无监督学习

■监督学习与无监督学习

- ◆**监督学习**：发现数据属性和类别属性之间的关联模式。并通过利用
  - ●这些模式用来预测未知数据实例的类别属性。
- ◆**无监督学习**：数据没有目标属性。
  - ●发现数据中存在的内在结构。

聚类

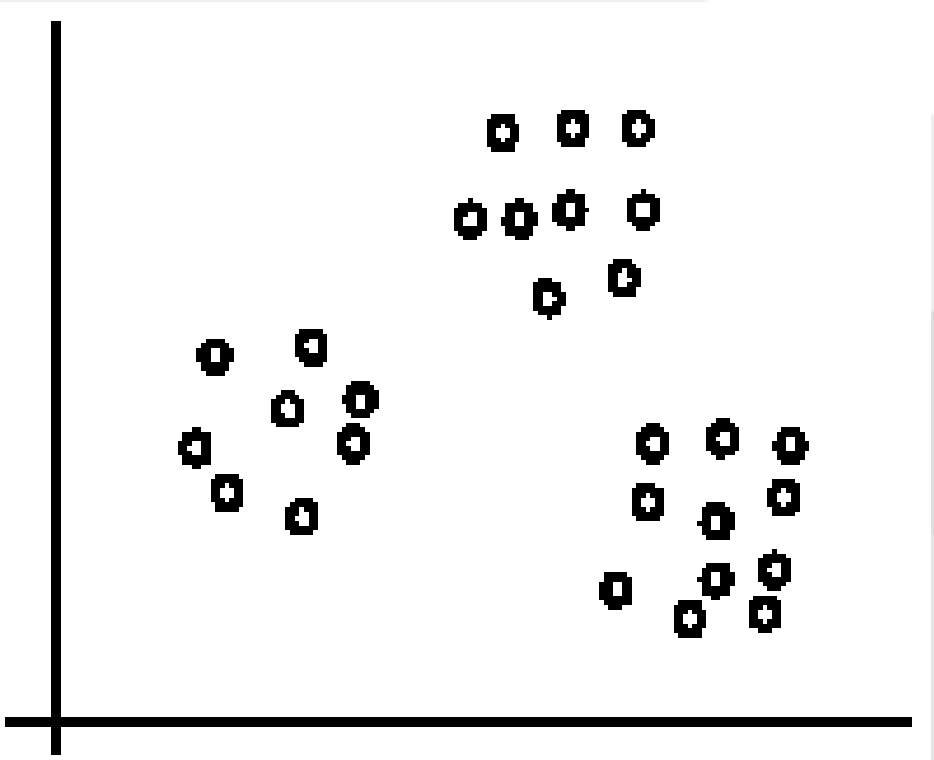
- **聚类（**Clustering**）**是一种发现数据中的相似群（聚类，clusters）的技术。
  - ◆处于相同聚类的数据实例彼此相似，处于不同聚类中的实例则彼此不同。
- 聚类通常被称为无监督学习。在聚类中那些表示数据类别的分类或分组信息没有事先给出。
- 由于历史的原因，聚类和无监督学习的关系更加紧密，甚至被认为是同义词。
  - ◆事实上，关联规则挖掘也是无监督学习。
- 本章主要介绍聚类算法。

1.1 基本概念

- **聚类**是一个将数据集中在某些方面相似的数据成员进行分类组织的过程。
- 一个聚类就是一些数据实例的集合
  - 这个集合中的元素彼此相似；
  - 与其他聚类中的元素不同。
- 聚类的相关文献中，一个数据实例有时被称作**对象**——数据实例很可能代表现实世界中的一个对象。
- 有时也被称作**数据点**——数据实例可以被看作是r维空间中的一个点，其中r表示数据的属性个数。

引例

- 如下图所示，一个二维数据集，由三组数据点（聚类）组成。



■聚类的目的

◆来自不同应用领域的真实实例。

◆**实例1：**根据身材把人分组的方法通常就采用聚类。T恤分“小号”、“中号”、“大号”。

- ●为每个顾客量身定做：太贵
- ●仅一种型号的T恤：大多数人不合身。

◆**实例2：**在营销学中，对客户进行分割，为每组客户指定一个套营销策略，也是采用聚类来完成。

- 实例3：对给定文本，需要根据它们内容的相似性来进行组织。
  - 建立一个主题层次。
- 事实上，聚类是数据挖掘技术中应用最广泛的技术之一。
  - 发展历史长，应用领域广泛。比如：医学类、心理学、植物学、社会学、生物学、营销学、保险、图书馆等。
  - 近年来，在线文档的快速发展，文本聚类研究成为关注的重点。

聚类的概述

- **聚类算法**
  - 划分聚类
  - 层次聚类
  - …（密度聚类）
- **距离函数**（相似性或相异性）：度量两个数据点（对象）的相似程度。
- **聚类评价**
  - 类内差异（聚类内部距离）：最小化
  - 类间差异（聚类外部距离）：最大化
- 聚类结果的质量与**算法**、**距离函数**和**应用领域**有很大关系。

1.2 k-均值聚类

- k-均值算法是**划分聚类算法**。
- k-均值算法根据**某个距离函数**反复地把数据分入k个聚类中。
- 设数据点（或实例）的集合D为$\{x_1, x_2, \ldots, x_n\}$，其中，$x_i = (x_{i1}, x_{i2}, \ldots, x_{ir})$是实数空间$X \subseteq R^r$中的向量。并且r表示数据的属性数目（数据空间维数）。
- k-均值算法把给定的数据划分为k个聚类。
  - 每个聚类中有一个聚类的中心（也称聚类中心），它用来表示某个聚类，这个中心是聚类中所有数据点的**均值**。
  - K是由用户指定的。

■ k-均值算法

◆给定k，k-均值算法执行步骤：

- ●随机选取k个数据点作为**初始聚类中心**。
- ●计算每个数据点与各个中心之间的距离，把每个**数据点分配**给距离它最近的聚类中心。
- ●数据点分配以后，每个聚类的聚类中心会根据聚类现有的数据点**重新计算**。
- ●这个过程将不断重复直道满足某个终止条件为止。

■ 算法内容

Algorithm *k*-means(*k*, *D*)

1 Choose *k* data points as the initial centroids (cluster centers)
2 **repeat**
3 **for** each data point $\mathbf{x} \in D$ do
4 compute the distance from **x** to each centroid;
5 assign **x** to the closest centroid // a centroid represents a cluster
6 **endfor**
7 re-compute the centroids using the current cluster memberships
8 **until** the stopping criterion is met

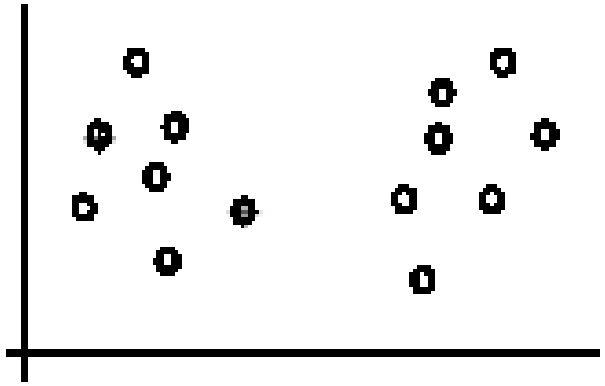
终止条件

- 没有（或最小数目）数据点被重新分配给不同的聚类。
- 没有（或最小数目）聚类中心再发生变化。
- 误差平方和（sum of squared error，SSE）局部最小。

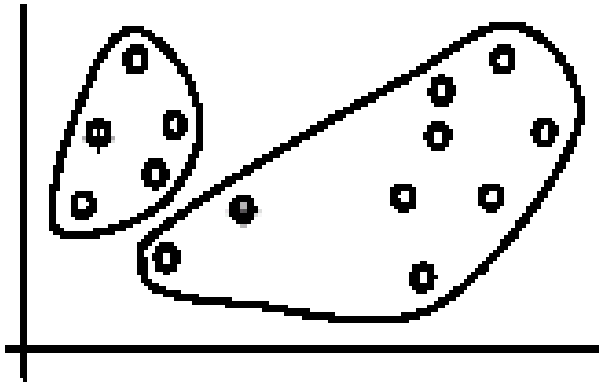
$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2 \qquad (1)$$

