层次聚类方法

戴 奇

主要内容

01 凝聚和分裂层次聚类

02 BIRCH：利用层次方法的平衡迭代归约和聚类

03 ROCK:分类属性的层次聚类算法

04 CURE：基于质心和基于代表对象方法之间的中间策略

05 Chameleon：利用动态建模的层次聚类算法

概要

- 层次聚类方法将数据对象组成一棵**聚类树**。
- 根据**层次分解**是以自底向上（合并）还是自顶向下（分裂）方式，层次聚类方法可以进一步分为凝聚的和分裂的。
- 一种纯粹的层次聚类方法的质量**受限**于：一旦合并或分裂执行，就不能修正。也就是说，如果某个合并或分裂决策在后来证明是不好的选择，该方法无法退回并更正。

主要内容

01 凝聚和分裂层次聚类

02 BIRCH：利用层次方法的平衡迭代归约和聚类

03 ROCK:分类属性的层次聚类算法

04 CURE：基于质心和基于代表对象方法之间的中间策略

05 Chameleon：利用动态建模的层次聚类算法

层次聚类方法

- 一般来说，有两种类型的**层次聚类方法**：
  - **凝聚**层次聚类：采用自底向上策略，首先将每个对象作为单独的一个原子簇，然后合并这些原子簇形成越来越大的簇，直到所有的对象都在一个簇中（层次的最上层），或者达到一个终止条件。绝大多数层次聚类方法属于这一类。
  - **分裂**层次聚类：采用自顶向下策略，首先将所有对象置于一个簇中，然后逐渐细分为越来越小的簇，直到每个对象自成一个簇，或者达到某个终止条件，例如达到了某个希望的簇的数目，或者两个最近的簇之间的距离超过了某个阈值。

例子

- 下图描述了一种凝聚层次聚类算法**AGNES**和一种分裂层次聚类算法**DIANA**对一个包含五个对象的数据集合{a,b,c,d,e}的处理过程。

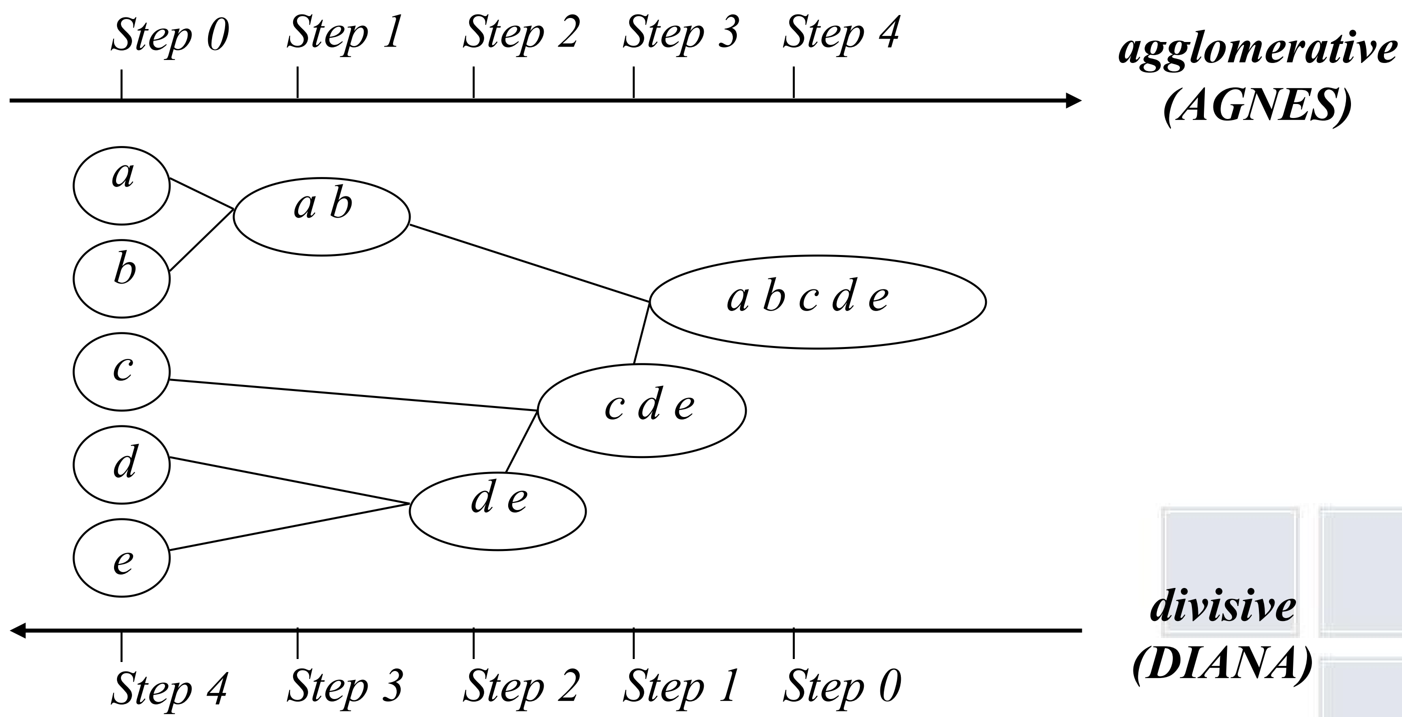


图1 对数据对象{a,b,c,d,e}的凝聚和分裂层次聚类

- 初始，AGNES将每个对象自为一簇，然后这些簇根据某种准则逐步合并，直到所有的对象最终合并形成一个簇。
  - 例如，如果簇C1中的一个对象和簇C2中的一个对象之间的距离是所有属于不同簇的对象间欧氏距离中最小的，则C1和C2合并。
- 在DIANA中，所有的对象用于形成一个初始簇。根据某种原则（如，簇中最近的相邻对象的最大欧氏距离），将该簇分裂。簇的分裂过程反复进行，直到最终每个新簇只包含一个对象。
- 在凝聚或者分裂层次聚类方法中，用户可以定义希望得到的簇数目作为一个终止条件。

