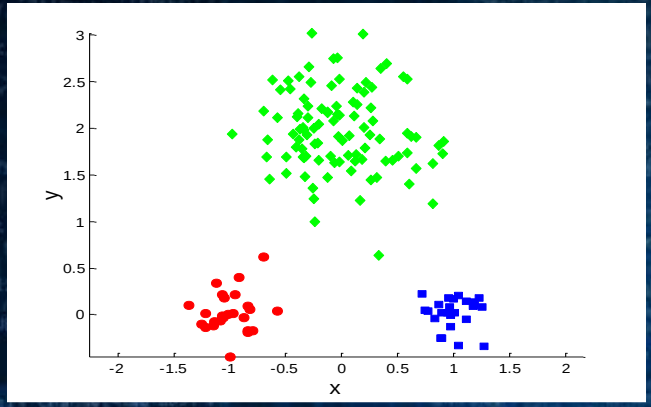
K-means聚类结果评估

- 最普遍的评估方法是误差平方和（SSE）
  - 只考虑簇内是否紧密，未考虑簇间是否远离
  - 对于每个点，误差是点到所在簇簇心的距离
  - SSE的公式：

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)$$

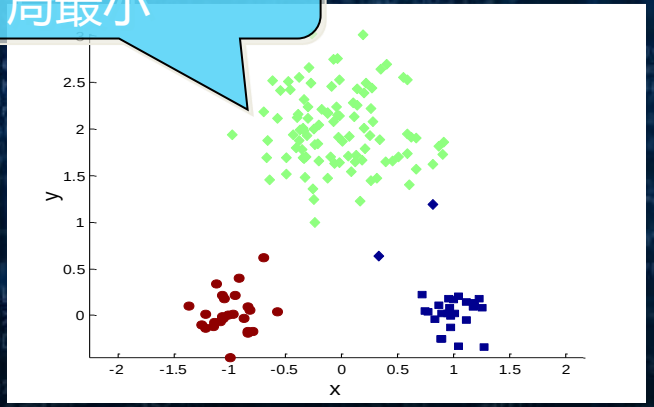
    - x是簇C_i中的一个数据点，m_i是簇C_i的代表点，即簇心
  - 给定两个簇集合，可以选择误差平方和小的一个
  - 一个降低SSE的方法是增加簇的数目，即K值
    - K值较小但聚类结果好的簇集合比K值较大但聚类效果差的簇集合的SSE要低