其中，C_i表示第j个聚类，$\mathbf{m}_j$是聚类C_j的聚类中心（C_j中所有数据点的均值向量），$dist(\mathbf{x}, \mathbf{m}_j)$表示数据点x和聚类中心$\mathbf{m}_j$之间的距离。

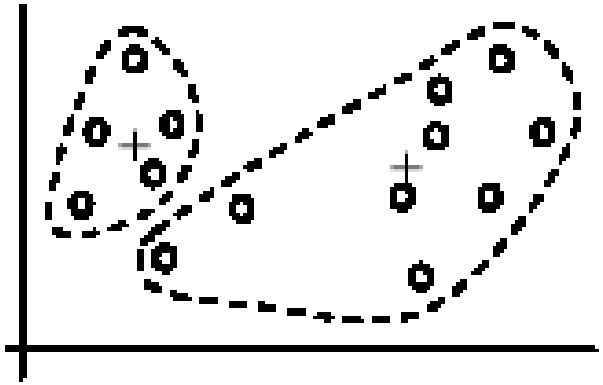
■ 举例



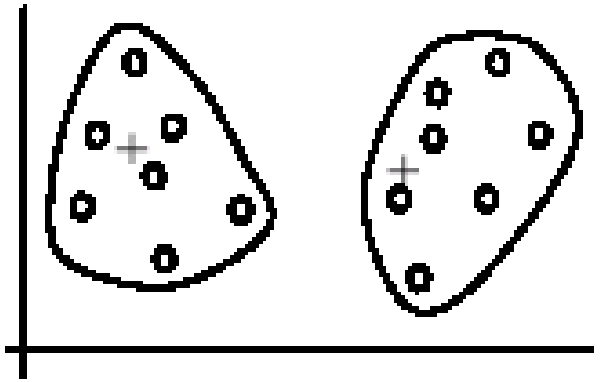
(A). Random selection of k centers



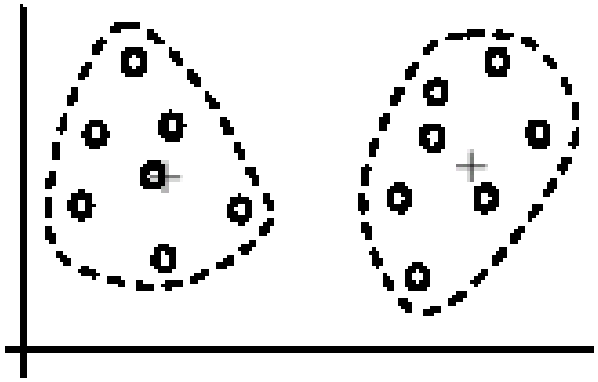
Iteration 1: (B). Cluster assignment



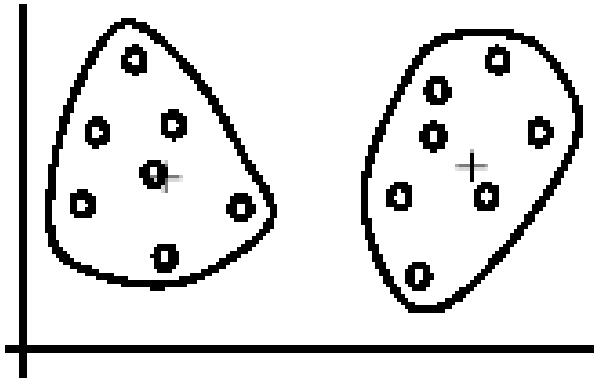
(C). Re-compute centroids



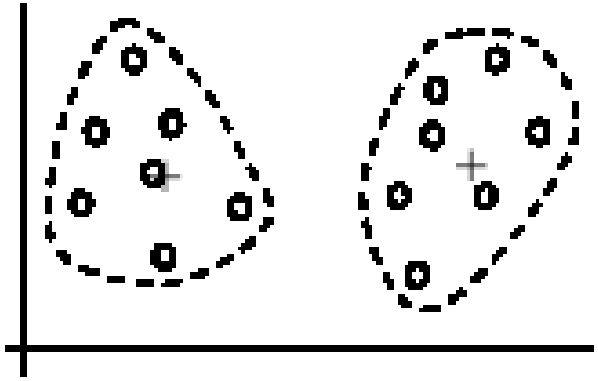
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids



Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

距离计算

- 在那些均值能被定义和计算的数据集上均能使用k-均值算法。
- 在**欧式空间**，聚类均值可以使用如下公式：

$$\mathbf{m}_j = \frac{1}{|C_j|}\sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \qquad (2)$$

- 数据点与聚类中心的距离使用如下公式：

$$dist(\mathbf{x}_i, \mathbf{m}_j) = \| \mathbf{x}_i - \mathbf{m}_j \| \qquad (3)$$

$$= \sqrt{(x_{i1} - m_{j1})^2 + (x_{i2} - m_{j2})^2 + \ldots + (x_{ir} - m_{jr})^2}$$

算法举例：

下面给出一个样本事务数据库，并对它实施k-平均算法。

序号	属性1	属性2
1	1	1
2	2	1
3	1	2
4	2	2
5	4	3
6	5	3
7	4	4
8	5	4

设n=8，k=2，执行下面的步骤：

第一次迭代：假定随机选择的两个对象，如序号1和序号3当作初始点，分别找到离两点最近的对象，并产生两个簇{1，2}和{3，4，5，6，7，8}

对于产生的簇分别计算平均值，得到平均值点。

对于{1，2}，平均值点（1.5,1）；

对于{3，4，5，6，7，8}，平均值点（3.5,3）

第二次迭代:

通过平均值调整对象所在的簇，重新聚类，即将所有点按离平均值点（1.5,1）和（3.5,3）最近的原则重新分配。得到两个簇:

{1,2,3,4}和{5,6,7,8}

重新计算簇平均值点，得到新的平均值点为:

（1.5,1.5）和（4.5,3.5）

第三次迭代:

通过平均值调整对象所在的簇，重新聚类，即将所有点按离平均值点1.5,1.5）和（4.5,3.5）最近的原则重新分配。得到两个簇:

{1,2,3,4}和{5,6,7,8}

发现没有出现重新分配，准则函数收敛，程序结束。

下图给出了该例子整个过程中平均值计算和簇生成的过程和结果。

迭代次数	平均值 簇1	平均值 簇2	产生的新簇	新平均值 簇1	新平均值 簇2
1	(1,1)	(1,2)	{1,2}, {3,4,5,6,7,8}	(1.5,1)	(3.5,3)
2	(1.5,1)	(3.5,3)	{1,2,3,4}, {5,6,7,8}	(1.5,1.5)	(4.5,3.5)
3	(1.5,1.5)	(4.5,3.5)	{1,2,3,4}, {5,6,7,8}	(1.5,1.5)	(4.5,3.5)

1.2.2 k-均值算法的硬盘版本

- **k-均值可以执行硬盘上的数据**
  - 每一次循环，算法扫描数据一次
  - 聚类中心可以在每次循环中增量计算。
- **k-均值在处理大规模数据时（内存不足以容纳这些数据）十分有用。**
- **需要控制循环次数**
  - 实际应用中，需要设置一个限制次数(<50)

K-均值硬盘版算法

Algorithm disk-k-means(k, D)

1 Choose k data points as the initial centriods $\mathbf{m}_j$, $j = 1, \ldots, k$;
2 **repeat**
3 initialize $\mathbf{s}_j = \mathbf{0}$, $j = 1, \ldots, k$; // $\mathbf{0}$ is a vector with all 0's
4 initialize $n_j = 0$, $j = 1, \ldots, k$; // n_j is the number points in cluster j
5 **for** each data point $\mathbf{x} \in D$ **do**
6 $j = \arg\min_j dist(\mathbf{x}, \mathbf{m}_j)$;
7 assign $\mathbf{x}$ to the cluster j;
8 $\mathbf{s}_j = \mathbf{s}_j + \mathbf{x}$;
9 $n_j = n_j + 1$;
10 **endfor**
11 $\mathbf{m}_i = \mathbf{s}_j / n_j$, $i = 1, \ldots, k$;
12 **until** the stopping criterion is met