树状图

- 通常，使用一种称作**树状图**的树形结构表示层次聚类的过程。它展示出对象是如何一步步分组的。图2显示图1的五个对象的树状图。

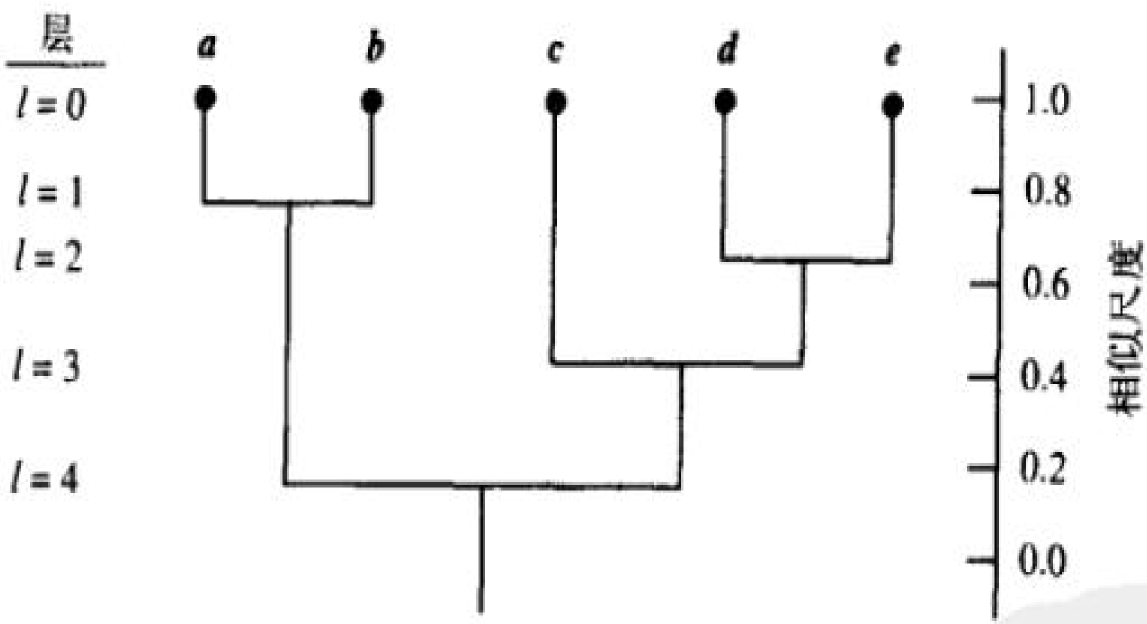
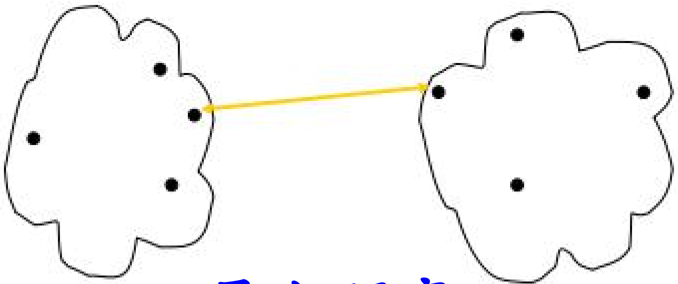


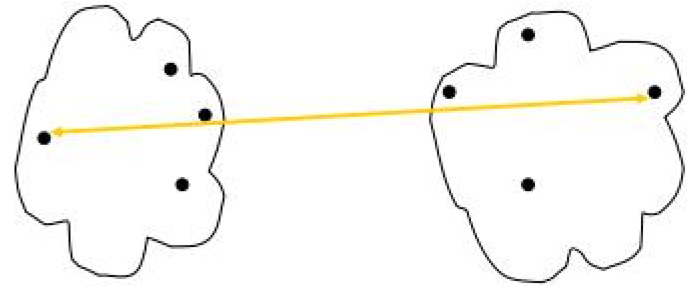
图2 数据对象{a,b,c,d,e}层次聚类的树状图表示

簇间距离

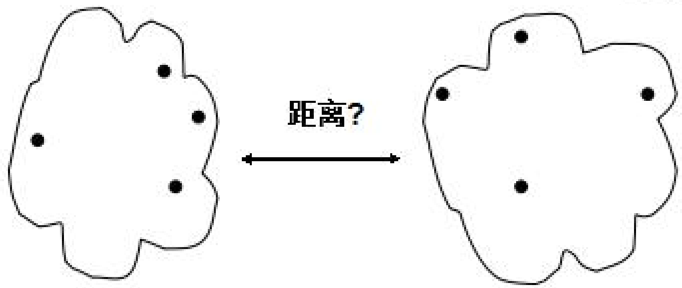
- 四个广泛采用的**簇间距离**度量方法如下，其中|p-p'|是两个对象或点p和p'之间的距离，m_i是簇C_i的均值，而n_i是簇C_i中对象的数目。
  - 最小距离：$d_{min}(C_i,C_j)=\min_{p\in C_i,p'\in C_j}|p-p'|$
  - 最大距离：$d_{max}(C_i,C_j)=\max_{p\in C_i,p'\in C_j}|p-p'|$
  - 均值距离：$d_{mean}(C_i,C_j)=|m_i-m_j|$
  - 平均距离：$d_{avg}(C_i,C_j)=\frac{1}{n_i n_j}\sum_{p\in C_i}\sum_{p'\in C_j}|p-p'|$