K-means聚类结果评估



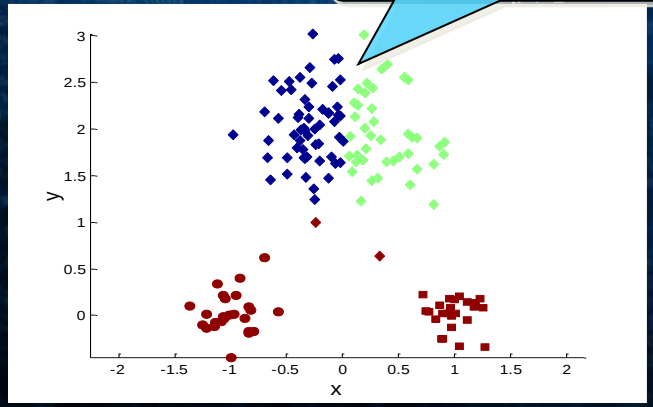
原始数据点

3个簇时SSE达到全局最小



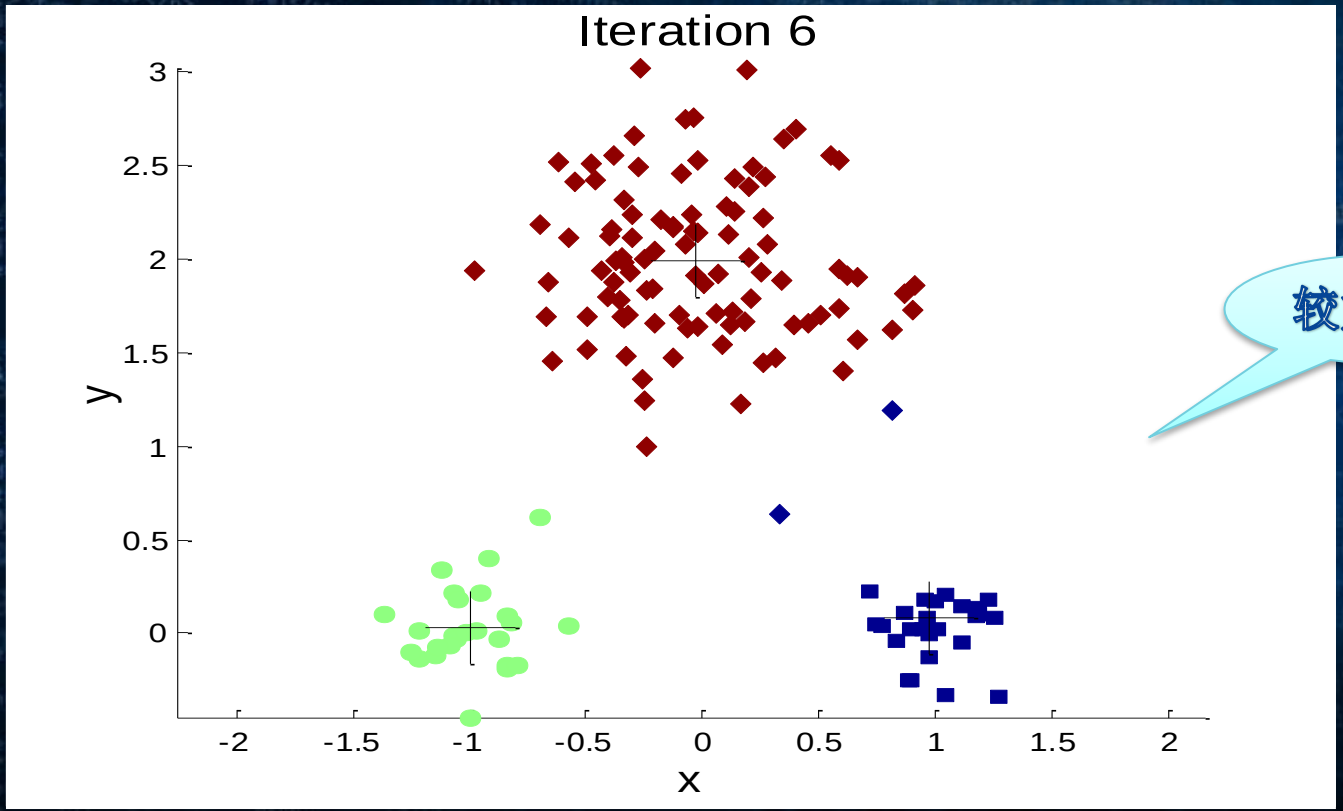
最优聚类

4个簇时SSE局部最小

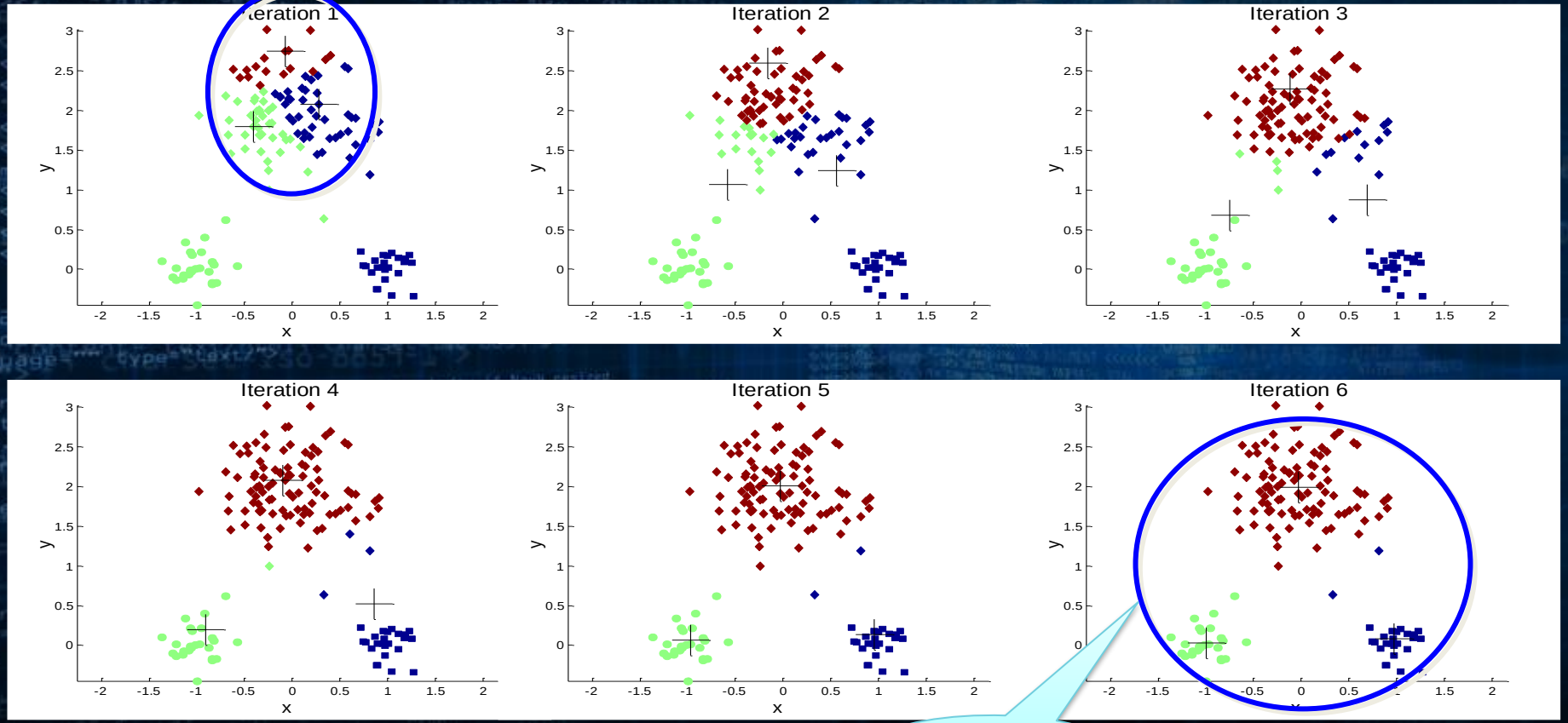


次优聚类

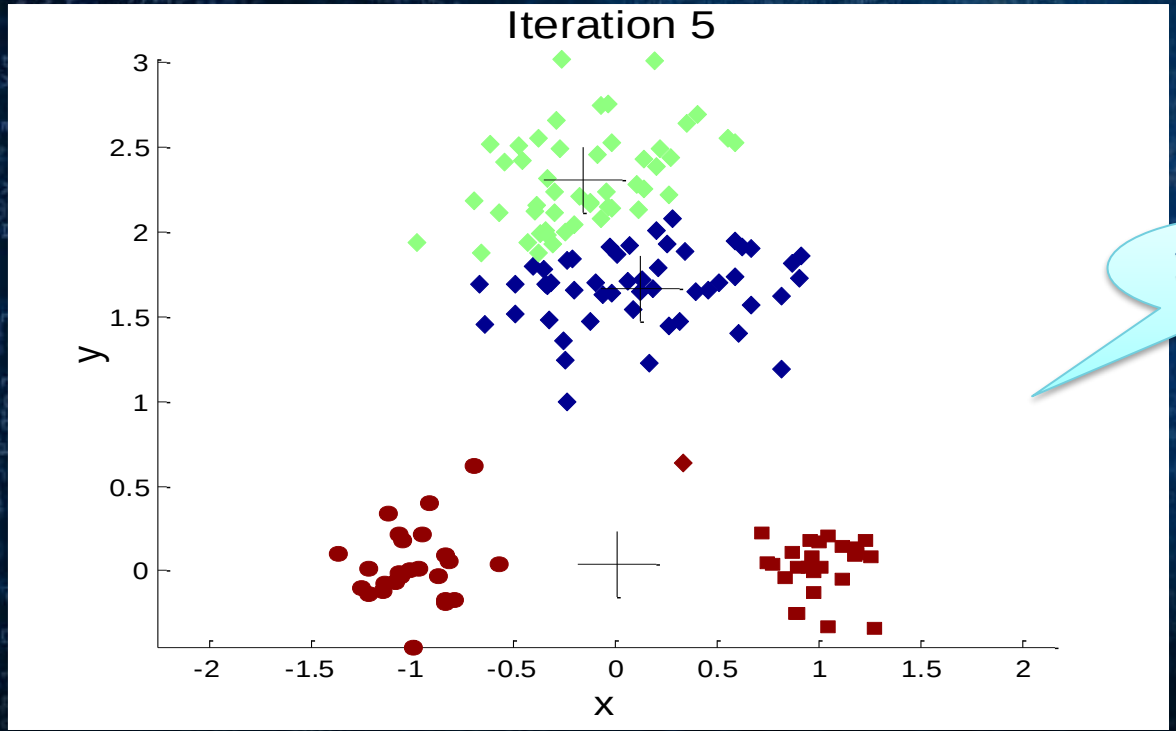
选择初始簇心的重要性



选择初始簇心的重要性


Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
x
y
较好结果