1.2.3 优势与劣势

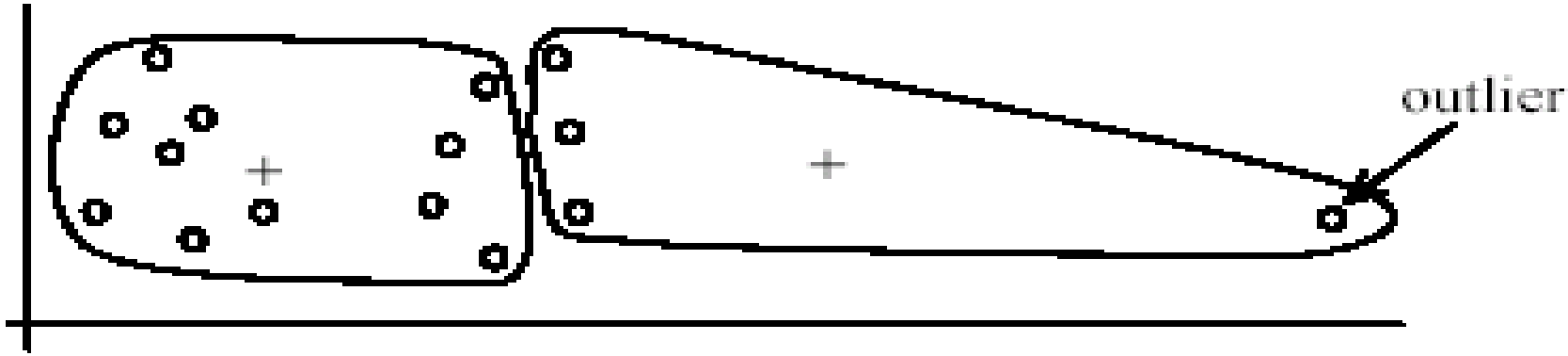
- **优势**
  - **简单**：容易被理解，同时也容易被实现。
  - **效率**：时间复杂度为$O(tkn)$。

    其中，n是数据点个数；k是聚类的个数；t是循环次数
  - 由于k和t通常都远远小于n，k-均值算法被认为相对于数据点的数目来说是线性的。
- K-均值算法是聚类算法中最流行的一种算法。

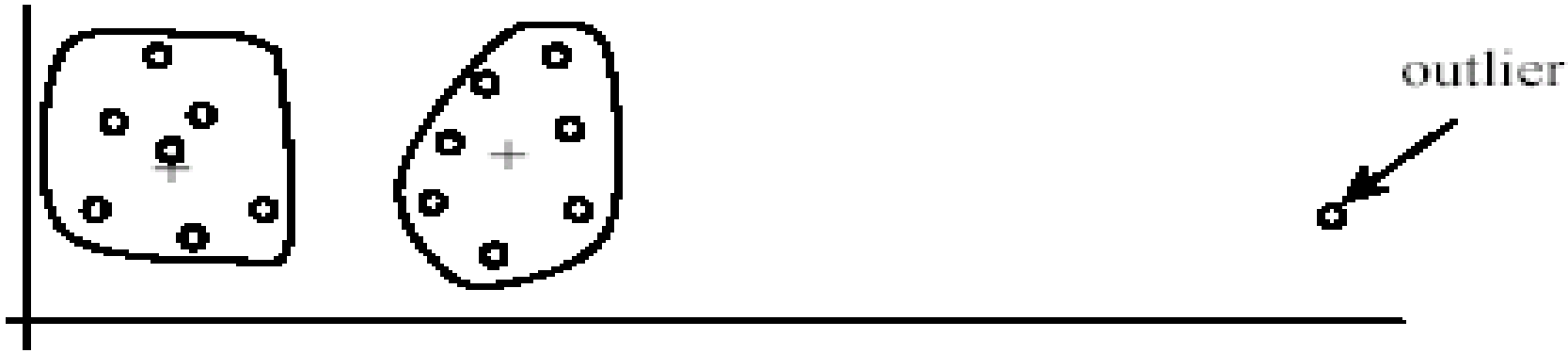
■ 劣势

- ◆算法只能用于那些均值能够被定义的数据集上。
  - ●对于**范畴数据**，有一种k-均值算法的变体——**k-模算法**，用于聚类范畴数据。
- ◆用户需要事先指定聚类数目**k**。
- ◆算法对于**异常值**十分敏感。
  - ●异常值是指数据中那些与其他数据点相隔很远的数据点。
  - ●异常值可能是数据采集时产生的错误或者一些具有不同值的特殊点。

异常值问题



(A): Undesirable clusters



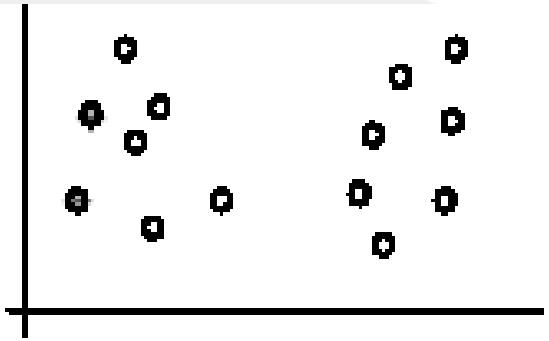
(B): Ideal clusters

■解决异常值问题

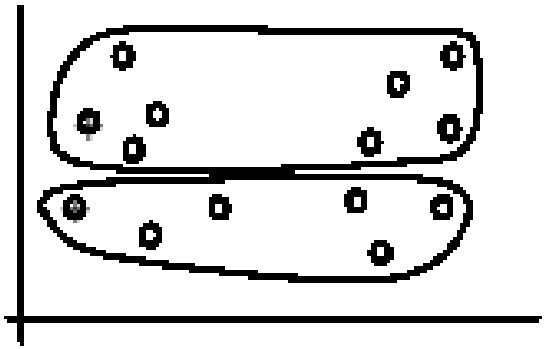
- **一种方法：**在聚类过程中去除那些比其他任何数据点离聚类中心都要远的数据点。
  - 安全起见，我们需要在多次循环中监控这些潜在的异常值，随后再决定是否删除它们。
- **另一个方法：**随机采样。因为在采样过程中，我们仅仅只选择很少一部分的数据点，因此选中异常值的概率将会很小。
  - 我们可以先用采样点进行预先聚类，然后把其他数据点分配给这些聚类（剩下的数据点分配给那些距离它最近的聚类中心/分类）。

■劣势（续）

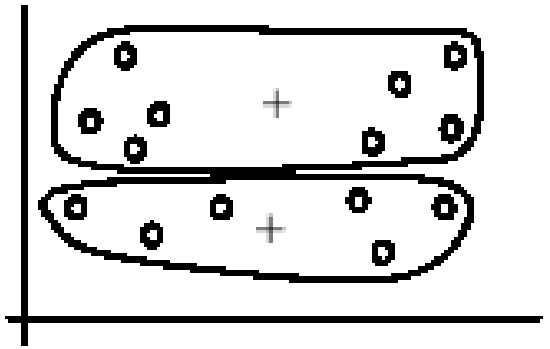
◆算法对于**初始种子**十分敏感。



(A). Random selection of seeds (centroids)

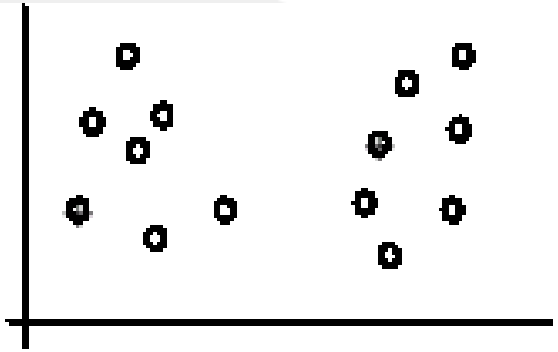


(B). Iteration 1

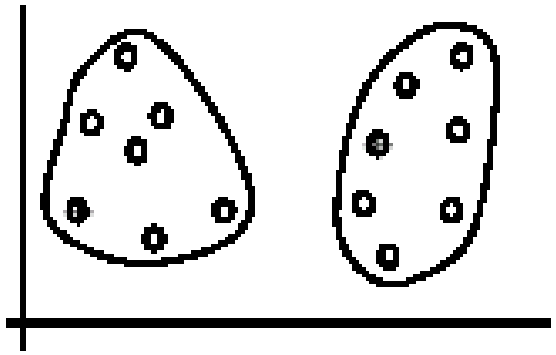


(C). Iteration 2

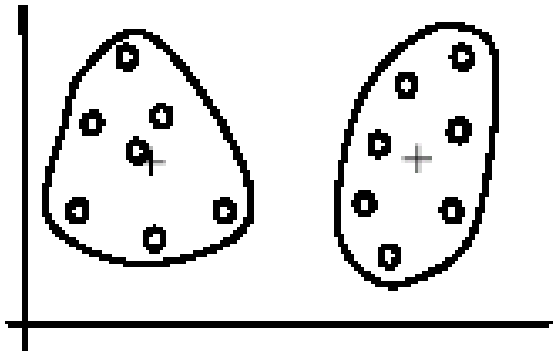
◆如果选择不同的种子：比较好的结果。



(A). Random selection of k seeds (centroids)