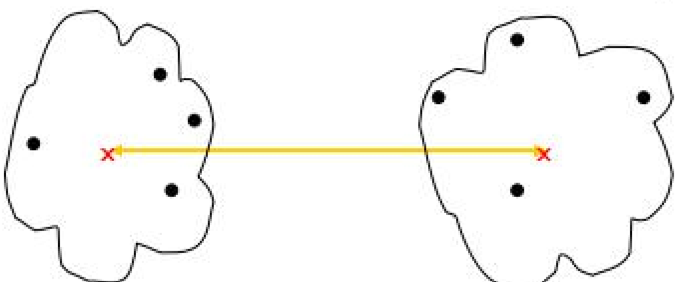


最小距离

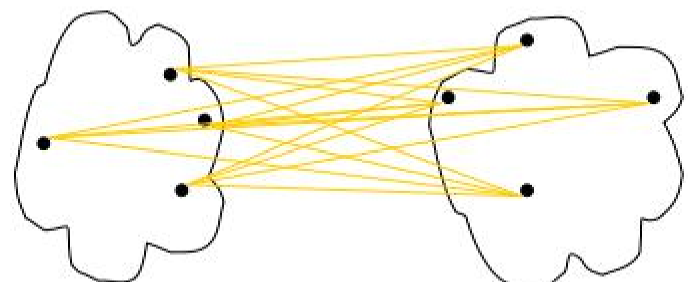


最大距离





均值距离



平均距离

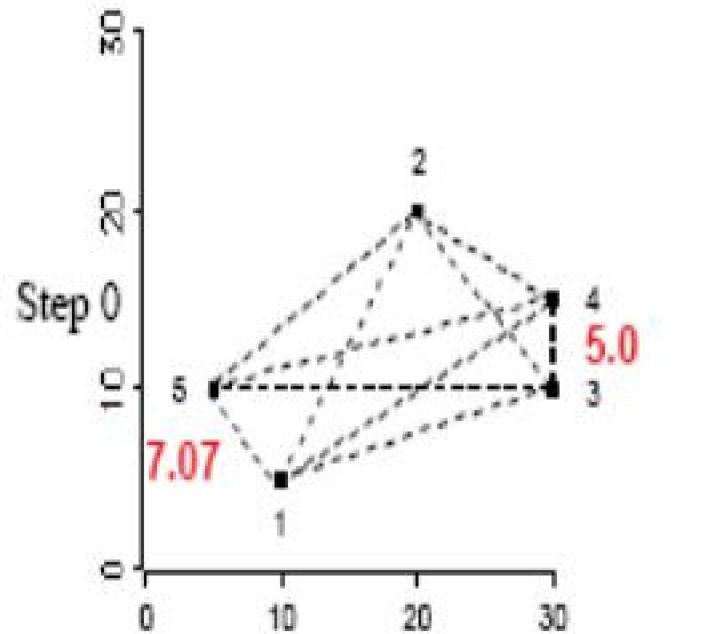
- 当算法使用最小距离$d_{min}(C_i,C_j)$衡量簇间距离时，有时称它为**最近邻聚类算法**。此外，如果当最近的簇之间的距离超过某个任意的阈值时聚类过程就会终止，则称其为**单连接算法**。
- 当一个算法使用最大距离$d_{max}(C_i,C_j)$度量簇间距离时，有时称为**最远邻聚类算法**。如果当最近簇之间的最大距离超过某个任意阈值时聚类过程便终止，则称其为**全连接算法**。

单连接算法例子

- 先将五个样本都分别看成是一个簇，最靠近的两个簇是3和4，因为他们具有最小的簇间距离D（3，4）=5.0。
- **第一步**：合并簇3和4，得到新簇集合1，2，（34），5

	x1	x2
1	10	5
2	20	20
3	30	10
4	30	15
5	5	10

	1	2	3	4	5
1	0.00				
2	18.0	0.00			
3	20.6	14.1	0.00		
4	22.4	11.2	5.00	0.00	
5	7.07	18.0	25.0	25.5	0.00



- **更新距离矩阵**：

D(1,(34))=min(D(1,3)，D(1,4))=min(20.6，22.4)=20.6

D(2,(34))=min(D(2,3)，D(2,4))=min(14.1，11.2)=11.2

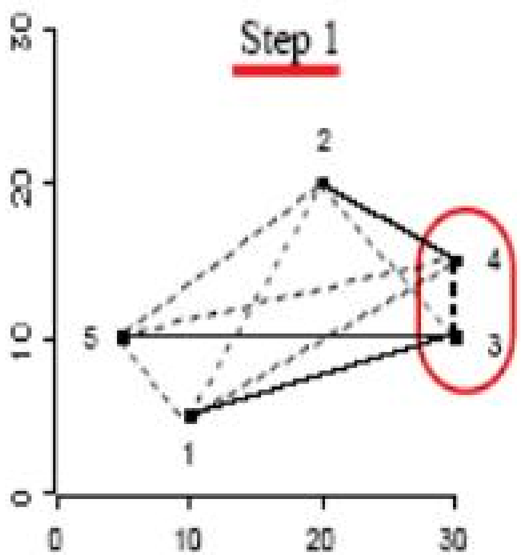
D(5,(34))=min(D(3,5)，D(4,5))=min(25.0，25.5)=25.0

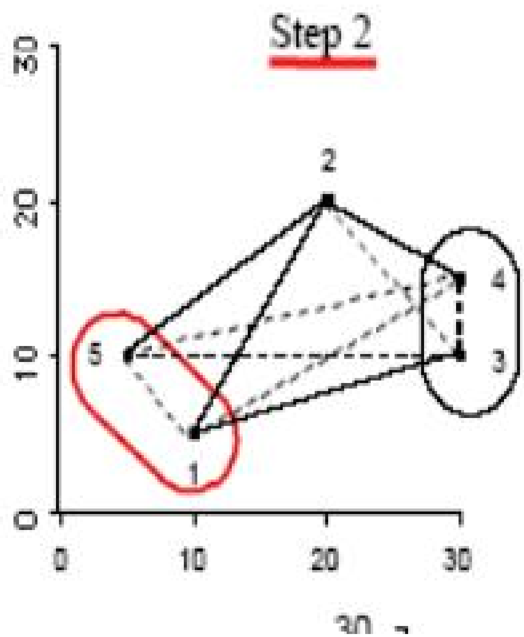
原有簇1，2，5间的距离不变，修改后的距离矩阵如图所示，在四个簇1，2，(34)，5中，最靠近的两个簇是1和5，它们具有最小簇间距离D（1,5）＝7.07。