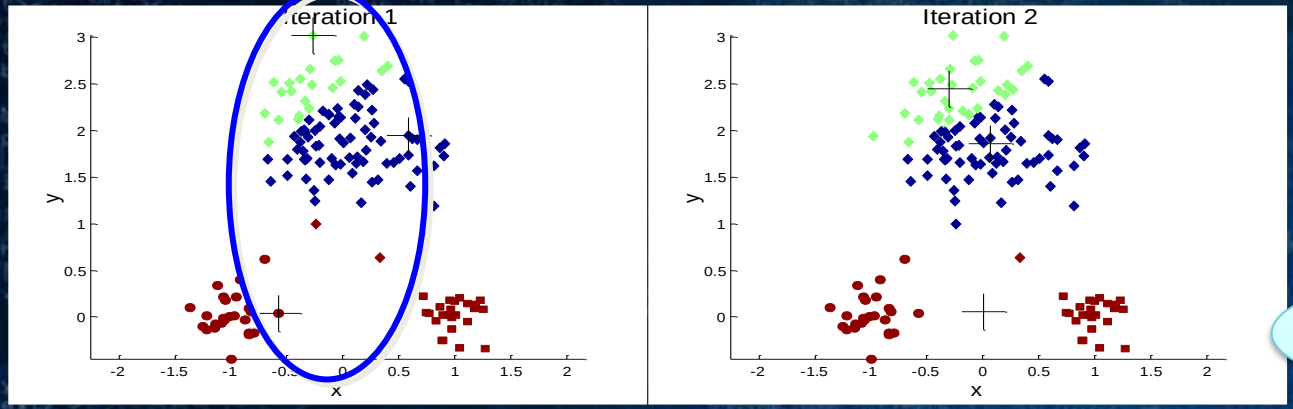
选择初始簇心的重要性



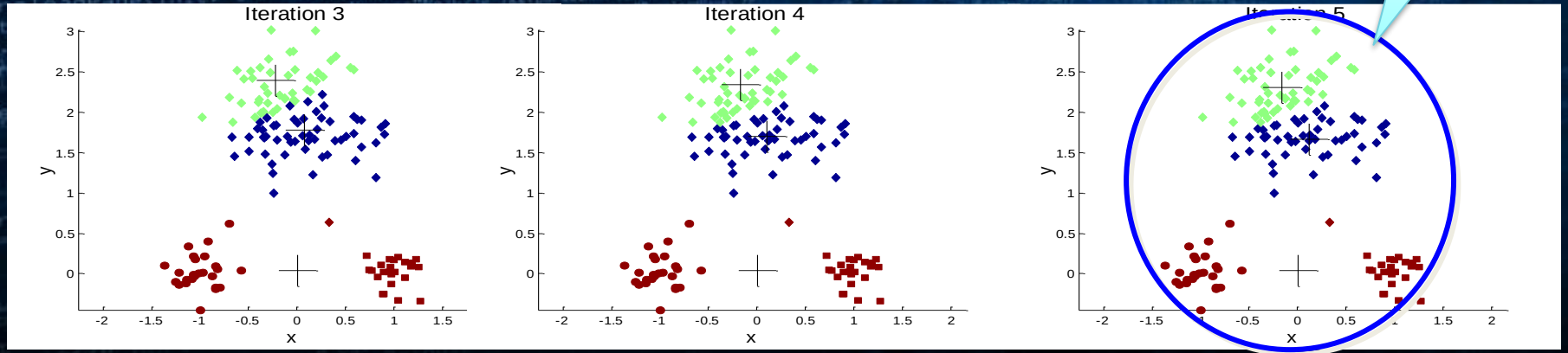
较差结果

选择初始簇心的重要性

Machine Learning & Data Mining



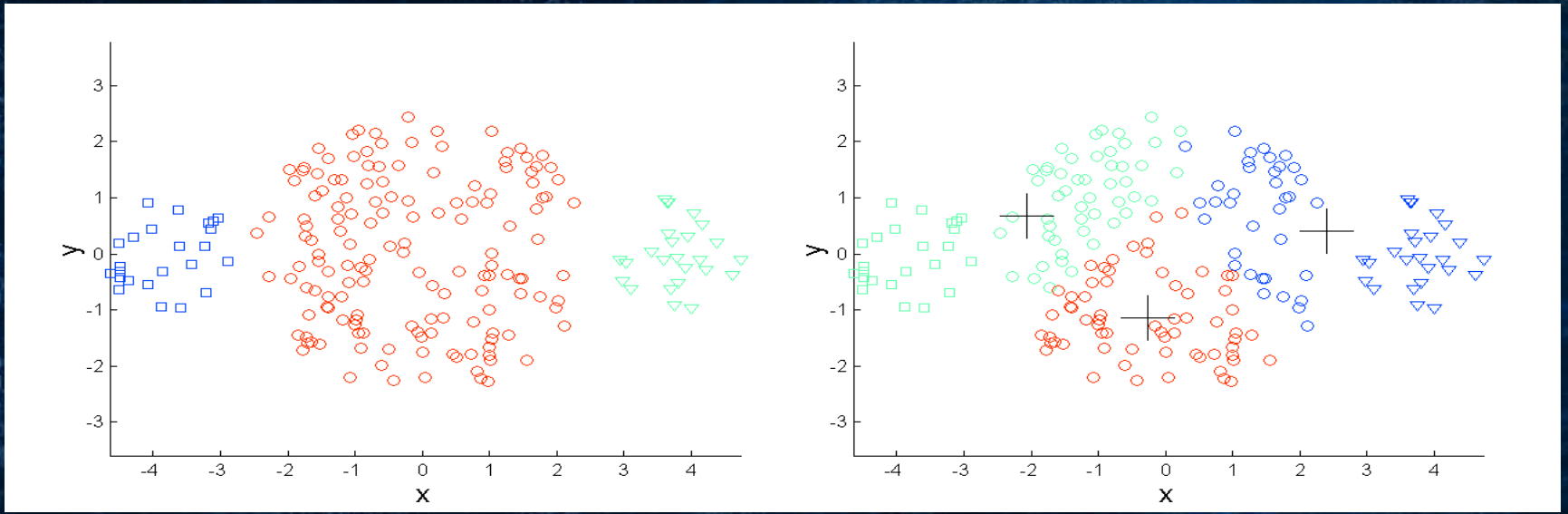
较差结果



K-means算法

- K-means算法经常会结束于局部最优解
  - 可以通过确定性退火和遗传算法达到全局最优
- K-means的局限性
  - 需要提前确定簇的数目
  - 对噪声和异常值敏感
  - 不适合发现非凸形状的簇
  - 无法处理簇的大小或密度不同的情况