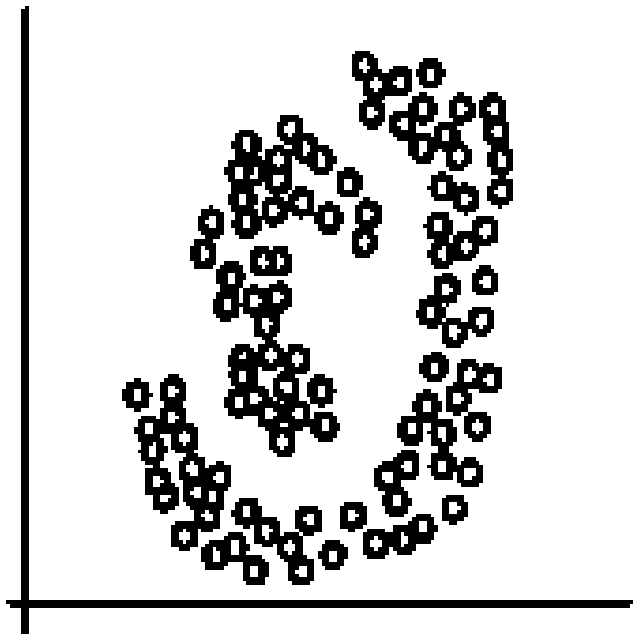
(B). Iteration 1



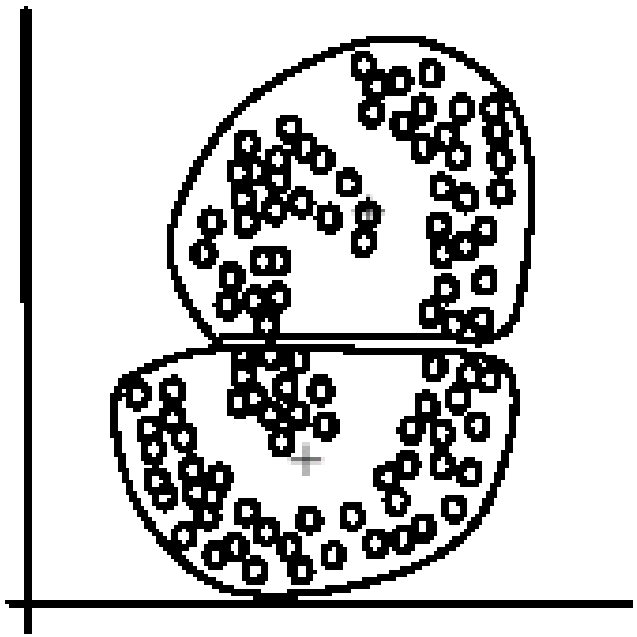
(C). Iteration 2

■ 劣势（续）

◆算法不适合发现那些形状**不是超维椭圆体（或超维球体）**的聚类。



(A): Two natural clusters



(B): *k*-means clusters

K-均值算法总结

- 虽然k-均值算法有不足，但它仍是在实践中采用最为广泛的算法。
  - 其他聚类算法也有它们的一系列不足之处。
- 没有直接的证据证明哪一种算法在整体表现上优于k-均值算法。
  - 虽然其他聚类算法在某些特殊数据下的表现优于k-均值算法。
- 比较不同聚类算法的优劣是一个很难的任务。没有人知道正确的聚类结果是什么。

1.3 聚类的表示

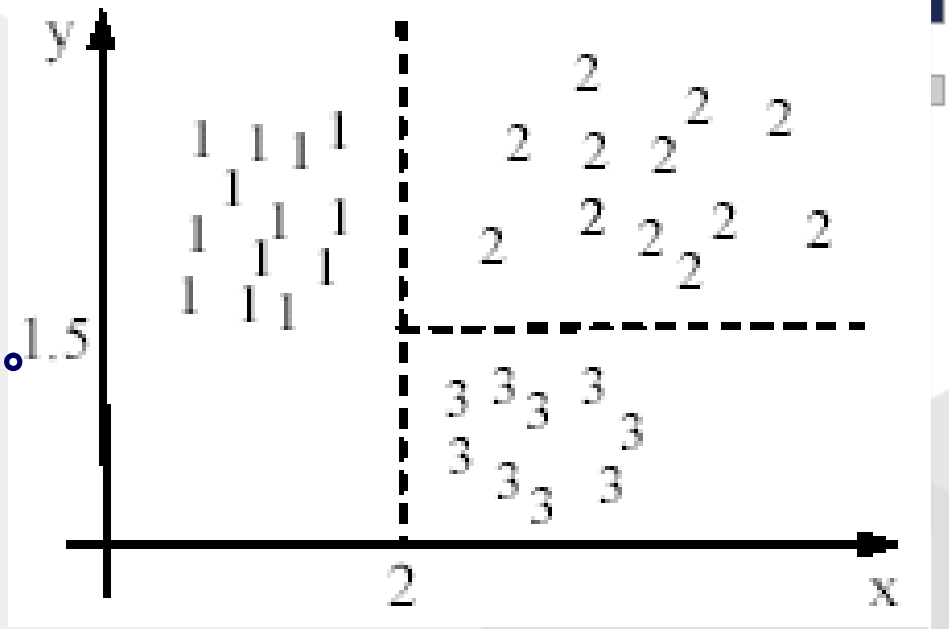
- 对于数据中的聚类需要寻找一种方法来表示这些聚类。
- 某些应用中，只需直接输出聚类中的数据点即可。
- 而有的应用，特别是决策相关应用中，聚类结果需要用一种压缩和可理解的方式表示。
- 有利于对聚类结果的评价。

1.3.1 聚类的一般表示方法

- **用聚类中心来表示每个聚类是使用最广泛的聚类表示方法**
  - 计算聚类的半径和标准差来确定聚类在各个维上的伸展度。
  - 聚类中心表示法对于那些高维球体形状的聚类来说已经足够。
  - 但如果聚类被拉长了或者是其他形状的话，聚类中心表示就可能不太适合。

利用分类模型来表示聚类

- 同一聚类中的所有数据点被看作是具有相同类别标识的，如聚类标识。
  - 通过在数据上执行某个监督学习算法来发现分类模型。



$x \le 2 \rightarrow$ cluster 1

$x > 2, y > 1.5 \rightarrow$ cluster 2

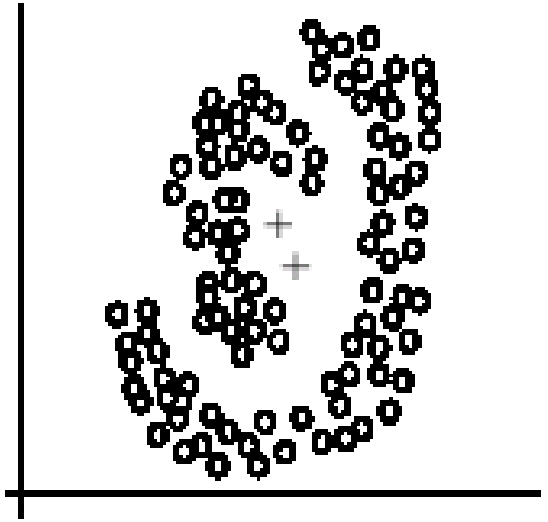
$x > 2, y \le 1.5 \rightarrow$ cluster 3