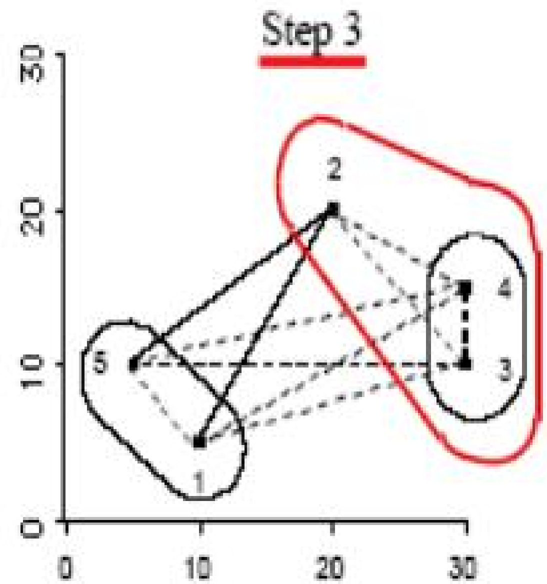
	1	2	5	(34)
1	0.00			
2	18.0	0.00		
5	7.07	18.0	0.00	
(34)	20.6	11.2	25.0	0.00

	2	(34)	(15)
2	0.00		
(34)	11.2	0.00	
(15)	18.0	20.6	0.00

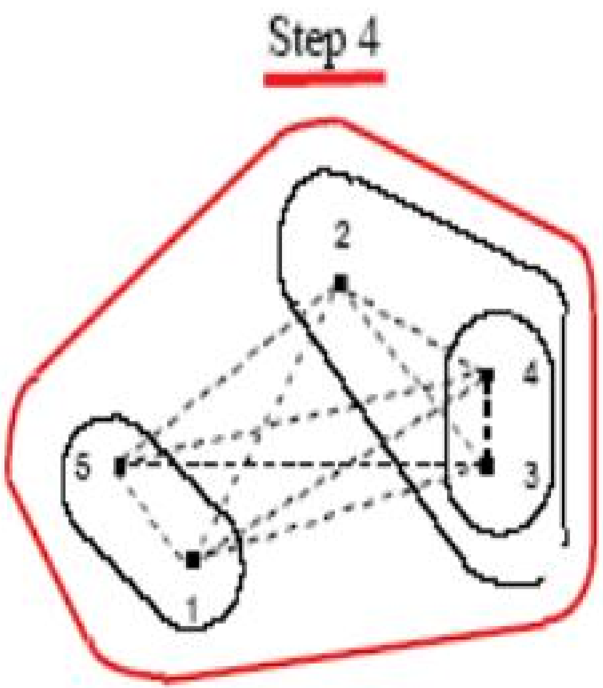
	(15)	(234)
(15)	0.00	
(234)	18.0	0.00



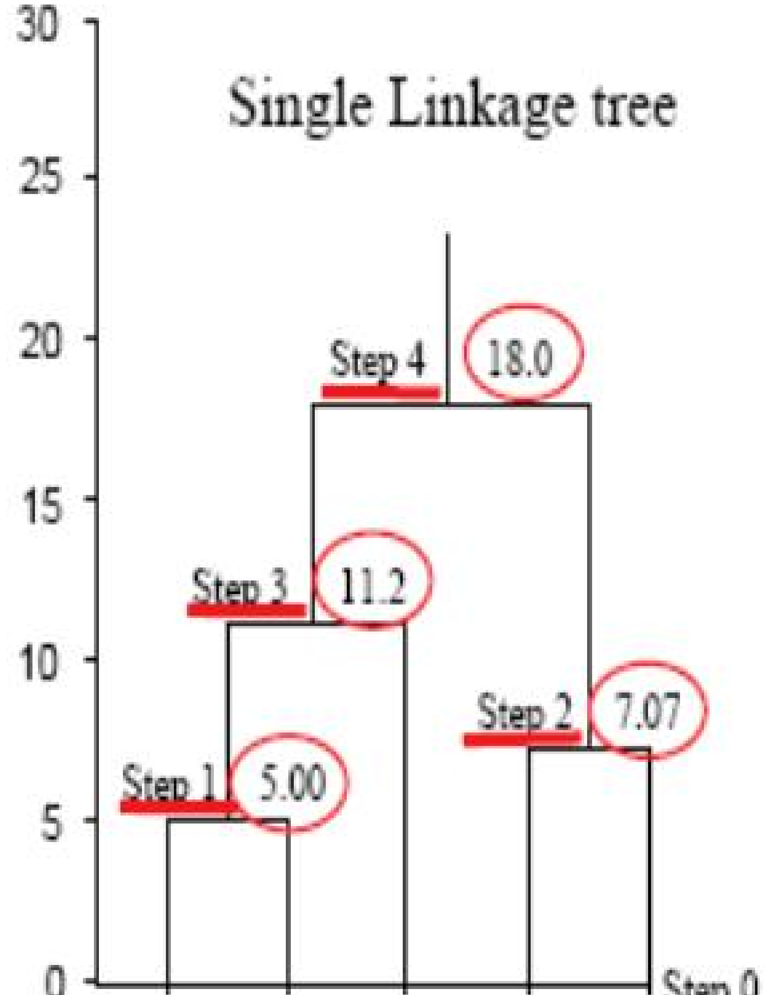




1 10 20 30



** solid lines show the minimum distances between clusters



- **最小**和**最大**度量代表了簇间距离度量的两个极端。它们趋向对**离群点**或**噪声数据**过分敏感。
- 使用**均值**距离和**平均**距离是对最小和最大距离之间的一种折中方法，而且可以克服离群点敏感性问题。
- 尽管均值距离计算简单，但是平均距离也有它的优势，因为它既能处理数值数据又能处理分类数据。

