一个改进的基于DBSCAN的空间聚类算法研究

李 杰,贾瑞玉,张璐璐
(安徽大学 计算机科学与技术学院,安徽 合肥 230039)

摘 要:DBSCAN是一个基于密度的聚类算法。该算法将具有足够高密度的区域划分为簇,并可以在带有"噪声"的空间数据库中发现任意形状的聚类。但DBSCAN算法没有考虑非空间属性,且DBSCAN算法需扫描空间数据库中每个点的 ε -邻域来寻找聚类,这使得DBSCAN算法的应用受到了一定的局限。文中提出了一种基于DBSCAN的算法,可以处理非空间属性,同时又可以加快聚类的速度。

关键词:空间数据挖掘;聚类;密度;非空间属性
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2007)01-0114-03

Research on Improving Spatial Clustering Algorithm Based on DBSCAN

LI Jie, JIA Rui-yu, ZHANG Lu-lu
(School of Computer Science and Technology, Anhui University, Hefei 230039, China)

Abstract: DBSCAN is a spatial clustering algorithm based on density. It can handle spatial data and spot any-shape clusters in a noised spatial database by dividing them into clusters with high enough density. But DBSCAN does not take non-spatial properties into account and its strategy of scanning every points' ε-neighborhood to find clusters is very time-consuming, which makes it hard to apply. Proposes an improved DBSCAN algorithm which can handle non-spatial properties and greatly accelerate the speed of clustering.

Key words: spatial data mining; clustering; density; non-spatial attribute

0 引 言

聚类就是将数据对象分组成为多个类或簇,在同一个簇中的对象之间具有较高的相似度,而不同簇中的对象差别较大。在许多应用中,可以将一个簇中的数据对象作为一个整体来对待。根据数据的类型、聚类的目的和应用,聚类方法主要分为以下几类:划分方法、层次的方法、基于密度的方法、基于网格的方法和基于模型的方法[1]。DBSCAN算法是一个有代表性的基于密度的方法,其主要思想是:只要临近区域的密度(对象或数据点的数目)超过某个阈值,就继续聚类,否则,视为"噪声"。这样的方法可以用来过滤孤立点数据,发现任意形状的簇。一些改进的DBSCAN算法也相继被提出[2,3],如OPTICS算法,有效地屏蔽了对输入参数的敏感性。但大部分算法没有考虑非空间属性,这对于某些依赖非空间属性的聚类是不适用的,且DBSCAN算法需要扫描空间数据库中每个点的 ε -邻域来进行聚类,因而效率较低。

文中提出了一个改进的基于DBSCAN的算法,可以很好地解决以上两种问题。

收稿日期:2006-03-27

作者简介:李 杰(1979-),男,安徽合肥人,硕士研究生,主要研究方向为数据挖掘、人工智能;贾瑞玉,副教授,硕导,主要研究方向为机器学习、图形学、可视化数据挖掘。

1 DBSCAN算法及其改进的空间聚类算法

DBSCAN是一个基于密度的聚类方法。它将具有足够高密度的区域划分为簇,并可以在带有"噪声"的空间数据库中发现任意形状的聚类。它定义簇为密度相连的点的最大集合。

DBSCAN使用阈值 ε 和MinPts来控制簇的生成。其中,给定对象半径 ε 内的区域称为该对象的 ε -邻域。如果一个对象的 ε -邻域至少包含最小数目MinPts个对象,则称该对象为核心对象。给定一个对象集合 D,如果 p 是在 q 的 ε -邻域内,而 q 是一个核心对象,则说对象 p 从对象 q 出发是直接密度可达的。如果存在一个对象链 $p_1, p_2, \cdots, p_n, p_1 = q, p_n = p$,对 $p_i \in D, (1 \leqslant i \leqslant n)$,$p_{i+1}$ 是从 p_i 关于 ε 和MinPts直接密度可达的,则对象 p 是从对象 q 关于 ε 和MinPts密度可达的。如果对象集合 D 中存在一个对象 o,使得对象 p 和 q 是从 o 关于 ε 和MinPts密度可达的,那么对象 p 和 q 是关于 ε 和MinPts密度相连的。