■利用聚类中最为常见的值来表示聚类

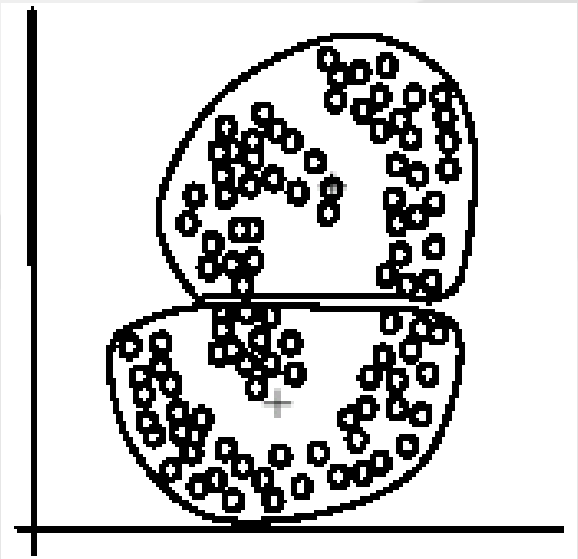
- ◆这种方法通常在对**范畴属性**进行聚类时采用（k-模聚类）。
- ◆它同样是文本聚类中的重要方法，比如：使用一个较小的集合：每个类中高频词来表示这个类。

1.3.2 任意形状的聚类

- 超维椭圆体或超维球体形状的聚类容易使用聚类中心以及聚类的伸展度（标准差）、规则或者它们的组合来表示。

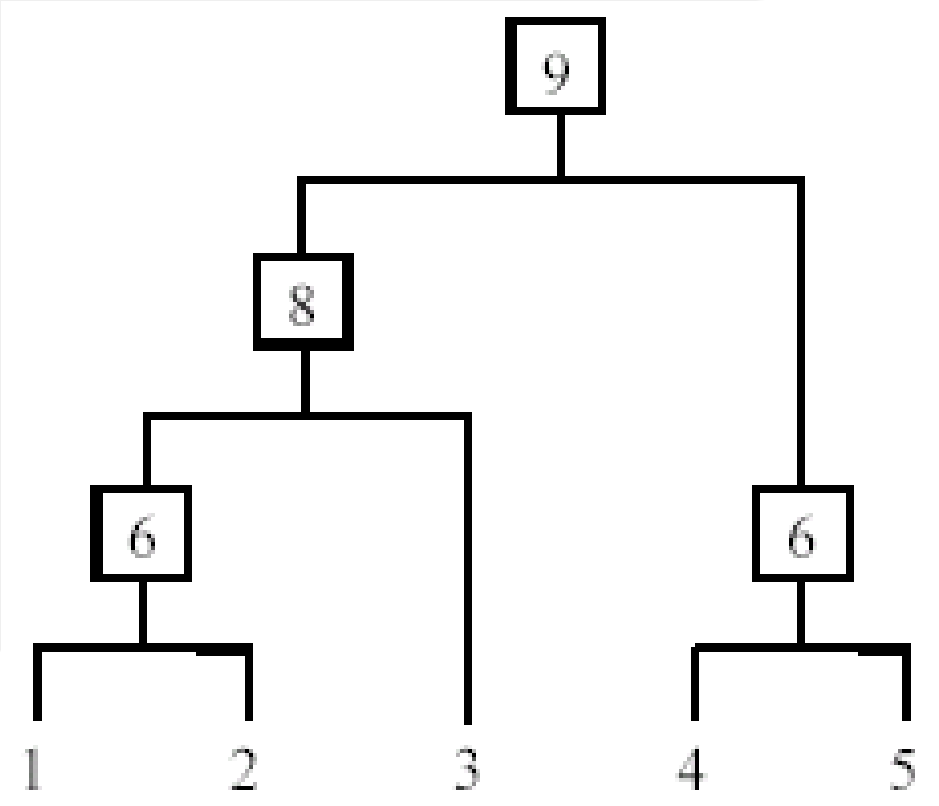


- **任意形状的聚类**是很难用它们来表示的。（一般分别输出每个聚类中的数据点）



1.4 层次聚类

- 生成一系列嵌套的聚类树（也叫树状图）



■层次聚类的两种方法

- ◆**合并（自下而上）聚类**：从树状图的最底层开始构建聚类树。
  - 合并最相似（距离最近）的聚类来形成上一层中的聚类。
  - 整个过程当全部数据点都合并到一个聚类中时停止。
- ◆**分裂（自上而下）聚类**：从一个包含全部数据点的聚类开始。
  - 根节点聚类分裂成一些子聚类。每个子聚类在递归地继续向下分裂。
  - 直到出现只包含一个数据点的单节点聚类出现时停止。

■合并聚类

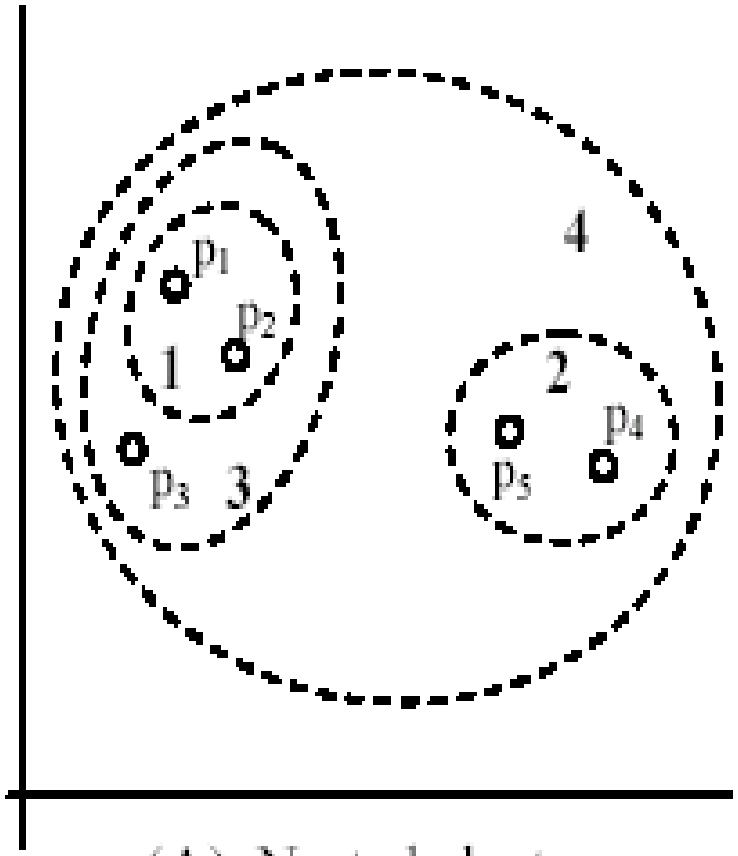
- ◆它比分裂聚类算法应用得更广泛。
  - ●起初，每一个数据点形成一个聚类（或叫节点）
  - ●合并具有最短距离的聚类/节点
  - ●继续合并
  - ●直到所有节点都在一个聚类中为止。

■合并聚类算法

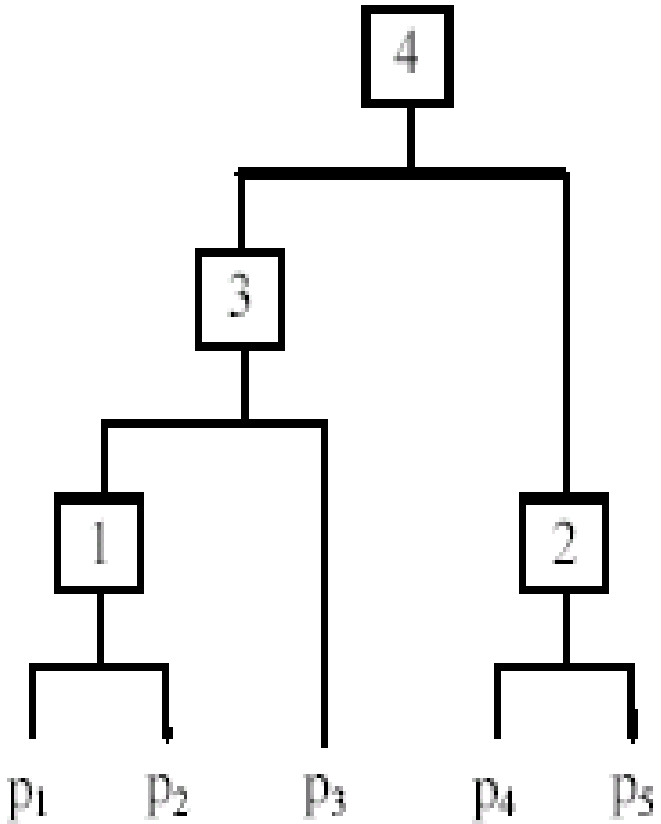
Algorithm Agglomerative(D)

1 Make each data point in the data set D a cluster,
2 Compute all pair-wise distances of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in D$;
2 **repeat**
3 find two clusters that are nearest to each other;
4 merge the two clusters form a new cluster c;
5 compute the distance from c to all other clusters;
12 **until** there is only one cluster left