层次聚类方法的困难之处

① 层次聚类方法尽管简单，但经常会遇到合并或分裂点选择的困难。这样的决定是非常关键的，因为一旦一组对象合并或者分裂，下一步的处理将对新生成的簇进行。

② 不具有很好的可伸缩性，因为合并或分裂的决定需要检查和估算大量的对象或簇。

层次聚类的改进

- 一个有希望的方向是**集成**层次聚类和其他的聚类技术，形成多阶段聚类。在下面的内容中会介绍四种这类的方法：

  ①**BIRCH**：首先用树结构对对象进行层次划分，其中叶节点或者是低层次的非叶节点可以看作是由分辨率决定的“微簇”，然后使用其他的聚类算法对这些微簇进行宏聚类。

  ②**ROCK**基于簇间的互联性进行合并。

  ③**CURE**选择基于质心和基于代表对象方法之间的中间策略。

  ④**Chameleon**探查层次聚类的动态建模。

主要内容

01 凝聚和分裂层次聚类

02 BIRCH：利用层次方法的平衡迭代归约和聚类

03 ROCK:分类属性的层次聚类算法

04 CURE：基于质心和基于代表对象方法之间的中间策略

05 Chameleon：利用动态建模的层次聚类算法

- BIRCH方法通过集成**层次聚类**和**其他聚类算法**来对大量数值数据进行聚类。其中层次聚类用于初始的**微聚类阶段**，而其他方法如迭代划分（在后来的**宏聚类阶段**）。
- 它克服了凝聚聚类方法所面临的两个困难：

  ① 可伸缩性；

  ②不能撤销前一步所做的工作。
- BIRCH使用**聚类特征**来概括一个簇，使用**聚类特征树**（**CF树**）来表示聚类的层次结构。这些结构帮助聚类方法在大型数据库中取得好的速度和伸缩性，还使得BIRCH方法对新对象增量和动态聚类也非常有效。

聚类特征（CF）

- 考虑一个n个d维的数据对象或点的簇，簇的聚类特征是一个3维向量，汇总了对象簇的信息。定义如下

  $$CF=\langle n, LS, SS\rangle$$

  其中，n是簇中点的数目，LS是n个点的线性和（即$\sum_{i=1}^{n} x_i$），SS是数据点的平方和（即$\sum_{i=1}^{n} x_i^2$）。

- 聚类特征本质上是给定簇的统计汇总：从统计学的观点来看，它是簇的零阶矩、一阶矩和二阶矩。

- 使用聚类特征，我们可以很容易地推导出簇的许多有用的**统计量**。例如，簇的形心x_0，半径R和直径D分别是:

$$x_0 = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{LS}{n}$$

$$R = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}} = \sqrt{\frac{nSS - 2LS^2}{n^2}}$$

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}$$

其中R是成员对象到形心的平均距离，D是簇中逐对对象的平均距离。R和D都反映了形心周围簇的紧凑程度。

- 使用聚类特征概括簇可以避免存储个体对象或点的详细信息。我们只需要固定大小的空间来存放聚类特征。这是空间中BIRCH有效性的关键。
- 聚类特征是**可加**的。也就是说，对于两个不相交的簇C_1和C_2，其聚类特征分别为$CF_1=<n_1，LS_1，SS_1>$和$CF_2=<n_2，LS_2，SS_2>$，合并C_1和C_2后的簇的聚类特征是

$$CF_1+CF_2=<n_1+n_2, LS_1+LS_2, SS_1+SS_2>$$

例子

- 假定在簇C_1中有三个点（2,5），（3,2）和（4,3）。

  C_1的聚类特征是：

  CF_1=<3，(2+3+4,5+2+3),($2^2+3^2+4^2$,$5^2+2^2+3^2$)>=<3,(9,10,(29,38))>

  假定C_1和第2个簇C_2是不相交的，其中

  CF_2=<3,(35,36),(417,440)>。

  C_1和C_2合并形成一个新的簇C_3，其聚类特征便是CF_1和CF_2之和，即：

  CF_3=<3+3,(9+35,10+36),(29+417,38+440)>=<6,(44,46),(446,478)>

CF树

- CF树是一棵高度平衡的树，它存储了层次聚类的聚类特征。图3给出了一个例子。根据定义，树中的非叶节点有后代或“子女”。非叶节点存储了其子女的CF的总和，因而汇总了关于其子女的聚类信息。
- CF树有**两个参数**：分支因子B和阈值T。
  - 分支因子定义了每个非叶节点子女的最大数目。
  - 而阈值参数给出了存储在树的叶节点中的子簇的最大直径。
  - 这两个参数影响结果数的大小。