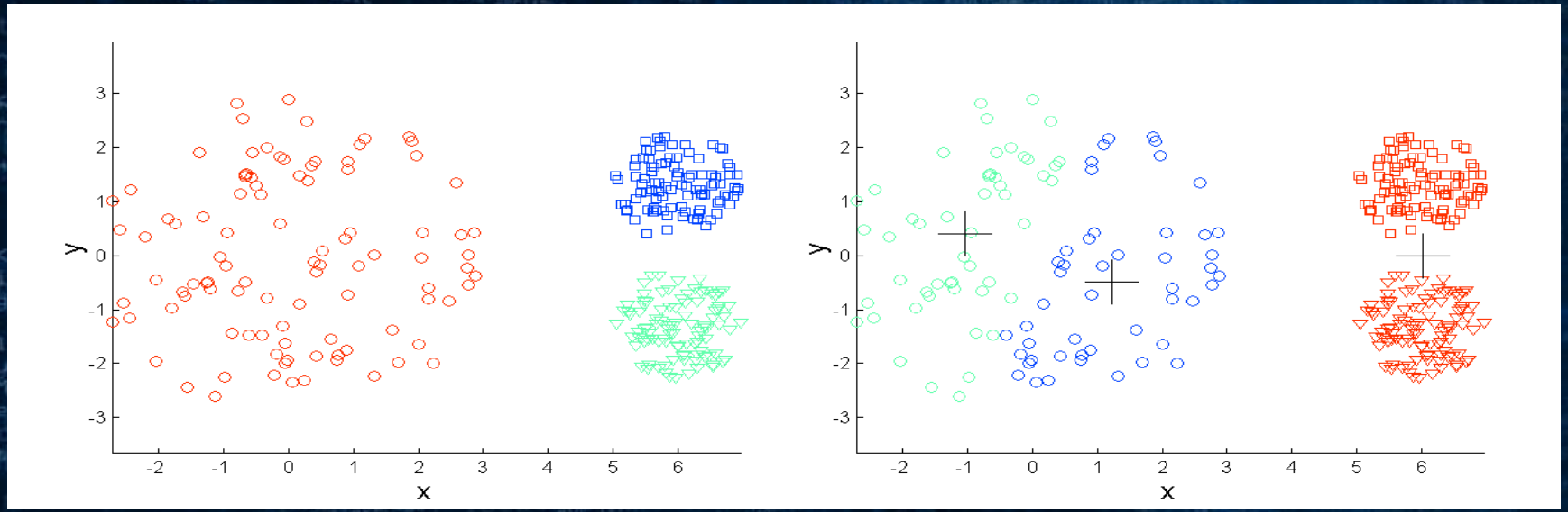
K-means的局限性: 簇不同大小



原始数据点

K-means3个簇

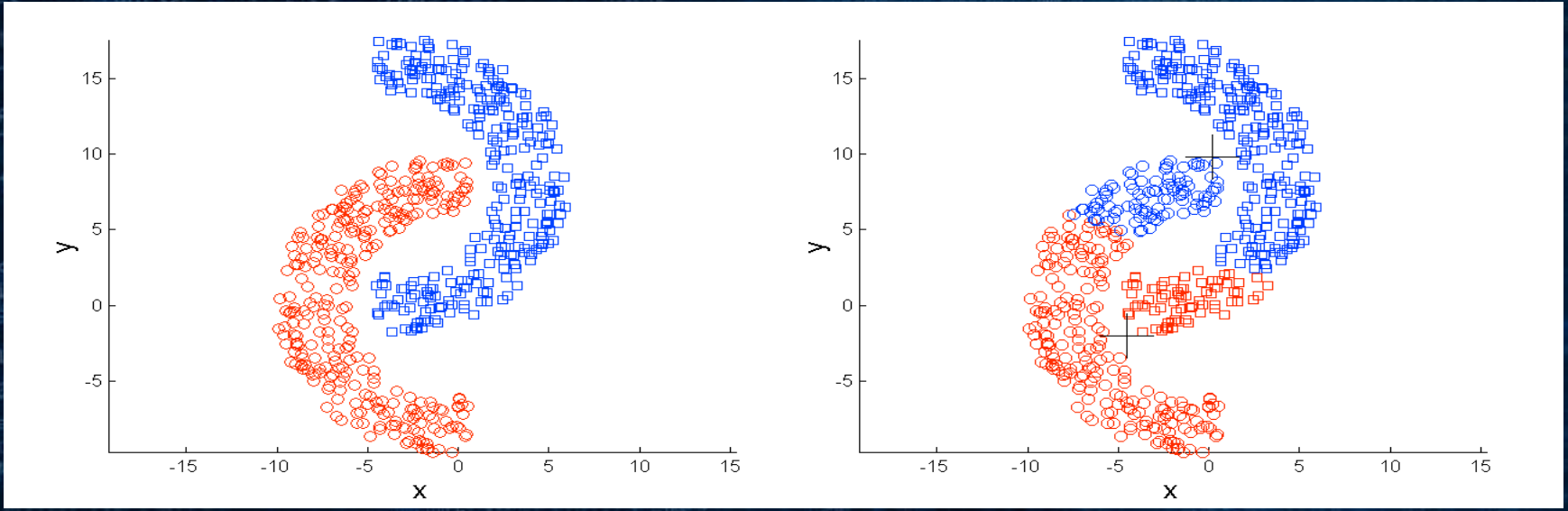
K-means的局限性: 簇密度大小



原始数据点

K-means3个簇

K-means的局限性：非球形簇



原始数据点

K-means2个簇

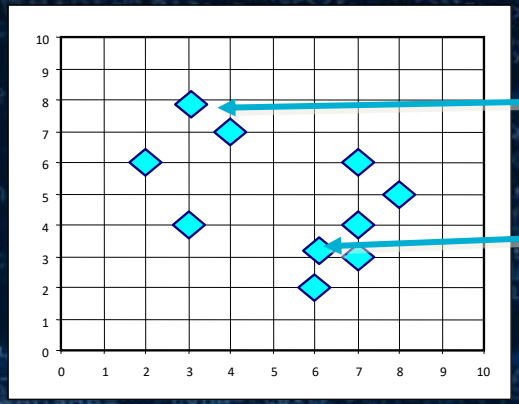
K代表点聚类方法（K-medoids）

- 与K平均聚类方法的过程基本相同。
- 在选择簇代表点时
  - 计算候选代表点与簇内其它所有点间相似度之和。
  - 然后取相似度和最大的点作为簇代表点。

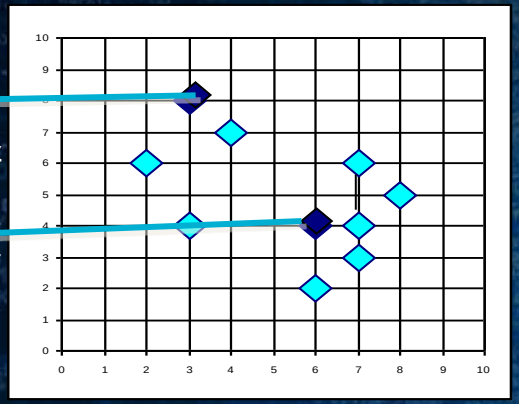
$$x^* = \arg\max \sum_{C_i} sim(x, x^*)$$

  - 如果采用距离定义相似度，则上式就变为使其它点到代表点的距离之和最小。
- 相似度可以有各种不同定义
  - 所以K代表点方法不局限于可度量数据，还可以处理具有分类属性的数据。

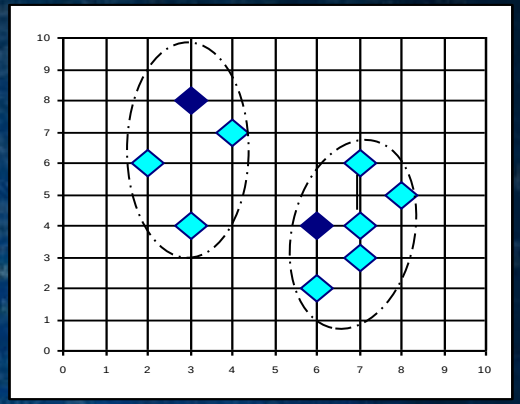
一种典型K-Medoids算法(PAM)



随机选择k个点作为初始质心

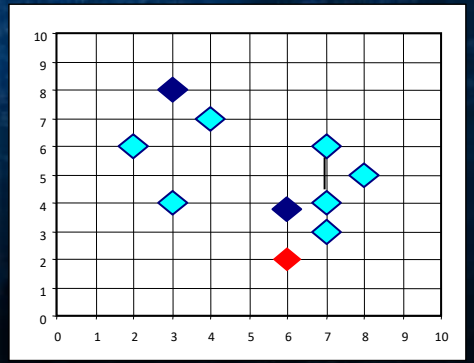


分配数据点

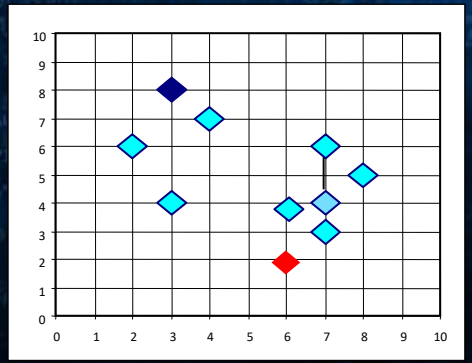


Total Cost = 20

随机选择一个非质心的点



计算簇心交换后的质量



Total Cost = 26

K=2

Do loop

Until no change

如果质量提高就变换簇心

划分聚类的特点

- 时间复杂度与数据集大小成线性关系。
- 对于非凸集合以及簇大小相差悬殊的数据效果不好，并且数据扫描顺序会影响选择簇中心。
- K平均方法由于要求理论中心，所以只能处理可度量的数据，难以处理具有分类属性的数据。K代表点方法则可以处理任何数据。
- 孤立点和噪声数据对K平均方法的影响更大一些。
- 簇个数k对于划分聚类方法很重要。
  - 如何确定最优k值仍然依赖于经验。
- 初始簇中心（代表点）对于划分聚类结果的影响很关键。
  - 随机选择的初始簇中心往往不能获得较好结果。

第四章 聚类方法

- ★ 什么是聚类
- ★ 层次聚类方法
- ★ 划分聚类方法
- ★ 基于密度的聚类方法
- ★ 基于子空间的聚类方法
- ★ 人工神经网络聚类方法

基于密度的聚类方法

★ Density Based Clustering

★ 基本思想

- 将簇看作是数据空间中被低密度区域分割开的高密度区域。

★ 主要特点

- 可以发现任意形状的簇
- 可以处理噪音和异常点
- 扫描一遍数据集
- 需要密度参数作为终止条件

基于密度的聚类方法

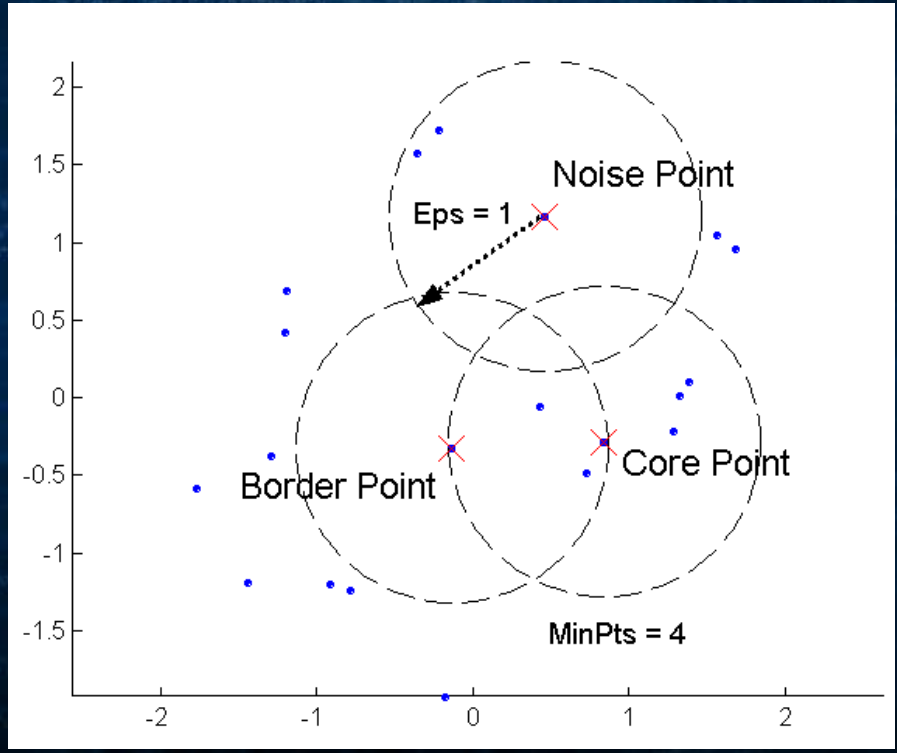
- DBSCAN算法
  - 依赖于邻域半径和密度阈值两个参数
  - 但是这两个参数并不易确定最优值
- OPTICS算法
  - 通过一系列的邻域半径来控制簇生长
- DENCLUE算法
  - 用密度分布函数来聚类

DBSCAN算法

- DBSCAN可以在含有噪音的空间数据库中发现任意形状的簇
- 依赖于基于密度的簇的概念
  - 一个簇被定义为一个最大密度连通点集。
- 定义
  - Eps：最大邻域半径
  - MinPts：密度阈值，一个点的Eps邻域内数据点的最小数