■算法举例



(A). Nested clusters



(B) Dendrogram

算法举例：

下面给出一个样本事务数据库，并对它实施**合并聚类**算法。

序号	属性1	属性2
1	1	1
2	1	2
3	2	1
4	2	2
5	3	4
6	3	5
7	4	4
8	4	5

设n=8，用户输入的终止条件为两个簇，执行下面的步骤：

初始簇：{1}，{2}，{3}，{4}，{5}，{6}，{7}，{8}

第一步：根据初始簇计算每个簇之间的距离，随机找出距离最小的两个簇，进行合并，最小距离为1，合并后，1，2点合并为一个簇。

第二步：对第一步合并后的簇计算簇间距离，找出距离最近的两个簇进行合并（**合并聚类算法中计算两个簇间的相似度可由这个两不同簇距离最近的数据点对的相似度来确定**）。经计算，选择最近的簇距离1，确定{3}{4}合并为一簇。

第三步：按第二步的操作方式，对{1,2}{3,4}{5}{6}{7}{8}计算6个簇间距离，找出距离最近的两个簇进行合并。经计算，选择最近的簇距离1，确定{5}{6}合并为一簇。

第四步：同样按第二步的操作，{7}{8}合并为一簇，得到{1,2}{3,4}{5,6}{7,8}

第五步：分别计算簇之间的距离，合并{1,2}{3,4}为一簇（两簇中，{1}{3}的距离为1，即可合并）

第六步：分别计算簇{1,2,3,4}{5,6}{7,8}的距离，合并{5,6}{7,8}为一簇，得到{1,2,3,4}{5,6,7,8}两个簇，由于数目已经达到了用户输入的条件，程序结束。

下图给出了该例子整个过程中簇间距离计算和簇合并的过程和结果。

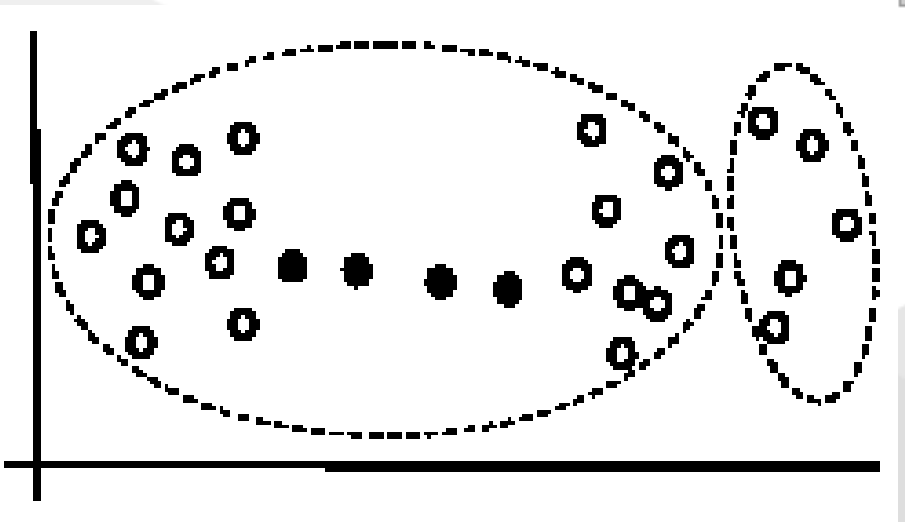
步骤	最近的簇距离	最近的两个簇	合并后的新簇
1	1	{1} {2}	{1,2} {3} {4} {5} {6} {7} {8}
2	1	{3} {4}	{1,2} {3,4} {5} {6} {7} {8}
3	1	{5} {6}	{1,2} {3,4} {5,6} {7} {8}
4	1	{7} {8}	{1,2} {3,4} {5,6} {7,8}
5	1	{1,2} {3,4}	{1,2,3,4} {5,6} {7,8}
6	1	{5,6} {7,8}	{1,2,3,4} {5,6,7,8}

■两个聚类之间的距离计算

- **◆层次聚类中计算两个聚类之间的距离方法有很多。**
- ◆在不同的算法中可以用不同方法去确定两个聚类之间的距离。
  - ●单链接方法
  - ●全链接方法
  - ●平均链接方法
  - ●聚类中心方法
  - ●…

1.4.1 单链接方法

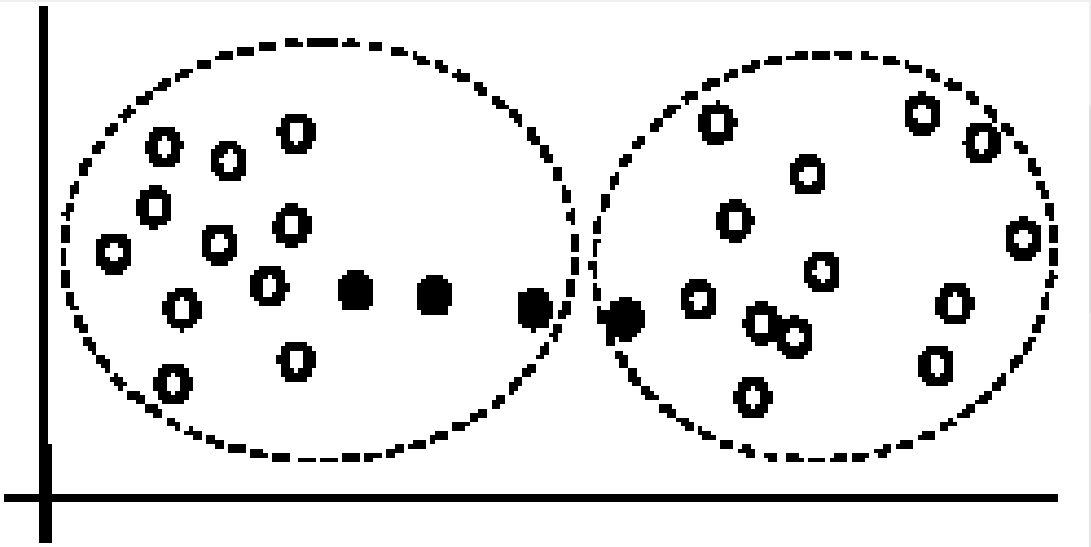
- 两个聚类之间的距离是两个聚类**中距离最近的两个数据点**（来自不同聚类的数据点）之间的距离。
- 单链接方法适合于寻找那些形状怪异的聚类。
  - 但是它对数据中的噪声非常敏感。



链接两个自然聚类的噪音数据点的存在使得其中一个自然聚类被分裂

1.4.2 全链接方法

- 两个聚类之间的距离是两个聚类中所有数据点之间聚类的最大值。
- 全链接方法对异常值十分敏感。



1.4.3 平均链接方法

- 平均链接方法：一种折衷方法。
  - 介于全链接方法对于异常值的敏感性和单链接方法形成长链（这种长链不符合聚类是紧密地椭圆体对象这一常识）的趋势之间的折衷方法。
  - 两个聚类之间的距离是两个聚类之中多个数据点对之间距离之和的平均值。
- 聚类中心方法：两个聚类之间的距离是两个聚类中心之间的距离。

■ 复杂度

- ◆层次聚类算法中至少有$O(n^2)$的计算复杂度。N为数据点的个数。
- ◆单链接的计算复杂度为$O(n^2)$。
- ◆全链接和平均链接算法的复杂度达到了$O(n^2 \log n)$。
- ◆层次聚类算法在处理大规模数据时十分低效。
  - ● 采样
  - ● 向大规模数据扩展的方法（比如：BIRCH）

1.5 距离函数

- 距离或者相似度函数在聚类算法中扮演者十分重要的角色。
- 有许多不同的距离函数用于
  - 不同数据类型
    - 数值型数据
    - 符号数据
  - 不同的应用领域

1.5.1 数值的属性

- 最常用的距离函数是**欧几里德距离**（Euclidean distance）和**曼哈顿距离**（Manhattan distance）。
- 用$dist(\mathbf{x}_i, \mathbf{x}_j)$来表示r维空间中的两个数据点之间的距离。
- 这两种函数是**闵可夫斯基距离**（Minkowski distance）的特殊情况。h表示一个正整数

$$dist(\mathbf{x}_i, \mathbf{x}_j) = ((x_{i1} - x_{j1})^h + (x_{i2} - x_{j2})^h + \ldots + (x_{ir} - x_{jr})^h)^{\frac{1}{h}}$$