DBSCAN通过检查数据库中每个点的 ε -邻域来寻找聚类。如果一个点 p 的 ε -邻域包含多于MinPts个点,则创建一个以 p 作为核心对象的新簇。然后,DBSCAN反复地寻找从这些核心对象直接密度可达的对象并加入该簇,直到没有新的点可以被添加,该过程结束。

1.1 DBSCAN处理非空间属性

DBSCAN通过阈值来控制聚类的发现,要使DBSCAN能够处理非空间属性,可以通过修改阈值域来实

现。对于非空间属性,根据应用,可以使用两种不同的修改方式。一是使用某一邻域内相同属性的绝对数量;二是使用某一邻域内相同属性的相对数量。定义与点 p 匹配的邻域 $N_\varepsilon(p) = \{q \in D \mid \mathrm{dist}(p,q) \leqslant \varepsilon \text{ and } \mathrm{propMatched}(p,q)\}$,其中,propMatched 为定义的布尔函数,用来匹配对象 p 和对象 q 的非空间属性,若相匹配,则返回 true,否则,返回 false。若 p 和 q 包含二维以上的非空间属性,则可遍历每个属性。对于第一种方式,直接计算 $|N_\varepsilon(p)|$,若 $|N_\varepsilon(p)| \geqslant \mathrm{MinPts}$,则 p 被聚类,否则,p 被视为"噪声"或孤立点。对于第二种方式,定义阈值 MinDen($\leqslant 1$),要求 $\{q \in D \mid \mathrm{dist}(p,q) \leqslant \varepsilon \text{ and } \mathrm{propMatched}(p,q)\} / \{q \in D \mid \mathrm{dist}(p,q) \leqslant \varepsilon\} \geqslant \mathrm{MinDen}$,即半径 ε 内相同属性的近邻数量和总近邻数量的比值要不小于 MinDen,才进行聚类。

1.2　加速 DBSCAN 聚类速度

为了保证发现密度相连的簇,DBSCAN 需要扫描每一个点及其邻域。从空间搜索的角度看,DBSCAN 算法走的是一条深度优先的搜索树。给定簇 $C1$,$C2$ 和 $C3$,如图 1 所示,假定 MinPts = 3,DBSCAN 随机选取点 1。点 1 有近邻点 2 和点 4,因此点 2 和点 4 成为如图 2 所示 $C1$ 树中点 1 的子树。接下来搜索点 2 的近邻,点 2 的近邻有 1,3,4,5,6。点 1 和点 4 已经在簇 $C1$ 中,则把 3,5,6 作为点 2 的子树。点 3 的近邻 2,5,6 都已在簇 $C1$ 中,因此点 3 没有子树。接着搜索点 2 的第二个近邻点 5,依此类推。最终的生成树如图 2 所示。