DBSCAN算法

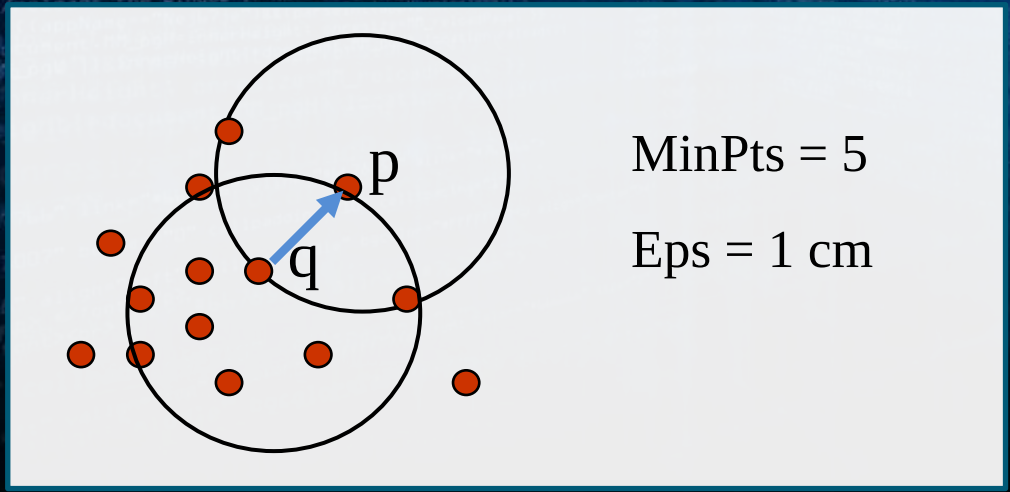
- ★ 核心点
  - 核心点的Eps邻域中的数据点数目大于MinPts
- ★ 边界点
  - 边界点的Eps邻域中数据点数目小于MinPts，但该点位于一个核心点的Eps邻域中
- ★ 噪音点
  - 不是核心点也不是边界点的点



密度可达和密度相连

- ★ $N_{Eps}(p)$: $\{q \in D \mid dist(p,q) <= Eps\}$
- ★ 直接密度可达

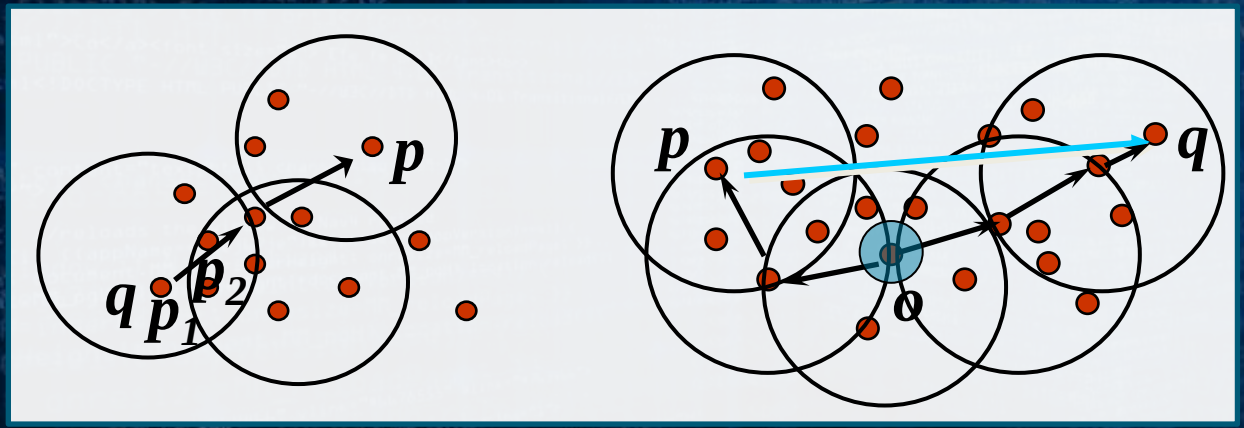
  称一个点p是从点q直接密度可达的，当
  - $p \in N_{Eps}(q)$
  - 核心点条件：$|N_{Eps}(q)| >= MinPts$，即q是核心点



密度可达和密度相连

★ 密度可达

称一个点p是从点q密度可达的，当存在一连串的点$p_1,\ldots,p_n$，$p_1=q$，$p_n=p$并且p_{i+1}是从p_i直接密度可达的



★ 密度相连

称点p与点q是密度相连的，当存在一个点o使得p和q都是从o密度可达的

DBSCAN算法

★ 算法步骤

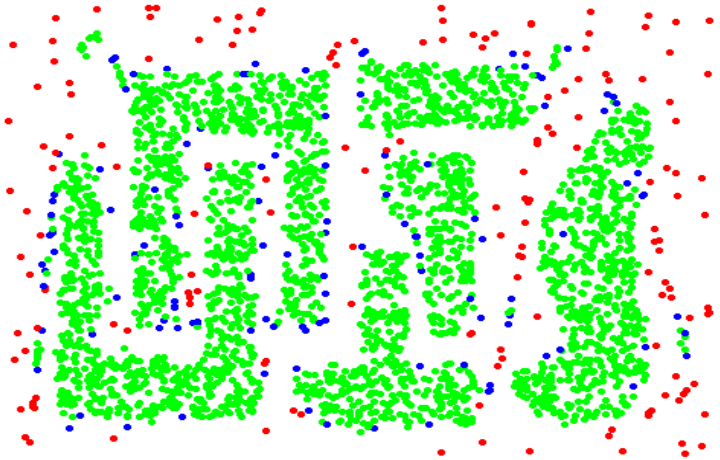
- 随机选择一个点p
- 检索所有从p密度可达的点
- 如果p是一个核心点，就形成了一个簇
- 如果p是一个边界点，没有点从p密度可达，此时DNSCAN选择下一个点
- 重复上述过程直到遍历完所有的点

★ 时间复杂度

- 由于可使用树结构，所以DNSCAN算法的时间复杂度为O(Nlog(N))