■欧几里德距离和曼哈顿距离

◆如果*h* = 2, 就得到了**欧几里德距离**Euclidean distance

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ir} - x_{jr})^2}$$

◆如果 *h* = 1, 就得到了**曼哈顿距离**Manhattan distance

$$dist(\mathbf{x}_i, \mathbf{x}_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{ir} - x_{jr}|$$

◆加权欧几里德距离

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \ldots + w_r(x_{ir} - x_{jr})^2}$$

■平方欧几里德距离和切比雪夫距离

◆平方欧几里德距离Euclidean distance：标准欧几里德距离平方，加大了距离较远的数据点的权重。

$$dist(\mathbf{x}_i, \mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ir} - x_{jr})^2$$

◆切比雪夫距离Chebychev distance：当需要定义当两个数据点有一个属性值不同时它们就是“不同”的，就用该距离公式。

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \ldots, |x_{ir} - x_{jr}|)$$

1.5.2 布尔属性和符号属性

- **布尔属性**：具有两个状态或值。比如：性别中只有男和女。
- 使用**混合矩阵**来介绍这种距离度量方法。
- 给定第i个和第j个数据点，即$\mathbf{x}_i$ 和 $\mathbf{x}_j$ 。

■ 混合矩阵

数据点x_j

数据点x_i

	1	0	
1	a	b	$a+b$
0	c	d	$c+d$
	$a+c$	$b+d$	$a+b+c+d$

a：x_i，x_j两个数据点中，具有相同属性值1的属性个数

b：x_i，x_j两个数据点中满足$x_{if}=1$且$x_{jf}=0$条件的属性个数，其中x_{if}（x_{jf}）分别表示x_i，x_j两个数据点的第f个属性值。

c：x_i，x_j两个数据点中满足$x_{if}=0$且$x_{jf}=1$条件的属性个数，其中x_{if}（x_{jf}）分别表示x_i，x_j两个数据点的第f个属性值。

d：x_i，x_j两个数据点中具有相同属性值0的属性个数。

■ 对称布尔属性

- ◆ 当布尔属性的两个状态具有相同的重要性并且具有相同的权重时，我们称这个布尔属性是**对称**的。如：性别属性，男和女。
- ◆对称属性中使用普遍的距离函数**简单匹配距离**（Simple Matching Coefficient），是指属性中值不匹配的属性的比例。

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b+c}{a+b+c+d}$$

■举例

$\mathbf{x}_1$	1	1	1	0	1	0	0
$\mathbf{x}_2$	0	1	1	0	0	1	0

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{2+1}{2+2+1+2} = \frac{3}{7} = 0.429$$

■ 非对称布尔属性

- ◆ 当布尔属性的一个状态比其他状态更重要时，我们称这个布尔属性是**非对称**的。通常用1表示那个更为重要的状态，即出现的较少或不太频繁的状态。
- ◆非对称属性中使用最多的距离度量函数是**Jaccard距离**（Jaccard coefficient）。

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b + c}{a + b + c}$$

- ◆我们也可以在距离中进行一些变化，增加权重。

■ 符号属性

- ◆ 具有多于两个状态或值的属性。
  - ●使用最多的距离度量函数也是基于**简单匹配距离**的。
  - ●给定两个数据点$\mathbf{x}_i$和$\mathbf{x}_j$，设数据的属性数目为r，并且$\mathbf{x}_i$和$\mathbf{x}_j$之间属性值匹配的属性数目为q：

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{r - q}{r}$$

1.5.3 文本文档

- 文本文档是由一系列由单词序列组成的句子组成的。
  - 为了简化，在文档聚类中通常把文本文档看作是一组单词的集合（a “bag” of words）。单词的顺序和位置信息被忽略
- 因此，文档可以和普通数据点一样用**向量**来表示。
- 比较两个文档时，通常计算的是两个文档的**相似度**而不是它们的距离。
  - 使用最多的相似度函数是**向量夹角余弦相似度**（**cosine similarity**）。

$$\cos(\theta) = \frac{\sum_{i=1}^{n}(X_i \times Y_i)}{\sqrt{\sum_{i=1}^{n}(X_i)^2} \times \sqrt{\sum_{i=1}^{n}(Y_i)^2}}$$

计算两个句子向量

句子A：(1，1，2，1，1，1，0，0，0)

和句子B：(1，1，1，0，1，1，1，1，1)的向量余弦值来确定两个句子的相似度。

计算过程如下：

$$\cos(\theta) = \frac{1\times1+1\times1+2\times1+1\times0+1\times1+1\times1+0\times1+0\times1+0\times1}{\sqrt{1^2+1^2+2^2+1^2+1^2+1^2+0^2+0^2+0^2} \times \sqrt{1^2+1^2+1^2+0^2+1^2+1^2+1^2+1^2+1^2}}$$

$$= \frac{6}{\sqrt{7}\times\sqrt{8}}$$

$$= 0.81$$

1.6 数据标准化

- 在使用欧几里德距离时，由于数据标准化能使得各个属性在距离计算过程中具有相同的作用，因此数据标准化有必要。
- 考虑下列一对数据点：

  $\mathbf{x}_i$: (0.1, 20) and $\mathbf{x}_j$: (0.9, 720).

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(0.9-0.1)^2 + (720-20)^2} = 700.000457,$$

- 这个距离完全由(720-20) = 700所主导。
- 标准化属性：强制各个属性都在一个相同的范围内变化。

■ 区间度量属性（Interval-scaled attributes）

- ◆这些属性是指数字/连续属性，它们是符合线性标量的实数。比如：年龄、身高、体重、价格等。
  - ● 10岁到20岁之间的差异与40岁到50岁之间的差异是相同的。
  - ● 主要思想：在整个线性度量空间上相同长度的区间具有相同的重要性。
- ◆两种主要的标准化区间度量属性的方法分别是：**范围标准化**和**z-score标准化**。f是一个属性。

$$range(x_{if}) = \frac{x_{if} - \min(f)}{\max(f) - \min(f)},$$

◆**Z-score标准化**：基于属性的**平均值**和**标准差**来对属性进行标准化的。

- Z-score标准化数值指出的是属性值从哪个方向远离属性均值以及偏离属性均值的距离，这个值的单位量是属性的标准差。属性f的标准差用s_f表示。计算公式如下：

$$s_f = \frac{1}{n}\left(|x_{1f} - m_f| + |x_{2f} - m_f| + \ldots + |x_{nf} - m_f|\right),$$

$$m_f = \frac{1}{n}\left(x_{1f} + x_{2f} + \ldots + x_{nf}\right),$$

Z-score:

$$z(x_{if}) = \frac{x_{if} - m_f}{s_f}.$$

■ 比例度量属性

- ◆属性是数字属性，但与区间度量属性不同的是，它的度量**不是线性的。**
- ◆比如：维生物数量随时间变化的规律可有由下面式子给出。

$$Ae^{Bt}$$

其中，A和B是某个正常数。

- ◆对于这样的属性，标准化的方法有：
  - ● 采用与区间度量属性一样的处理方法。会在刻度上扭曲，故不推荐。
  - ● **使用对数函数进行转换。**

$$\log(x_{if})$$

■ 符号属性

- ◆有时需要将符号属性转换成数值属性。
- ◆将**符号属性转换成布尔属性。**
  - ● 某个符号属性的属性值个数为v。
  - ● 首先建立v个布尔属性分别表示它们。
  - ● 当一个数据实例的该符号属性取了一个特定的属性值时，我们就把该属性值对应的布尔属性的属性值取为1。其余的取0。
- ◆得到的布尔属性可以被当作数值属性（取值0或1）

■符号属性-举例

- ◆对于符号属性*fruit*，有三个取值：Apple, Orange, and Pear。
- ◆建立三个布尔属性：Apple, Orange, and Pear
- ◆如果在数据中某个数据实例的*fruit*属性取值为Apple。
  - ●则在转换后的数据中，我们置属性Apple的值为1
  - ●属性Orange 和Pear 为 0

■顺序属性

- ◆与符号属性相似，但它的取值是有**特定顺序**的。比如：
  - ●年龄属性可能取值： Young, MiddleAge 和 Old，它们是有顺序的。
  - ●常用的标准化方法：把它当成区间度量属性，采用与其相同的方法来对这些属性进行标准化。