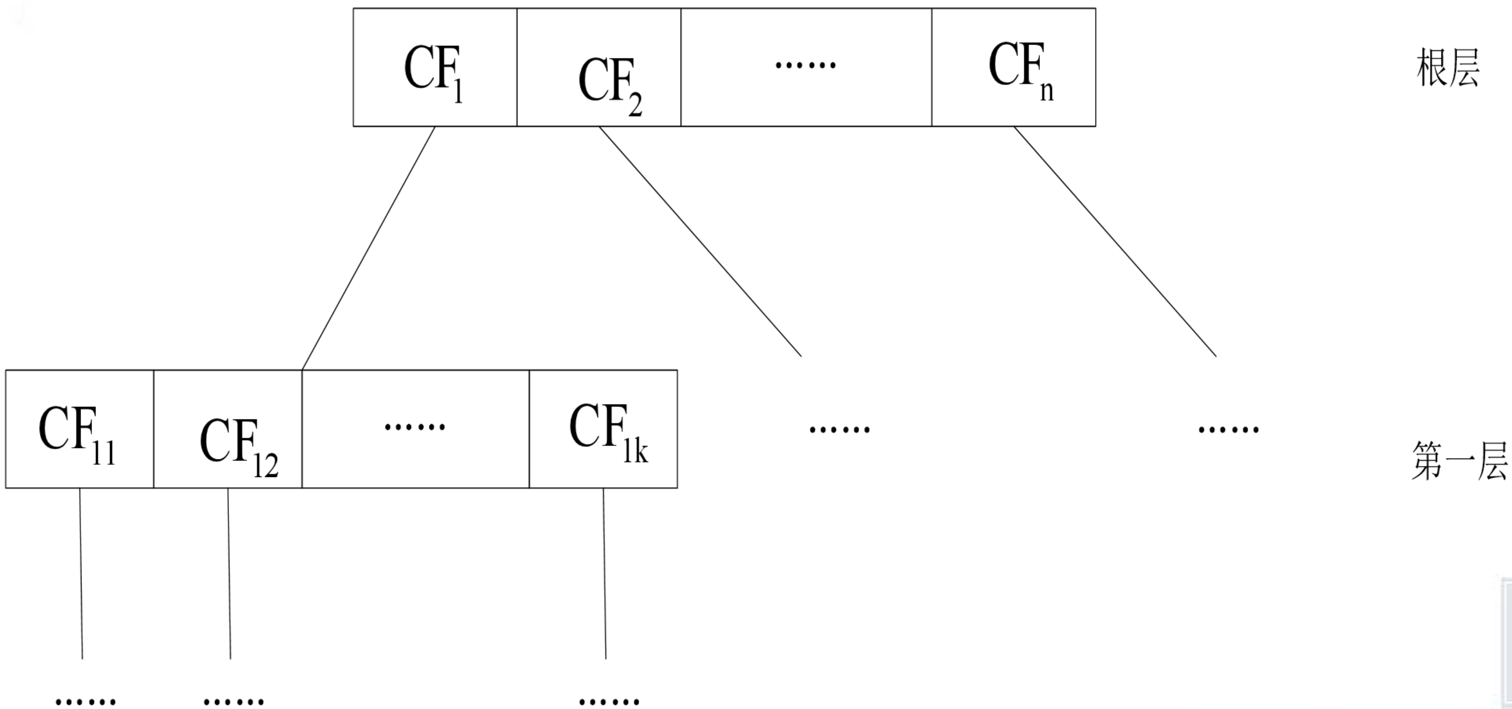


图3 CF树结构

- BIRCH试图利用可用的资源生成最好的簇。给定有限的主存，一个重要的考虑是最小化I/O所需时间。BIRCH采用了一种多阶段聚类技术：数据集的单遍扫描产生一个基本的好聚类，一或多遍的额外扫描可以用来进一步（优化）改进聚类质量。它主要包括**两个阶段**：
  - 阶段一：BIRCH扫描数据库，建立一棵存放于内存的初始CF树，它可以看作数据的多层压缩，试图保留数据的内在的聚类结构。
  - 阶段二：BIRCH采用某个（选定的）聚类算法对CF树的叶节点进行聚类，把稀疏的簇当作离群点删除，而把稠密的簇合并为更大的簇。

CF树的构造

- 在阶段一中，随着对象被插入，CF树被动态地构造。这样，该方法支持增量聚类。
- 一个对象被插入到最近的叶条目（子簇）。如果在插入后，存储在叶节点中的子簇的直径大于阈值，则该叶节点和可能的其他节点被分裂。新对象插入后，关于该对象的信息向树根节点传递。
- 通过修改阈值，CF树的大小可以改变。如果存储CF树需要的内存大于主存的大小，可以定义较大的阈值，并重建CF树。

- 在 CF 树重建过程中，通过利用老树的叶节点来重新构建一棵新树，因而树的重建过程不需要访问所有点，即构建 CF 树只需访问数据一次就行。
- 可以在阶段二使用任意聚类算法，例如典型的划分方法。

BIRCH的有效性

- 该算法的计算复杂度是O(n)，其中n是聚类的对象的数目。实验表明该算法关于对象数目是线性可伸缩的，并且具有较好的数据聚类质量。
- 然而，既然CF树的每个节点由于大小限制只能包含有限数目的条目，一个CF树节点并不总是对应于用户所考虑的一个自然簇。
- 此外，如果簇不是球形的，BIRCH不能很好地工作，因为它使用半径或直径的概念来控制簇的边界。

主要内容

01 凝聚和分裂层次聚类

02 BIRCH：利用层次方法的平衡迭代归约和聚类

03 ROCK:分类属性的层次聚类算法

04 CURE：基于质心和基于代表对象方法之间的中间策略

05 Chameleon：利用动态建模的层次聚类算法

- 对于聚类包含**布尔**或**分类**属性的数据，传统聚类算法使用距离函数。然而，实验表明对分类数据聚类时，这些距离度量不能产生高质量的簇。
- 此外，大多数聚类算法在进行聚类时只估计点与点之间的相似度；也就是说，在每一步中那些最相似的点合并到一个簇中。这种“**局部**”方法很容易导致错误。

- ROCK是一种层次聚类算法，针对具有**分类属性**的数据使用了**链接**（指两个对象间共同的近邻数目）这一概念。
- ROCK采用一种比较**全局**的观点，通过考虑成对点的邻域情况进行聚类。如果两个**相似的点**同时具有**相似的邻域**，那么这两个点可能属于同一个簇而合并。

- 两个点p_i和p_j是**近邻**，如果 $sim(p_i, p_j) \ge \theta$
  - 其中sim是相似度函数， sim可以选择为距离度量，甚至可以选择为非度量，非度量被规范化，使其值落在0和1之间，值越大表明两个点越相似。
  - θ是用户指定的阈值。
- p_i和p_j之间的**链接数**定义为这两点的**共同近邻个数**。如果两个点的链接数很大，则他们很可能属于相同的簇。
- 由于在确定点对之间的关系时考虑邻近的数据点，ROCK比起只关注点间相似度的标准聚类方法就显得更加**鲁棒**。

