文章编号：1006-2475(2008)12-0008-04

基于网格和密度的 CLIQUE聚类算法的研究与实现

邓庚盛[1]，刘承启[1]，熊 艳[2]

（1.南昌大学网络中心，江西 南昌 330029；2.江西省通信管理局，江西 南昌 330046）

摘要：考虑到基于距离的传统聚类算法在实际应用中存在多方面的不足，本文研究了一种能弥补传统聚类算法不足的基于网格和密度的 CLIQUE聚类算法，并给出该算法实现的具体过程和主要代码。最后总结了该算法的特点和应用前景。

关键词：聚类；聚类高维空间；密度；网格

中图分类号：TP301.6 文献标识码：A

Research and Implementation of CLIQUE Clustering Arithmetic Based on Density and Gridding

DENG Geng-sheng, LIU Cheng-qi, XIONG Yan

(1. Network Center, Nanchang University, Nanchang 330029, China;
2. Jiangxi Communication Administration, Nanchang 330046, China)

Abstract: Considering much shortcoming of the traditional clustering arithmetics based on distance during practical implementation, the article researches on a new clustering arithmetic based on density and gridding named CLIQUE which can make up shortcoming of the traditional clustering arithmetics. Then, publishes the detailed process and mainly code for the arithmetic implemented. At the end of the article summarizes its characteristics and its application foreground.

Key words: clustering; CLIQUE; density; gridding

0 引 言

聚类分析的算法有很多，其中一大类的传统算法是基于距离的，这种基于距离的缺点在于只能发现球状的簇、处理大数据集和高维数据集时不够有效[1]，另一方面它能发现的聚类个数常常依赖于用户参数的指定，这对用户来说经常是很困难的。而基于密度（density-based）是一种只要一个区域中的点的密度大过某个阈值，就把它加到与之相近的聚类中去的方法，所以这种方法可以用于过滤“噪声”孤立点数据，发现任意形状的簇；基于网格（gridding-based）指将对象空间量化为有限数目的单元，形成一个网格结构，所有聚类都在这个网格结构上进行。这种方法的优点是它的处理速度很快，其处理时间独立于数据对象的数目，只与量化空间中每一维的单元数目有关。基于网格的和基于密度的方法分别从不同的方面弥补了传统算法的不足。目前有一种新的算法，这种算法抛弃了距离的概念，而采取一种新的思路，它就是基于网格和密度的聚类算法，它满足数据挖掘对聚类的典型要求[2]。基于网格和密度的典型聚类算法有 WaveCluster(小波变换聚类)算法和 CLIQUE(Clustering In QUEst,聚类高维空间)算法[3]。本文只讨论 CLIQUE聚类算法。

1 CLIQUE聚类算法的中心思想

CLIQUE聚类算法的中心思想如下：

（1）给定一个多维数据点的大集合，数据点在数据空间中通常不是均衡分布的。CLIQUE区分空间中稀疏的和“拥挤的”区域（或单元），以发现数据集合的全局分布模式。

（2）如果一个单元中的包含数据点超过了某个输入模型

收稿日期：2007-11-30

作者简介：邓庚盛（1980-），男，江西吉安人，南昌大学网络中心助理工程师，硕士，研究方向：计算机网格；刘承启（1977-），男，江西波阳人，助理工程师，硕士，研究方向：计算机网格；熊艳（1981-），女，江西南昌人，江西省通信管理局助理工程师，硕士，研究方向：计算机网络。

参数, 则该单元是密集的。在 CLIQUE算法中, 簇定义为相连的密集单元的最大集合[2]。

根据上述这种算法的中心思想, 基于网格和密度的聚类算法可定义为: 将数据空间分割成网格状, 然后将落到某个单元中的点的个数当成这个单元的密度。这时可以指定一个数值, 当某格单元各种点的个数大于该数值时, 就说这个单元格是密集的。最后, 聚类也就定义为连通的所有"密集"单元格的集合。

2　CLIQUE聚类算法的形式化描述

定义 1: 设 $A=\{D_1, D_2, \cdots, D_n\}$是 n个有界定义域, 则 $S=D_1\times D_2\times, \cdots, \times D_n$就是一个 n维空间, 将 $D_1, D_2, \cdots, D_n$看作 S的维 (属性、字段)。

CLIQUE算法的输入是一个 n维空间中的点集, 设为 $V=\{V_1, V_2, \cdots, V_n\}$, 其中 $V_i=\{v_{i1}, v_{i2}, \cdots, v_{in}\}$, V_i的第 j个分量 $v_{ij}\in D_j$。