1.7 混合属性的处理

- 距离函数假定了数据中只有一种数据类型：都是数值属性/符号属性等。
- 实际上，数据集中可能含有混合属性：
  - 区间度量属性
  - 对称布尔属性
  - 非对称布尔属性
  - 比例度量属性
  - 顺序属性
  - 符号属性

■转换成一个类型

◆**一种方法**就是：

- ●先在混合属性中确定一个属性类型作为**主导类型**。
- ●然后把数据集中的其他类型的属性转换成该数据类型。

◆比如：一个数据集中大多数属性都是**区间度量属性**

- ●则可将顺序属性、比例属性转换成区间属性
- ●同样，也可以把对称布尔属性转换成区间属性

■转换成一个类型

- ◆将具有多于两个状态的符号属性或非对称属性转换为区间度量属性是没有意义的。
  - ●但是在实际情况中，人们还是经常根据某些隐含的顺序信息给这类属性赋予一些数值把它们转换成区间度量函数。如：fruit中的价格来给它们排序。
- ◆前面介绍过：可以把一个**符号属性**转换成多个对称布尔属性，从而使它被看作是一个区间度量属性。

■合并距离

◆另一种方法就是：

- 首先分别根据各个不同类型的属性计算两个数据点之间的距离。
- 然后把这些不同属性的距离结合起来得到一个最终距离。

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{f=1}^{r} \delta_{ij}^{f} d_{ij}^{f}}{\sum_{f=1}^{r} \delta_{ij}^{f}}$$

1.8 采用哪种聚类算法

- 聚类算法的研究和应用有着悠久的历史。人们设计出了一大批聚类算法。
  - 我们仅介绍了其中一部分算法。
- **选择一个在给定应用的数据集上“最好”的算法是一个具有挑战性的工作。**
  - 每一个聚类算法都有自己的局限性使得它们只能在某种特定的数据分布情况下有较好的聚类结果。
  - 要知道应用中的数据集到底是何种分布通常是很难的，甚至是不可能的。事实上，现实应用中的数据集往往并不能完全符合任何一种算法所要求的结构或者分布。
  - 另外，还需要如何标准化数据，以此确定一个合适的距离函数和选择其他参数（k-均值算法中的k）

- 由于这些复杂度，通常的做法是：
  - 执行不同的算法，使用不同的距离函数和不同的参数设置。
  - 并对结果进行仔细的分析和比较。
- 结果的理解需要同时建立在对原始数据的深刻理解和对于所使用的算法的认识之上。

1.9 聚类的评估

- **对于聚类算法的评估是很困难的**
  - 我们不知道在某个数据集上的正确聚类是什么。
- 有些聚类评估方法
  - 用户验证
    - 理解聚类结果的聚类中心
    - 依据规则
    - 文本文档的聚类。用户很容易阅读这些文档。

■ 聚类评估——真实数据

- ◆使用一些有标记的数据**（用分类数据集来评估聚类算法）**
- ◆**假设：**每一个类别对应一个聚类。
- ◆聚类操作之后，可以根据结果构造一个混合矩阵。从这个矩阵中，可以计算熵、纯度、查准率、查全率和F-score值来进行评估聚类的质量。
  - ●设数据集D中的类别集合$C = (c_1, c_2, \ldots, c_k)$.，聚类算法也生成k个聚类，这k个聚类把数据集D划分成k个两两不相交的集合$D_1, D_2, \ldots, D_k$

■熵

对于每个聚类，我们可以按下列方法度量它们的熵。

$$entropy(D_i) = -\sum_{j=1}^{k} \Pr_i(c_j) \log_2 \Pr_i(c_j),$$

其中，$\Pr_i(c_j)$是聚类i或D_i中属于类别c_j的数据点所占的比例。整个聚类结果（即所有聚类）的熵总和为：

$$entropy_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times entropy(D_i)$$

■纯度

这个度量方法度量的是一个聚类中仅包含一个类别的数据的程度。每个聚类的纯度可以按如下方法计算：

$$purity(D_i) = \max_j(\Pr_i(c_j))$$

整个聚类结果的纯度总和为：

$$purity_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times purity(D_i)$$

例子

假设有一个包含来自科学、体育和政治三个主题的900篇文档的文本集合D，并且每个文档都被标记为其中一个主题。我们在这个数据集上用聚类方法来发现三个聚类。主题标识在聚类过程中是不被使用到的。聚类完成以后，来评估这个聚类算法的效率。

Cluster	Science	Sports	Politics		Entropy	Purity
1	250	20	10		0.589	0.893
2	20	180	80		1.198	0.643
3	30	100	210		1.257	0.617
Total	300	300	300		1.031	0.711

■真实数据评估法的讨论

- ◆经常用来比较不同聚类算法的效率。
- ◆用于聚类的真实数据是没有类别标识。
  - ●因此，即使算法在某些已标注的数据集上有很好的聚类结果，但并不能保证这个算法在没有类别标识的实际应用数据上也会表现良好。
- ◆事实上，在一些已标注数据集上的良好表现能够使我们对于算法的质量有一定信心。
- ◆这种评估方法被称作是基于**外部数据**或**信息**的。

■基于内部信息的聚类评估

◆**聚类内紧密度**

- 数据点和聚类中心的相似度。
- 通常使用Sum of squared error (SSE)来进行测量。

◆**聚类间分离度**

- 不同的聚类中心之间的差异度。

◆在很多应用中，专家的意见仍然是最为重要的。

$$S_W^{(i)} = \frac{1}{N_i}\sum_{k=1}^{N_i}\left(\mathbf{X}_k^{(i)} - \mathbf{m}^{(i)}\right)\left(\mathbf{X}_k^{(i)} - \mathbf{m}^{(i)}\right)^T$$

$$S_B^{(ij)} = \left(\mathbf{m}^{(i)} - \mathbf{m}^{(j)}\right)\left(\mathbf{m}^{(i)} - \mathbf{m}^{(j)}\right)^T$$

■ 间接评估

- ◆在一些应用中，聚类操作并不是主要的任务，而是作为另一个任务的辅助。
- ◆我们可以利用主要任务的效率来衡量到底哪个聚类算法对于这个任务更合适。
- ◆比如：在Web使用信息挖掘的应用中，主要的任务是给网上客户提供图书的推荐功能。
  - ●如果能够根据客户的兴趣和以前历史购买记录来对顾客进行聚类，我们可能会提供一个更好的推荐服务。
  - ●我们就可以通过它们对于推荐服务提供的帮助上来评估这些聚类技术。
  - ●这里，我们假设推荐服务的结果可以很可靠地被评价。

■本章小结

- ◆聚类研究有着很长的历史，并且有着大量的工作
  - ●有很多聚类算法
  - ●每一年都有新的进展
- ◆本章仅介绍了一写广泛使用的算法，还有其他很多算法，
- ◆聚类评估很难，但在实际应用中却很有用。这也是为什么每一年还会研究出一系列聚类算法的原因。

多分类评价

- 通过混淆矩阵来求多分类问题的精确率、召回率和F1值
  - 若共有F类，则定义一个F × F 的混淆矩阵M，其内部元素$M_{i,j}$表示类i的节点被分类为j的个数。

$$Precision_i = \frac{M_{i,i}}{\sum_j M_{i,j}}$$

$$Recall_i = \frac{M_{i,i}}{\sum_i M_{i,j}}$$

$$F1_i = \frac{2Precision_i \times Recall_i}{Precision_i + Recall_i}$$

$$\begin{cases} \text{Precision} = \frac{\sum_{i=1}^{F} Precision_i}{F} \\ \text{Recall} = \frac{\sum_{i=1}^{F} Recall_i}{F} \\ \text{F1} = \frac{\sum_{i=1}^{F} F1_i}{F} \end{cases}$$

$$\text{Accuracy} = \frac{\sum_i M_{i,i}}{\sum_i \sum_j M_{i,j}}$$

聚类——作业

假设挖掘任务是将如下的八个点（用(x, y)代表位置）：

$A_1(2,10)$，$A_2(2,5)$，$A_3(8,4)$，$B_1(5,8)$，$B_2(7,5)$，$B_3(6,4)$，$C_1(1,2)$，$C_2(4,9)$

1、要求聚类为三个簇，假设初始我们选择分别$A_1B_1C_1$为每个簇的中心，用K-均值算法给出：

1）第一轮执行后的三个簇中心

2）最后的三个簇

2、要求聚类为两个簇，采用层次聚类中的合并聚类方法完成聚类。