- 包含分类属性数据的一个很好的例子就是**购物篮数据**。
  - 这种数据由事务数据库组成，其中每个事务都是商品的集合
  - 事务看作具有布尔属性的记录，每个属性对应于一个单独的商品，如面包或奶酪。
  - 如果一个事务包含某个商品，那么该事务的记录中对应于此商品的属性值就为真；否则为假。
  - 其他含有分类属性的数据集可以用类似的方式处理。
- ROCK中**近邻**和链接的概念将在下面的例子中阐述，其中两个“点”即两个事务T_i和T_j之间的相似度用Jaccard系数定义为：

$$sim(T_i,T_j)=\frac{|T_i \cap T_j|}{|T_i \cup T_j|}$$

例子

- 假定一个购物篮数据库包含关于商品a,b,...,g的事务记录。考虑这些事务的两个簇C_1和C_2。
  - C_1涉及商品{a,b,c,d,e}，包含事务{a,b,c}，{a,b,d}，{a,b,e}，{a,c,d}，{a,c,e}，{a,d,e}，{b,c,d}，{b,c,e}，{b,d,e}，{c,d,e}
  - C_2涉及商品{a,b,f,g}，包含事务{a,b,f}，{a,b,g}，{a,f,g}，{b,f,g}
  - 假设我们首先只考虑点间的相似度而忽略邻域信息。C_1中事务{a,b,c}和{b,d,e}之间的Jaccard系数为1/5=0.2。
  - 事实上，C_1中任意一对事务之间的Jaccard系数都在0.2和0.5之间，而属于不同簇的两个事务之间的Jaccard系数也可能达到0.5。
  - 很明显，仅仅使用Jaccard系数本身，无法得到所期望的簇。

- 另一方面，ROCK基于链接的方法可以成功地把这些事务划分到恰当的簇中。事实证明，对于每一个事务，与之链接最多的那个事务总是和它处于同一个簇中。例如，
  - 令 θ =0.5，则C_2中的事务{a,b,f}与同样来自同一簇中的事务{a,b,g}之间的链接数为5（因为它们有共同的近邻{a,b,c}，{a,b,d}，{a,b,e}，{a,f,g}和{b,f,g}）
  - 然而，C_2中的事务{b,f,g}与C_1中的事务{a,b,c}之间的链接数仅为3（其共同的邻居为{a,b,d}，{a,b,e}，{a,b,g}）
  - 类似地，C_2中的事务{a,f,g}与C_2中其他每个事务之间的链接数均为2，而与C_1中所有事务的链接数都为0。因此，这种基于链接的方法能够正确地区分出两个不同的事务簇，因为它除了考虑对象间的相似度之外还考虑邻域信息。

- 基于这些思想，ROCK使用一个相似度阈值和共享近邻的概念从一个给定的数据相似度矩阵中首先构建一个稀疏图。然后在这个稀疏图上执行凝聚层次聚类。
- ROCK算法在最坏情况下的时间复杂度为

$$O(n^2+nm_m m_a+n^2\log n)$$

其中m_m和m_a分别是近邻数目的最大值和平均值，n是对象的个数。

主要内容

- 很多聚类算法只擅长处理球形或相似大小的聚类，另外有些聚类算法对孤立点比较敏感。
- CURE算法解决了上述两方面的问题，选择基于质心和基于代表对象方法之间的中间策略，即选择空间中固定数目的具有代表性的点，而不是用单个中心或对象来代表一个簇。
- 簇的代表点产生方式：首先选择簇中分散的对象，然后根据一个特定的分数或收缩因子向簇中心收缩或移动它们。在算法的每一步，有最近距离的代表点对（每个点来自于一个不同的簇）的两个簇被合并
- 该算法首先把每个数据点看成一簇，然后再以一个特定的收缩因子向簇中心“收缩”它们，即合并两个距离最近的代表点的簇。

核心步骤

- CURE算法采用**随机取样**和**划分**两种方法的组合，**核心步骤**如下：
  ①从源数据对象中抽取一个随机样本S；
  ②将样本S分割为一组划分；
  ③对每个划分局部地聚类；
  ④通过随机取样剔除孤立点。如果一个簇增长的太慢，就去掉它；
  ⑤对局部的簇进行聚类。落在每个新形成的簇中的代表点根据用户定义的一个收缩因子α收缩或向簇中心移动。这些点代表了簇的形状；
  ⑥用相应的簇标签来标记数据。