核心点、边界点与噪音点

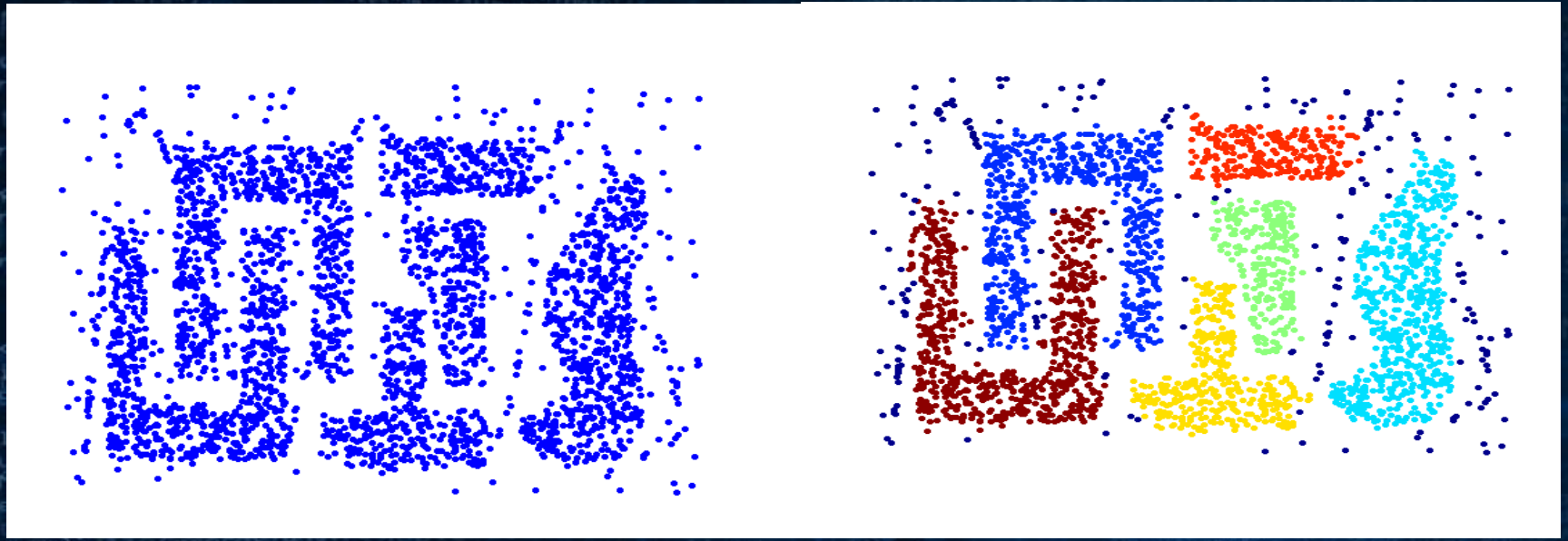
原始数据点



点的类型：核心点、边界点和噪音点

Eps = 10, MinPts = 4

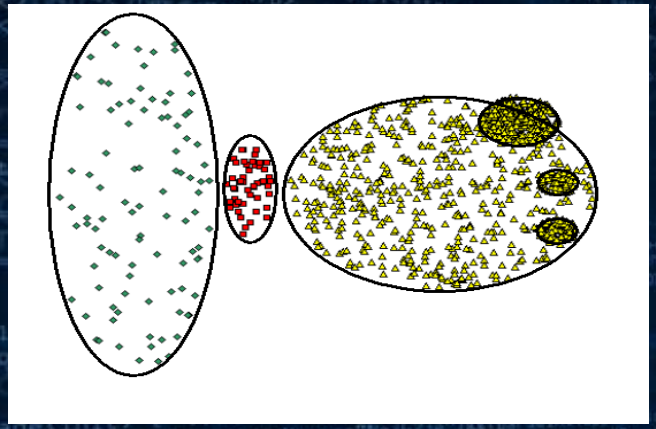
DBSCAN算法的优点



原始数据点　　簇集合

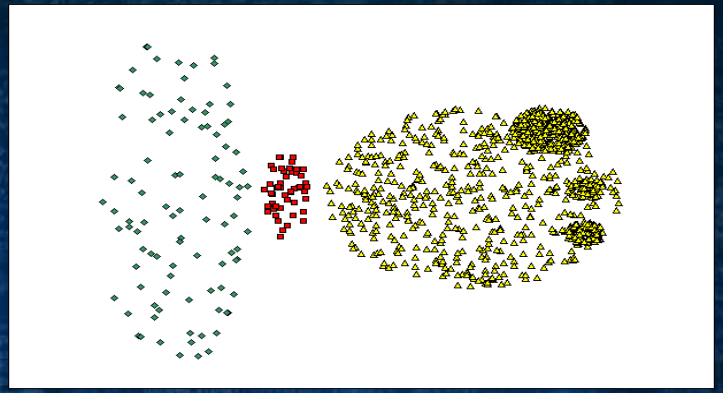
- 抗噪
- 可以处理不同形状和大小的簇

DBSCAN算法的局限性

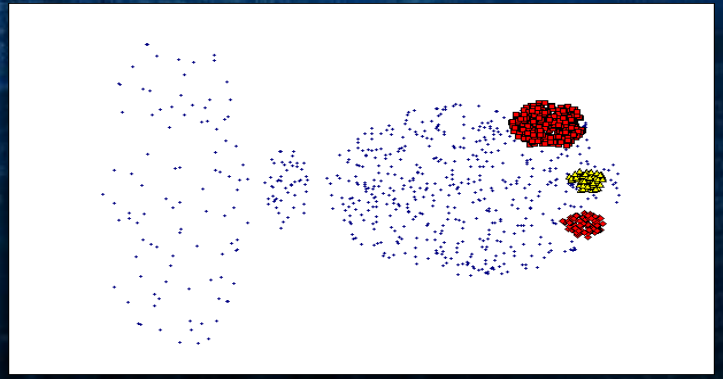


原始数据点

- 簇的密度不同
- 高维数据



(MinPts=4, Eps=9.92)



(MinPts=4, Eps=9.75)

OPTICS算法

- 对于真实的，高维的数据集合而言，参数的设置通常是依靠经验，难以确定的。
- 绝大多数算法对参数值是非常敏感的：设置的细微不同可能导致差别很大的聚类结果。
- OPTICS算法通过对象排序识别聚类结构。
- OPTICS没有显示地产生一个数据集合簇，它为自动和交互的聚类分析计算一个簇排序，这个次序代表了数据的基于密度的聚类结构。

DENCLUE算法

- DENCLUE算法
  - 对k-means聚类算法的一个推广；
  - 可得到全局最优划分。
- 主要基于：
  - 每个数据点的影响可以用一个数学函数来形式化地模拟，它描述了一个数据点在邻域内的影响，被称为影响函数；
  - 数据空间的整体密度可以被模型化为所有数据点的影响函数的总和；
  - 然后聚类可以通过确定密度吸引点来得到，这里的密度吸引点是全局密度函数的局部最大。

DENCLUE算法

- 算法步骤：
  - 1. 对数据点占据的空间推导密度函数；
  - 2. 识别局部最大点；
  - 3. 通过沿密度增长最大的方向移动，将每个点关联到一个密度吸引点；
  - 4. 定义与特定的密度吸引点相关联的点构成的簇；
  - 5. 丢弃密度吸引点的密度小于用户指定阈值的簇；
  - 6. 合并通过密度大于等于指定阈值的点路径连接的簇。

第四章 聚类方法

- ★ 什么是聚类
- ★ 层次聚类方法
- ★ 划分聚类方法
- ★ 基于密度的聚类方法
- ★ 基于子空间的聚类方法
- ★ 人工神经网络聚类方法

基于子空间的聚类算法

★Subspace Clustering

- 主要用于处理高维、海量数据

★基本思想

- 样本之间的差异往往是由若干个关键的特征所引起的，
- 如果能恰当的找出这些重要特征，不仅可以减少模型的建立时间，提高模型预测的准确率，还能有效地提高数据挖掘算法的鲁棒性和适应性。
- 因此，对于数据的高维特征，对其各个特征的重要性进行加权，或者挑选出最重要的特征子集，减少或消除冗余特征以及不相关特征的影响，最大限度的保留和利用原始数据中的关键特征。

基于子空间的聚类算法

- ★ 子空间
  - 即数据某几个维构成的空间。
- ★ 子空间聚类算法
  - 把数据的原始特征空间分割为不同的特征子集，
  - 从不同的子空间角度考察各个数据簇聚类划分的意义，
  - 同时在聚类过程中为每个数据簇寻找到相应的特征子空间。
- ★ 子空间聚类算法将传统的特征选择技术和聚类算法进行结合。
- ★ 可以分为硬子空间聚类和软子空间聚类两种形式。

基于子空间的聚类算法

- ★ 硬子空间聚类方法

  根据子空间搜索方式的不同，可以分为两类：

  - 自底向上的子空间聚类算法
  - 自顶向下的子空间聚类算法

- ★ 软子空间聚类方法
  - 软子空间聚类是指在聚类过程中，对每个数据簇的各个特征赋予相应的特征加权系数，在聚类过程中得到各个特征在对应数据簇中的重要性。
  - 与硬子空间聚类方法相比，具有更好的适应性与灵活性。

自底向上子空间聚类算法

- 自底向上子空间聚类算法一般基于网格密度
  - 先将原始特征空间分成若干个网格
  - 再以落到某网格中样本点的概率表示该子空间的密度情况
  - 对于密度超过一定阈值的子空间作为密集单元进行保留，同时舍弃非密集的子空间
- 经典算法
  - 静态网格聚类算法CLIQUE（维增长子空间聚类方法）
  - 利用熵理论作为密度度量的ENCLUS方法
  - MAFIA和DOC方法等