通过输入一个参数 ξ, 可以将空间 S的每一维分成相同的 ξ个区间, 从而将整个空间分成有限个不相交的类矩形单元, 每一个这样的矩形单元可以描述为 $\{u_1, u_2, \cdots, u_n\}$, 其中 $u_i=[l_i, h_i)$, 一个前闭、后开的区间。

我们说一个 $V=\{V_1, V_2 \cdots, V_n\}$落入一个区间 $u=\{u_1, u_2, \cdots, u_n\}$, 当且仅当对于每一个 u_i都有 $l_i\leqslant v_i< h_i$成立。定义一个单元格 u的选择率 selectivity(u)为:

定义 2: selectivity=单元格中点的个数/数据空间中总的点个数。

对于用户的输入参数 τ, 称数据单元 u是密集的, 当且仅当 selectivity(u)>τ。

对于 S的任何子空间, 例如子空间 $Sub=D_1\times D_2\times, \cdots, \times D_k$($k< n$并且当 $i< j$时有 $t_i< t_j$成立), 同样可以在该子空间中定义单元格、选择率等相同概念。

定义 3: 一个聚类可以定义为: 在 k维空间中由一些连通的密集单元格组成的连通分支。两个 k维中的单元格 u_1、u_2称为连通的当且仅当: (1)这两个单元格有一个公共的面; (2) u_1、u_2都跟另一个单元格 u_3连通。两个单元格 $u_1=\{R_1, R_2, \cdots, R_{1k}\}$, $u_2=\{R'_1, R'_2, \cdots, R'_{1k}\}$有一个公共的面是指, 存在一个 k-1个维度 (不妨设这 k-1维就是 $t_1, t_2 \cdots t_{k-1}$), 有 $R_i=R'_i$成立 ($i=1, 2 \cdots, k$), 并且对于第 t_k维有 $h_{1k}=l_{2k}$或者 $h_{1k}=l_{2k}$成立。

3　CLIQUE聚类算法的实现

CLIQUE聚类算法的实现分三步:

第 1步: 将数据空间划分。

将数据空间划分为互不相交的长方形单元, 记录每个单元格中的对象数, 并且用先验性质识别包含簇的子空间。

先验性质是: 如果一个 k维单元是密集的, 那么它在 k-1空间上的投影也是密集的。即给定一个 k维的侯选密集单元, 如果检查它的 k-1维投影空间, 发现任何一个不是密集的, 那么知道第 k维的单元也不可能是密集的[3]。算法如下 (假定数据字段/维是经过排序的, "<"代表字段的辞典序):

① 令 k= 1, 通过遍历一遍目标数据库找出所有 1维的密集单元格, 不妨令所有组成的集合记为 D_1。

② 利用下面的方法, 由 k维的密集单元格集合 D_k生成 k+1维的候选密集单元格集合 C_{k+1}。

```
insert into C_{k+1}
select u1.[l1, h1), u1.[l2, h2), ..., u1.[lk, hk), u2.[lk, hk)
from D_k u1, D_k u2
where u1.d1= u2.d1, u1.l1= u2.l1, u1.h1= u2.h1, u1.d2= u2.d2, u1.l2= u2.l2
u1.h2= u2.h2, ..., u1.d_{k-1}= u2.d_{k-1}, u1.l_{k-1}= u2.l_{k-1}, u1.h_{k-1}= u2.h_{k-1}, u1.dk< u2.dk
```

③ 如果 C_{k+1}为空, 转④, 否则再一次遍历目标数据库, 计算侯选单元格中的选择率 selectivity, 在将非密集单元格去掉之后 (这样做的依据就是上文提到的先验性质), 记作为集合 D_{k+1}, k= k+1并转②。