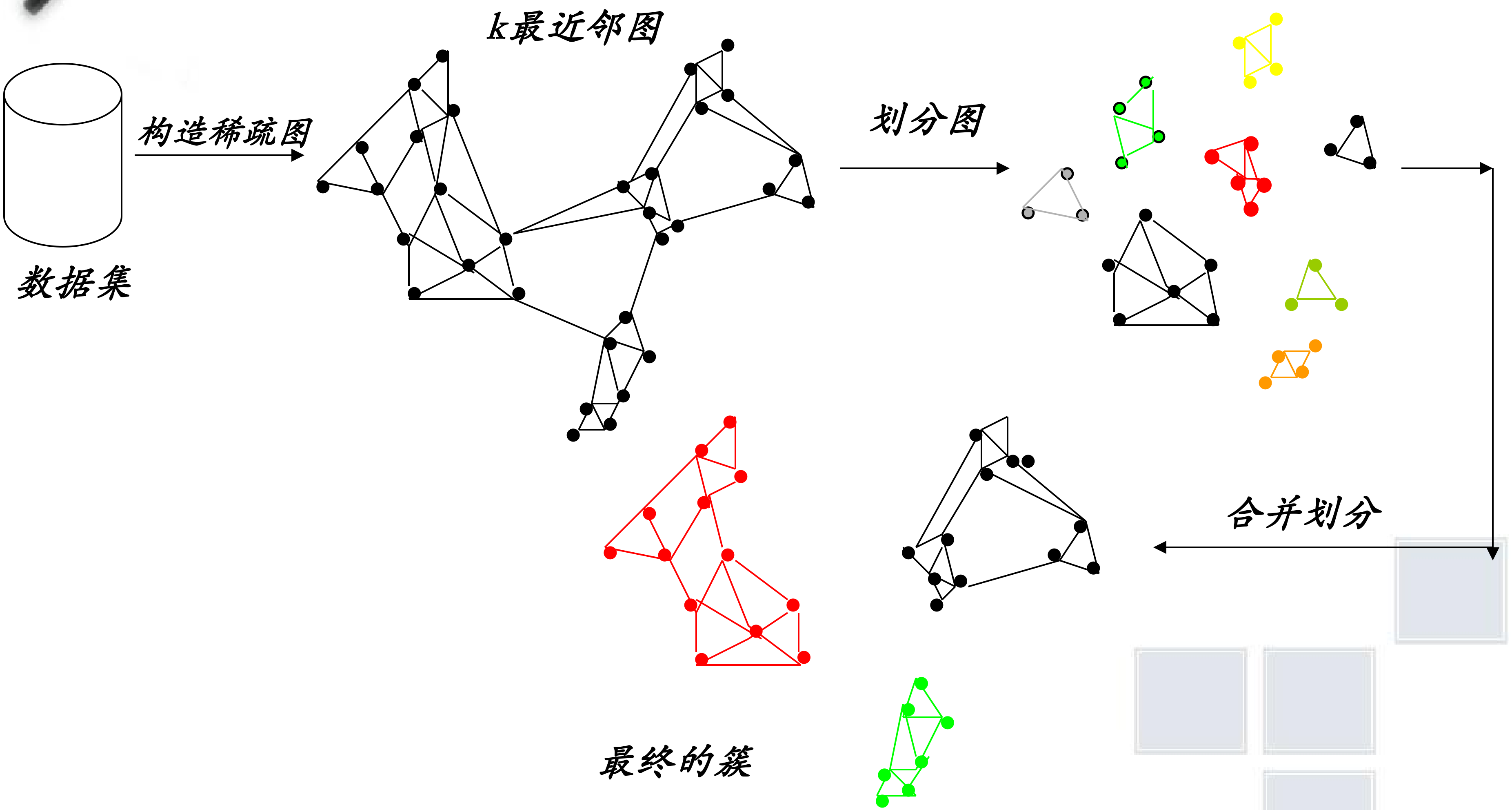
优点

- CURE算法优点：
  - 可以适应非球形的几何形状。
    - ✓将一个簇用多个代表点来表示，使得类的外延可以向非球形的形状扩展，从而可调整类的形状以表达那些非球形的类。
  - 对孤立点的处理更加健壮。
    - ✓收缩因子降底了噪音对聚类的影响，从而使CURE对孤立点的处理更加健壮
  - 而且能够识别非球形和大小变化较大的簇。
  - 对大型数据库有良好的伸缩性。
  - CURE算法的复杂性为$O(n)$。n是对象的数目，所以该算法适合大型数据的聚类。

主要内容

- Chameleon是一种层次聚类算法，它采用**动态建模**来确定一对簇之间的相似度。
- 在Chameleon中，簇的相似度依据如下两点评估：
  - （1）簇中对象的连接情况
  - （2）簇的邻近性
- 也就是说，如果两个簇的互连性都很高并且它们又靠的很近就将其合并。

Chameleon怎样工作?



- Chameleon算法的思想是：
  - 首先使用一种图划分算法将k最近邻图划分成大量相对较小的子簇。
  - 然后使用凝聚层次聚类算法，基于子簇的相似度反复地合并子簇。

图划分算法划分标准

- 为了确定最相似的子簇对，它既考虑每个簇的互连性，又考虑簇的邻近性。
- 图划分算法划分k最近邻图，使得**割边最小化**。也就是说，簇C划分为两个子簇C_i和C_j时需要切断的边的加权和最小。割边用EC（C_i，C_j）表示，用于评估簇C_i和C_j之间的绝对互连性。

- Chameleon根据每对簇C_i和C_j的相对互连度RI（C_i，C_j）和相对接近度RC（C_i，C_j）来决定它们之间的相似度：
  - 两个簇C_i和C_j之间的**相对互连度**RI（C_i，C_j）定义为C_i和C_j之间的绝对互连度关于两个簇C_i和C_j的内部互连度的规范化，即

$$RI(C_i, C_j) = \frac{\left|EC_{\{C_i, C_j\}}\right|}{\frac{\left|EC_{C_i}\right| + \left|EC_{C_j}\right|}{2}}$$

其中$EC_{\{Ci,Cj\}}$是包含C_i和C_j的簇的割边，如上面所定义。类似地，EC_{Ci}(或EC_{Cj})是将C_i(或C_j)划分成大致相等的两部分的割边的最小和。

- 两个簇C_i和C_j的的**相对接近度**RC（C_i，C_j）定义为C_i和C_j之间的绝对接近度关于两个簇C_i和C_j的内部接近度的规范化，定义如下：

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

其中 $\bar{S}_{EC_{\{C_i, C_j\}}}$是连接$C_i$中顶点和$C_j$中顶点的边的平均权重，$\bar{S}_{EC_{C_i}}$(或$\bar{S}_{EC_{C_j}}$)是最小二分簇$C_i$(或$C_j$)的边的平均权重。

特点

- 与一些著名的算法（如BIRCH和基于密度的DBSCAN）相比，Chameleon在发现高质量的任意形状的簇方面具有很强的能力。
- 然而，在最坏的情况下，高维数据的处理代价可能对n个对象需要 $O(n^2)$ 的时间。

谢谢大家