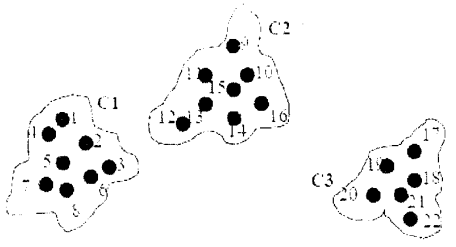

图 1　示例簇

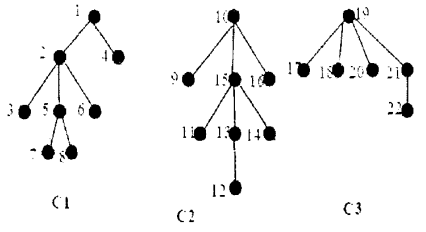

图 2　DBSCAN 构建的深度优先树

然而,搜索近邻而进行的空间查询是十分耗时的,事实上,当邻域含有的近邻数远远大于 MinPts 时,检索每个点的邻域是不值得的,因为已经知道这些近邻是属于这个簇的。可以把簇看做由核心对象和它的近邻所组成,在核心对象上执行区域查询,就可以发现簇。这里定义密度相交。给定点 q,簇 $C1$ 和 $C2$,若点 $q \in C1 \cap C2$,并且 $|N_\varepsilon(q)| \geqslant \mathrm{MinPts}$,则簇 $C1$ 和 $C2$ 是密度相交的。若簇 $C1$ 和 $C2$ 是密度相交的,则簇 $C1$ 和 $C2$ 中的核心对象和 q 是关于 ε 和 MinPts 密度相连的。这可由定义直接得出。随机选取一个未被聚类的点,检索它们的近邻,如果是核心点的话,若密度相交,则合并它们,否则,创建一个包含此核心点的新簇。重复此过程,如果足够的点被采样,则类被构造成功,从而避免了对每个点进行区域查询。在最坏的情况下,如果所有的点都是"噪声",它将和 DBSCAN 具有相同的耗费。对于图 1 的示例簇,使用改进的 DBSCAN 算法生成的搜索树如图 3 所示。

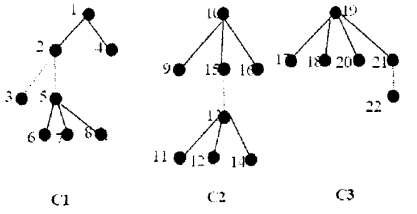

图 3　改进后生成的搜索树

该算法随机从深度和广度扩充已产生的簇。随机选取点 1,10,19,5,13,22,3。图中虚线表示分支被连接在已发现的簇上。检索完点 1,10,19,算法产生了三棵树。在 $C1$ 中,点 2 和 4 是点 1 的子树;在 $C2$ 中,点 9,15,16 是点 10 的子树;在 $C3$ 中,点 17,18,20,21 是点 19 的子树。当算法开始检索点 5 时,它的近邻是点 2,4,6,7,8。它和已有簇的分支相交点 2 和 4,因此,将点 5,6,7,8 插入到已有的簇的分支下(一般插入到相交的第一个节点)。其他依此类推,最终得到如图 3 所示的搜索树。

1.3　改进的空间聚类算法

下面给出改进后的算法,该算法随机选取一个点 q,检索它的邻域 Neighbors = NeighborsMatched(q,ε)。若 Neighbors 的数量大于 MinPts(如果按照第二种方式进行匹配,还须 Neighbors. Den 大于 MinDen),则 q 是一个核心点,否则,先把 q 作为"噪声"。如果 q 是核心点,算法检查它是否和已知的簇密度相交,若相交,则合并之,否则,产生一个新簇。

```
Algorithm E_DB(D, ε, MinPts, [MinDen])
ClusterTable = Null;
While(! D.Clustered)/* 未完成聚类 */
{ randomly select one unclustered point from D;/* 随即选取一个未分类的点 */
  Neighbors = NeighborsMatched (q, ε);/* 搜索该点的近邻 */
  If ( |Neighbors| ≤MinPts)/* or ( Neighbors.Den<MinDen) */
    q.Cluster = -1;/* q 暂时作为"噪声" */
  else
    { FirstToMerge = true;
      Ci = ClusterTable.firstCluster;
      /* 和已存在的簇进行比较 */
      While( Ci ! = Null)
        { if(Intersect(Neighbors, Ci)) /* 是否密度相交
```

```
*/
        if(FirstToMerge) /* 第一次相交 */
            /* 将 Neighbors 合并到 Cj 中 */
            { NewCluster = Merge(Ci,Neighbors);
              FirstToMerge = false;}
          else
            { NewCluster = Merge(NewCluster,Ci);
              ClusterTable.removeCluster(Ci);}
            Ci = ClusterTable.nextCluster;
            }
        /* 如果不相交的话,则产生新簇 */
        If(FirstToMerge)
         { Produce a new cluster Cj of Neighbors;
           ClusterTable.addCluster(Cj);}
    }
  }
```