④ 算法结束。得到包含簇的维数最高的子空间, 这也正是这一步的目标。

该步实现的数据结构和主要代码为:

```
struct leaf{ //对数据空间中的一个 unit的描述
long lCount; //该 unit中包含的点的个数
struct leaf* pNext; //指向该空间中下一个 unit的描述
int* pIntervalNo; //记录该 unit所在的空间位置
};
union son{
struct innode* pIn; //儿子仍然是内部节点
struct leaf* pLeaf; //儿子是一个描述当前空间的 unit链
int dimNo; //维号, 只在指向 leaf的 innode中有意义
};
struct innode{ //该步中用到的树的内部节点, 类似于 trie树 (字典树)
union son* pSon; //儿子的个数由的总维数决定
struct innode* pRight; //便于访问所有的叶节点而设
};
//构造一维时的树
pRoot= pJoin= innode_alloc(); //分配一个内部节点
for(i=0; i<m; lCount; i++){
```

```
pLeaf1 = pJoin->pSon[ j]. pLeaf = leaf_alloc(); //分配一个单元格结构
pLeaf1->pIntervaNo[ 0] =0;
for( i=1; i< delta; i++){
pLeaf1 = pLeaf1->pNext= leaf_alloc();
pLeaf1->pIntervaNo[ 0] = i;
}
}
lCurrenDim=1; //当前考虑的子空间的维数
while( lCurrenDim<= m_lColCount){
pRs->MoveFirst();
while(! pRs->MY_EOF){
transform(); //将当前记录值变换成区间号
increaseSubCount(); //将相应的单元格中的 lcount值更新
pRs->MoveNext();
}
deleteNonDense(); //去掉非密集的(nondense)单元格(units)
doMdlPrunning(); //基于 MDL的裁剪
if( doJoin())//做联接操作,若没有新的 candiates产生,算法结束。
Break;
lCurrenDim++; //考虑的子空间的维数加 1
}
```

第 1步是算法中唯一的要访问数据库的地方,它的时间复杂性仍然较大,效率不高。如果采用基于 MDL的裁剪就可以限制每一步产生的密集单元格个数,再生成高维的侯选密集单元格。这样做之后可以减少很大的计算量,降低了时间复杂性。基于 MDL的裁剪[4]是先将各个子空间进行排序,排序的依据是该子空间中所有密集单元格中所包含的点的总数,包含的点的个数多的子空间予以保留,少的去掉。这样做的一个启发性的依据在于:先验性质。

第 2步:识别簇。

在符合兴趣度的子空间中找出密集单元和在符合兴趣度的子间中找出相连的密集单元。

输入:一个密集单元格集合 D,并且处在同一个子空间中。

输出:集合 D的一个划分{ D_1, D_2, …, D_q},满足处在 D_i中所有密集单元格都是相邻的,并且处在不同的 D_i, D_j中的单元格是不相邻的, $D_i \cap D_j = D_1 \cup D_2 \cup \cdots \cup D_q = D$。

这个过程类似于寻找图的连通分支,只要将 D中的单元格看成是图的顶点,并且两个单元格相邻,它们之间就有一条边。然后,就可以采用常见的深度或广度优先搜索来完成这个任务。

数据结构和主要代码为:

```
struct cluster{ //记录一个簇类 cluster的信息
long * pUnitNo; //单元格 unit号
long unitsCount; //记录该簇类 cluster有多少个单元格 units
struct cluster * pNextCluster; //指向当前子空间的下一个 cluster
};
struct onesubspace{ //描述一个子空间
int dim; //维数
int * pdimNo; //分别记录这些维数分别对应的维号,即目标表的列号
struct leaf * pUnits; //指向密度单元格 dense units链
struct cluster * pCluster; //指向簇类 cluster链
};
for( k=0; k< subUnitsCount; k++){ //建立邻接矩阵
for( j=0; j< subUnitsCount; j++){
if( isConnect( pRandomleaf, k, j)){ //相邻吗?
pConnectMatrix[ k* subUnitsCount+ j] = pConnectMatrix[ j* subUnitsCount+ k] =1;
}
}
}
while(1){ //找出所有的连通分支
for( long ltemp1=0; ltemp1< subUnitsCount; ltemp1++){
if( pUnitsFlag[ ltemp1] ==0) break; //找到一个还没有被访问到的点
}
if( ltemp1== subUnitsCount) break; //所有的连通分之都找到了
pUnitsFlag[ ltemp1] =1;
pstack[ 0] = ltemp1; //bfs(广度优先)用到用来存放连通分支中所有点的栈及种子点
lLow=0;
lHigh=1;
while( lLow!= lHigh){ //bfs算法主体
i= pstack[ lLow];
for( k=0; k< subUnitsCount; k++){
if( pConnectMatrix[ i* subUnitsCount+ k] ==1&&pUnitsFlag[ k] ==0){
pstack[ lHigh] = k;
pUnitsFlag[ k] =1; //设置已访问标志
lHigh++;
}
}
lLow++;
}
pCluster1= cluster_alloc( lHigh); //分配一个描述簇类
```

cluster的结构

```
for( k=0; k< Hight; k++)
pCluster1-> pUnitNo[ k] = pstack[ k]; //记录下该簇类 cluster中的点
pCluster1-> unitsCount= Hight;
put_in_Cluster( pOnesubspace, pCluster1);
}
```