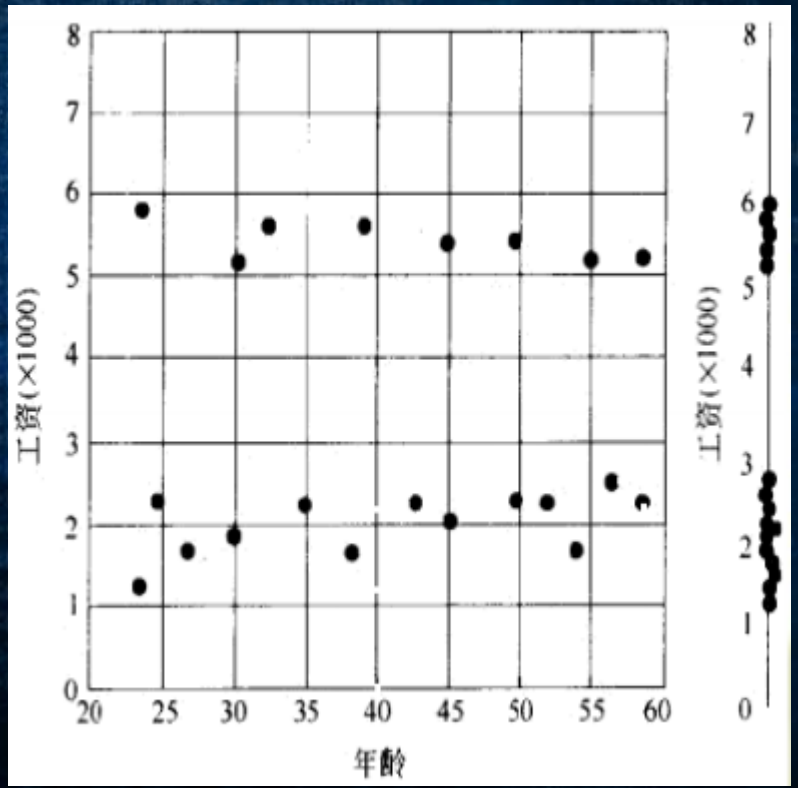
CLIQUE算法

- CLIQUE算法基本思想
  - 采用了基于网格和密度的方法。
  - 首先对每个属性进行等分，
  - 整个数据空间就被分成一个超立方集合，
  - 对每个单元进行数据点计数，大于某个阈值的单元称为稠密单元，
  - 然后对稠密单元进行连接构成类。

- 该算法可以自动识别嵌入在数据子空间中的类。

CLIQUE算法

- 在由年龄和工资构成的原始空间中没有密集区域，
- 但是在由工资一维构成的子空间中，存在两个密集区域，行成两个类（1000<=工资<=3000和5000<=工资<=6000）。
- 而在由年龄一维构成的子空间中没有密集区域，不形成聚类。



CLIQUE算法

- 算法步骤：
  - 1. 找出对应于每个属性的一维空间中的所有稠密区域；
  - 2. 2←k；
  - 3. repeat；
  - 4. 由稠密的k-1维单元产生所有的候选稠密k维单元；
  - 5. 删除点数低于阈值的单元；
  - 6. k←k+1；
  - 7. until 不存在候选稠密k维单元；
  - 8. 通过取所有邻接的、高密度的单元并发现簇；
  - 9. 使用一小组描述簇中单元的属性值域的不等式概括每一个簇。

CLIQUE算法的特点

- ★ 优点
  - 可以自动发现最高维的子空间，高密度聚类存在于这些子空间中
  - 对元组的输入顺序不敏感，无需假设任何规范的数据分布
  - 随输入数据的大小线性的扩展
  - 当数据维数增加时具有良好的可伸缩性
- ★ 缺点
  - 容易破坏密集区域的边缘，降低最终结果的准确性
  - 不能自动去除数据集中的孤立点，增加了计算复杂性
  - 可能会剪掉一些密集单元，对最终的聚类结果质量造成影响
  - 算法的多个步骤都采用近似算法，降低聚类结果的精确性

自顶向下子空间聚类算法

- 自顶向下子空间聚类算法主要基于数据投影技术以及迭代搜索策略
  - 首先将整个样本集划分为C个数据簇
  - 对于每个数据簇赋予相同的权值，并为每一类的各个特征赋予不同权重
  - 利用迭代策略对初始划分不断更新，产生新的权重和聚类划分
- 为了降低多次迭代所需的计算复杂度，通常使用采样技术以提高性能
- 经典算法
  - PROCLUS算法（维规约子空间聚类算法）

PROCLUS算法

- PROCLUS算法基本思想
  - 首先选取整个样本集的小部分数据作为初始样本，
  - 再从中选择c个聚类中心通过迭代策略对数据簇的质量进行改进。
  - 其执行过程分为三个阶段
    - 初始化阶段
    - 迭代阶段
    - 改进阶段

PROCLUS算法

- 一、初始化阶段
  - 对整个数据集进行随机抽样，
  - 利用贪心策略得到一个潜在中心电极和的超级M，
  - 并且保证每个数据簇至少包含一个样本点在这个超集中；

PROCLUS算法

★二、迭代阶段

- 从超集M中随机选择C个聚类中心，将随机抽取到的新中心替代当前集合中不好的样本点，直到获得更优的中心点集。
- 按照上述过程反复迭代，直到所得的聚类中心点的集合达到稳定。
- 同时，以各个子空间包含的样本点到其对应聚类中心的平均距离作为该数据簇的半径，找到各个数据簇对应的特征子集。

PROCLUS算法

- 三、改进阶段
  - 对每个数据簇的聚类中心再次进行扫描以确定其对应的特征子集，
  - 并在该特征子集上计算样本点到聚类中心的曼哈顿距离，进行新的划分，同时去除孤立点。

PROCLUS算法的特点

★ 优点

- PROCLUS算法适合发现超球面形状的数据簇
- 使用了采样技术，因此在聚类速度方面要明显优于CLIQUE算法

★ 缺点

- 在聚类过程中需要确定三个参数：簇的数量、簇的平均维数和最小偏差，因此对数据集的参数设置比较敏感

第四章 聚类方法

- ★ 什么是聚类
- ★ 层次聚类方法
- ★ 划分聚类方法
- ★ 基于密度的聚类方法
- ★ 基于子空间的聚类方法
- ★ 人工神经网络聚类方法

自组织特征映射神经网络聚类算法（SOM）

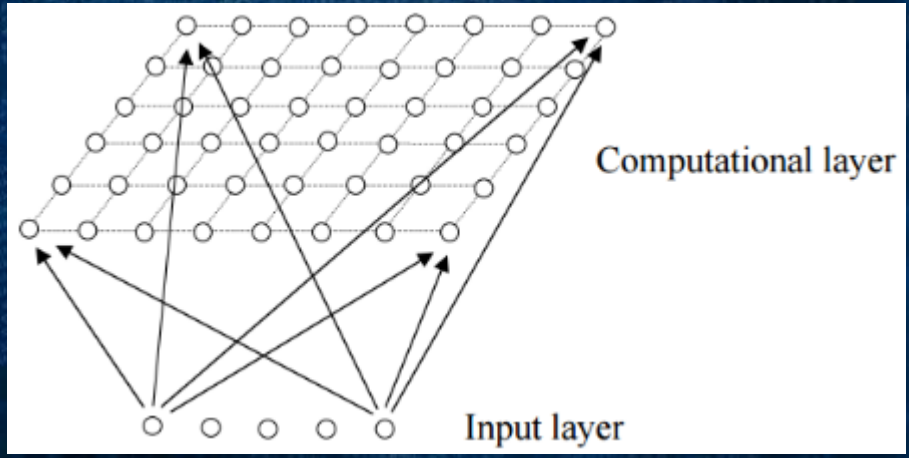
★ Self Organizing Map Network

★ 自组织特征映射神经网络

- 1981年由芬兰学者Kohonen最早提出
- 是无监督学习网络。它可以模仿人脑神经元的相关属性，通过训练自动对输入模式进行聚类
- 有输入层和输出层（竞争层）两层拓扑结构
- 输入结点和输出神经元全连接；输入结点的数目等同输入向量的维数，同时等同连接权向量的维数
- 目的：把高维空间的输入数据映射到低维（通常是一维或二维）的神经元网格上，并保持原来的拓扑次序