算法检索完一个点的邻域后,随机选取另一个未被分类的点,重复以上过程,直至所有的点都被分类或归为"噪声"。算法中,空间查询 NeighborsMatched (q,ε) 是最耗时的部分,如果采用空间存取算法,如 R* - 树[4] 或 SR - 树[5],时间复杂度可以控制在 $O(\log n)$ 以内。在最坏的情况下,当所有的点都是"噪声"时,空间查询的复杂度为 $O(n\log n)$。然而,一旦有核心点被发现,它的近邻将不会被再次检索,从而可以大大减少空间查询的次数。

2 实验结果

实验采用三组仿真数据,其中第一组数据的样本容量为 100,第二组为 1000,第三组为 400。实验对第一、二组数据的聚类结果如图 4、图 5 所示。可以看出,该算法同 DBSCAN 一样可以很好地对含有"噪声"的空间数据进行聚类并发现任意形状的簇。

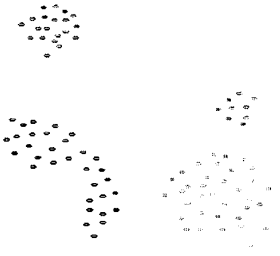

图 4 样本容量 100

图 5 样本容量 1000

图 6 给出了 DBSCAN 和 E_DB 在处理带有不同非空间属性后得出的结果。其中,给定的数据集中非空间属性包含 4 种不同的属性值。实验结果显示,DBSCAN 算法忽略了非空间属性,将全部数据集划分成一个簇;而该算法能够根据非空间属性发现更自然的 4 个簇,准确率达到 100%。对比几组实验数据,该算法比原始的 DBSCAN 算法节省 30%～40% 左右的时间,大大地加快了聚类的速度。

(a)DBSCAN (b)E_DB

图 6 DBSCAN 和 E_DB 的聚类结果

3 结 论

针对 DBSCAN 算法不能处理非空间属性和空间搜索效率低的问题提出了一些改进,改进后的算法能够有效地处理非空间属性,发现更自然的聚类,同时加快发现聚类的速度,具有一定的使用价值。对于大容量的数据,可以进一步通过引入启发条件来改进搜索算法,降低搜索的时间复杂度,从而获得理想的聚类结果。但本算法并未考虑障碍物[6]的存在,这使得聚类结果在某些场合难以适用,如商业网点的选址等。因此,如何处理含有障碍物的空间数据以产生符合实际应用的聚类将是下一步研究的主要工作。

参考文献:

[1] Han Jiawei, Kamber M. 数据挖掘概念与技术[M]. 北京:机械工业出版社,2001.

[2] 熊忠阳,孙 思,张玉芳,等. 一种基于划分的不同参数值的 DBSCAN 算法[J]. 计算机工程与设计,2005,26(9):2319 - 2321.

[3] 蔡颖琨,谢昆青,马修军. 屏蔽了输入参数敏感性的 DBSCAN 改进算法[J]. 北京大学学报:自然科学版,2004,40(3):480 - 486.

[4] Beckmann N, Kriegel H P, Schneider R, et al. The R* - Tree: An Efficient and Robust Access Method for Points and Rectangles[J]. SIGMOD Record, 1990, 19(2): 322 - 331.

[5] Katayama N, Satoh S. The SR - tree: An Index Structure for High - Dimensional Nearest Neighbor Queries[J]. SIGMOD Record, 1997, 26(2), 369 - 380.

[6] Tung A K H, Hou J, Han J. Spatial Clustering in the Presence of Obstacles[C]//In Proc of 2001 Int Conf on Data Engineering. Heidelberg, Germany: [s. n.], 2001: 359 - 367.