第 3步: 生成一个聚类的描述。

输入: k维子空间中的一个密集单元格的集合, 这个集合中的元素构成一个聚类 C。

输出: 一个区域 (其概念在上面已经定义过)的集合 R, R中的任意一个成员都包含在 C中, 且 C中任意一个单元格至少包含在 R的一个成员中。

数据结构和主要代码为:

```
structMaxRegion{ //描述一个类矩形, 即上下界
int* pRegionDef; //也即一个三位数组, low high区间个数
long count; //记录该 region覆盖了多少 units
loog* pUnitsCovered; //记录所覆盖的 units号
structMaxRegion* pNextRegion;
}
for( k=0; k< pCluster1-> unitsCount; k++) //清除覆盖标记
pConveredFlag[ k] =0;
while(1){ //一直需找直到 cluster中的 units被全部覆盖
for( k=0; k< pCluster1-> unitsCount; k++){
if( pCoveredFlag[ k] ==0) break;
}
if( k== pCluster1-> unitsCount) break; //全被覆盖, 结束
pRegion1= region_alloc(); //分配一个 region结构
initial_region( pCluster1-> pUnitNo[ k]);
k=0; //从第 0维开始增长, 以得到一个最大区域
while( k< CurrentDim){
up_increment(); //先往上增长
down_increment(); //再往下增长
k++;
} //结束后就得到一个 max region
insert_region( pRegion1);
} //结束后得到一个 max region链
minimize_region( pRegionList) //描述 cluster的最大区域的个数最小化
```

4 CLIQUE聚类算法的特点

从算法的实现过程可以看出, CLIQUE聚类算法具有两大特点:

(1)寻找包含簇的子空间时, 在面对高维的数据对象时, 根据单调性原理, 运用基于 MDL的裁剪方法, 将密集单元格的个数减少之后, 再生成更高维的候选密集单元格, 从而大大将低了时间复杂度。(2)寻找给定子空间的簇时, 采用深度优先的搜索策略; 同时建立索引, 便于随机访问; 在实现过程中, 采用堆栈模拟, 运用递归调用 DFS算法, 降低算法的时空复杂度。

5 结束语

数据挖掘是一个崭新的计算机应用领域, 而聚类分析技术已经成为数据挖掘研究领域中的一个非常活跃的研究课题。各种聚类算法有各自独有的特点和优势, 其中基于网格和密度的聚类算法由于综合了基于距离的聚类方法和基于网格的聚类方法两方面的优势, 所以它较其它的聚类方法有更广阔的应用领域。但是, 具体聚类算法选择要依据待处理数据的类型、聚类的目的和应用。

参考文献:

[1] 张红云, 石阳, 马垣. 数据挖掘中聚类算法比较研究[J]. 鞍山钢铁学院学报, 2001, 10(5): 364-363.

[2] Jiawei Han, Micheline Kamber. 数据挖掘概念与技术[M]. 范明, 孟小峰译. 北京: 机械工业出版社, 2001: 224-249.

[3] 陈京民. 数据仓库原理、设计与应用[M]. 北京: 中国水利水电出版社, 2004: 138-139.

[4] 彭青松, 张佑生, 汪荣贵, 等. 基于 MDL原理与混合遗传算法的 Bayesian网络结构学习[J]. 微电子学与计算机, 2002, 19(7): 27-29.

[5] 冯兴杰, 黄亚楼. 带约束条件的聚类算法研究[J]. 计算机工程与应用, 2005, 41(7): 12-14.

[6] 王洪艳. 基于聚类的数据挖掘技术在 CRM中的研究与应用[D]. 武汉: 武汉大学, 2005.

[7] 邱晓蕾. 基于网格的密度聚类算法[D]. 上海: 上海师范大学, 2006.

[8] 陈卓, 孟庆春, 魏振钢, 等. 一种基于网格和密度凝聚点的快速聚类算法[J]. 哈尔滨工业大学学报, 2005, 37(12): 1654-1657.

[9] 付淇, 李正凡. 基于 CLIQUE的聚类算法研究[J]. 华东交通大学学报, 2006, 23(5): 79-82.

[10] 石陆魁, 何丕廉. 一种基于密度的高效聚类算法[J]. 计算机应用, 2005, 22(8): 1824-1826.

[11] Abraham Silberschatz, Henry Ekorth, S Sudarshan. 数据库系统概念(第 4版)[M]. 杨冬青, 唐世谓, 等译. 北京: 机械工业出版社, 2004.

[12] Inmon W H. 数据仓库(第 5版)[M]. 王志海, 等译. 北京: 机械工业出版社, 2001.

[13] 单世民, 邓贵仕, 何英昊. 一种基于网格和密度的微粒群混合聚类算法[J]. 计算机科学, 2006, 33(11): 164-165.