SOM工作原理

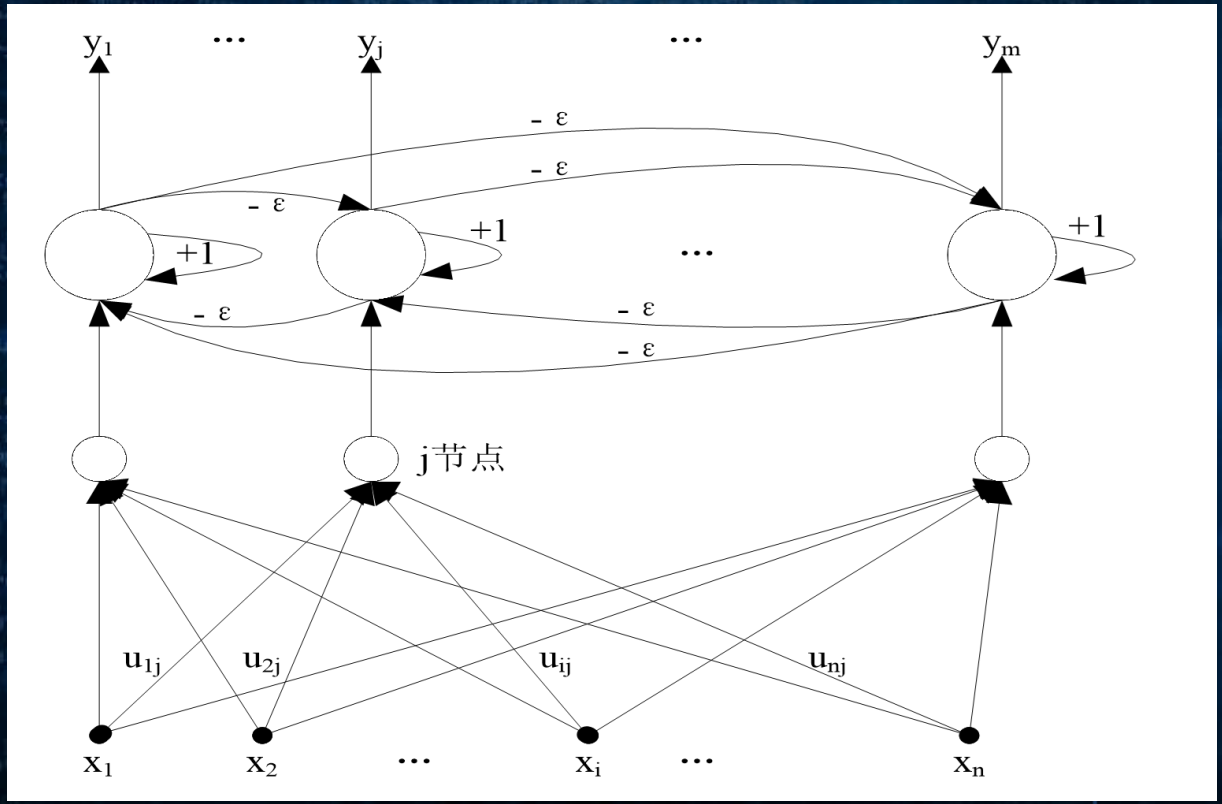
- ★ 输入层：
  - 接收外界信息，将输入模式向竞争层传递，起“观察”作用
- ★ 竞争层：
  - 负责对输入模式进行“分析比较”，寻找规律并归类
- ★ 输入结点和竞争层神经元全连接



SOM工作原理

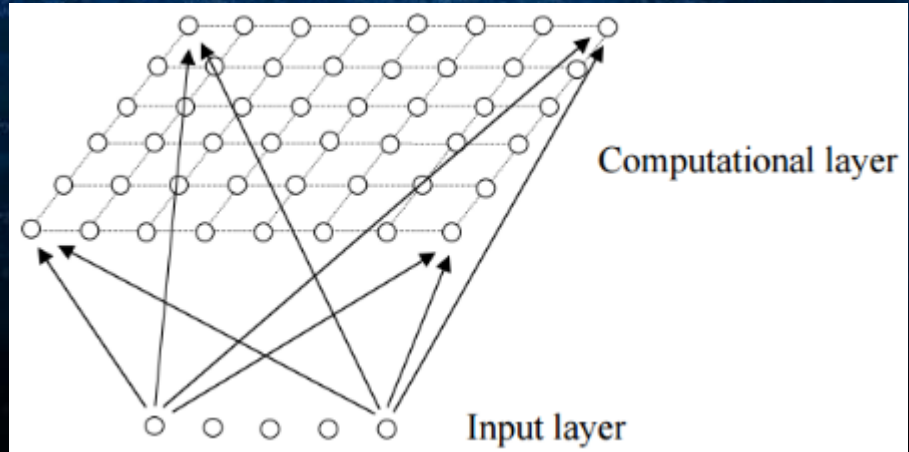
★ 竞争网络

- 层内有互连的网络
- 同层神经元之间有横向联系。所以同层神经元之间有相互作用，可以形成竞争。



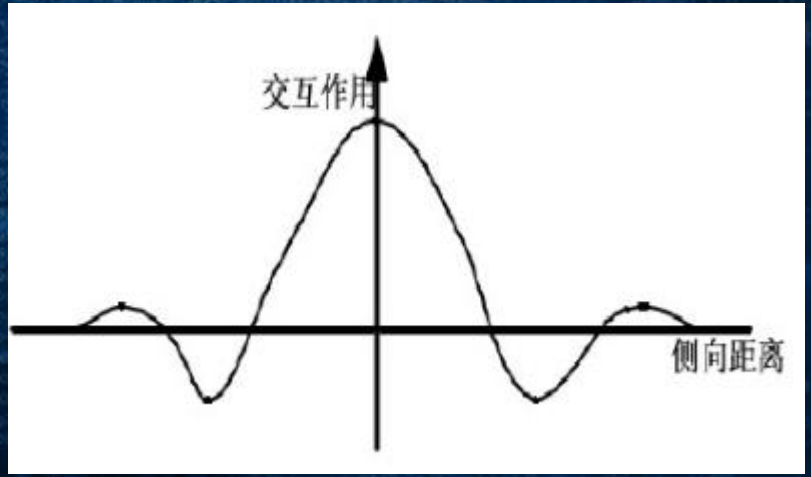
SOM工作原理

- ★ SOM是竞争式学习网络
  - 每当一个向量被提交，具有最近权值向量的那个神经元将竞争获胜。
  - 获胜神经元及其邻域内的神经元将移动它们的权值向量从而离输入向量更近一些。
- ★ 权值向量有两个趋势
  - 首先，它们随着更多的输入向量被提交而分布到整个输入空间。
  - 其次，它们移向邻域内的神经元。两个趋势共同作用使神经元在那一层重新排列，最终输入空间得到分类。



SOM工作原理

- 在竞争层中，对于获胜的神经元g，在其周围Ng的区域内，神经元在不同程度上都得到了兴奋，而在Ng区域以外的神经元都得到了抑制，即“以获胜神经元为圆心，对近邻的神经元表现出兴奋性侧反馈，而对远邻的神经元表现出抑制性侧反馈。近邻者相互激励，远邻者相互抑制”。
- 整体上表现出中间强度大，两边逐渐衰减，而远离中心的收到抑制的趋势。



SOM基本流程

★ 1. 初始化

对输出层各权向量赋予较小的随机数并进行归一化处理，得到$\hat{W}_j(j = 1,2,\ldots,m)$，建立初始优胜邻域$N_j^*(0)$和学习率η初值。m为输出层神经元数目。

★ 2. 接收输入

从训练集中随机取一输入模式并进行归一化处理，得到$\hat{X}^p(p = 1,2,\ldots,n)$，$n$为输入层神经元数目。

SOM基本流程

★ 3. 寻找获胜节点

计算$\hat{W}_j$和$\hat{X}^p$的点积，从中找到点积最大的获胜节点 j*。如果输入模式未经归一化，应按下式计算欧式距离，从中找出距离最小的获胜节点

$$d_j = \|\hat{X} - \hat{W}_j\| = \sqrt{\sum_{j=1}^{m}[X - \hat{W}_j]^2}$$

★ 4. 定义优胜邻域$N_j^*(t)$

设 j*为中心确定 t 时刻的权值调整域，一般初始邻域$N_j^*(0)$较大，训练过程中$N_j^*(t)$随训练时间收缩。

SOM基本流程

★ 5. 调整权值

对优胜邻域$N_j^*(t)$内的所有节点调整权值

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(t,N)\left[x_i^p - w_{ij}(t)\right]$$
$$i = 1,2,\ldots,n \quad j \in N_j^*(t)$$

式中$w_{ij}(t)$是神经元 i 在 j时刻的权值

$\alpha(t,N)$是训练时间和邻域内第 i 个神经元与获胜神经元 j*之间的拓扑距离N的函数

★ 6. 结束判定

当学习率$\alpha(t) \le \alpha_{min}$时，结束训练；否则转到步骤（2）继续。

SOM的特点

★ 优点

- 网络结构简单，具有很好的生物神经元特征
- 容错性强
- 具有特征映射的能力
- 结果可视化
- 网络具有自稳定性
- 具有自联想性

★ 缺点

- 用户必须预先指定聚类数目和初始的权值矩阵
- 一个SOM簇通常并不对应单个自然簇，可能有自然簇的合并和分裂
- 缺乏具体的目标函数
- 不保证收敛，尽管它通常收敛
- 可能会有一些神经元始终不能获胜和一些被过度利用的神经元，不能充分利用所有神经元信息将影响聚类质量
- 数据输入顺序会影响甚至决定输出的结果

感谢聆听
欢迎提问