计算机科学2002Vol.29№.8

快速关联规则挖掘与更新算法*)

An Fast Algorithm for Data Mining and Incremental Updating on Association Rules

杨 明[1,2] 孙志挥[1]

(东南大学计算机科学与工程系 南京210096)[1] (安徽机电学院计算机科学与工程系 芜湖241000)[2]

Abstract Data Mining and Incremental Updating on Association rules is a major aspect of data mining research. So how to fast discover or update association rules is the research focus. In this papers, we propose a new algorithm for fast data mining and incremental Updating on association rules by the prefix general linked list. Experimented results show that the algorithm not only facilitates the implementation ,but also improves the efficiency ,and avoids producing combinatorial explosion problem.

Keywords Prefix general linked list, Association rules, Incremental updating, Data mining

一、引言

众所周知,关联规则的挖掘就是发现支持度和信任度分别大于用户指定的最小支持度(minsup)和最小信任度的规则。支持度不小于 minsup 的项目集叫频繁项目集;反之,称为非频繁项目集。项目集中项目的数量叫做项目集的维数或长度,项目集 X 的支持度记作 sup(X)。有关项目集具有如下性质:(1)如果 X 是频繁项目集,那么 X 的任何子集都是频繁项目集;(2)如果 X 是非频繁项目集,那么 X 的任何超集都是非频繁项目集。

Agrawal 于1993年首次提出布尔型关联规则问题[1],并提出 Apriori 算法。其后数据挖掘领域的研究者在挖掘关联规则上做了大量的工作,主要致力于改进算法,提高算法速度和有效性,研究的工作体现在两方面。一是减少数据库的扫描次数,降低 I/O 负载代价,提高算法速度;二是减少候选频繁项目集的数量,避免知识的组合爆炸。

数据挖掘的研究者主要根据以上两个性质,在数据库关联规则的挖掘中通过剪枝、等价类、格等方法改进传统的 Apriori 算法,解决组合爆炸;同时进一步通过自底向上、自顶向下等搜索方法减少数据库的扫描次数,提高传统算法的速度[2~4]。由于数据库为非静态的,而是动态的,研究关联规则的更新算法具有十分重要的意义。已提出的关联规则更新算法的共同的不足是频繁集的负边界的求解代价高[5~10](空间代价和时间代价); DELTA 算法[10]对文[7~9]进行了综合改进,但仍须求解大量频繁集的负边界空间。我们在相关课题的研究中,发现要多次扫描数据库,是由于算法对已扫描的数据缺乏记忆的功能。为此,文[12]提出了一种前缀广义链表(GL)及基于此种结构的快速关联规则挖掘算法(PGLAR 算法),前缀广义链表灵活于 FP-Tree[11],它可以方便地进行分解和合并,有利于增量更新算法的实现,且相应的更新算法也可应用于分布式及并行环境下的数据库挖掘。限于篇幅,本文仅给出了增量更新算法(PGLIUA 算法),有关增量更新时前缀广义链表的调整及其与 FP-Tree[11]的比较将另文发表。PGLAR 算法仅须扫描原始数据库2遍,PGLIUA 算法扫描原始数据库最多为1遍,扫描新增数据库最多为2遍。本文以交易数据库为背景。

二、前缀广义链表

若把交易数据库的项目看成是某交通网的各站点,每次交易看成是某个旅客从某个起始站点经过若干中间站点到达目的站点。经常要统计哪些站点是旅客流量最大及哪些路径是频繁路径。经分析,只要统计经过各站点的旅客数,即知各站点的客流量,从而知哪些站点旅客流量最大。而通过统计某起始站点到达目的站点的旅客数,即可判断那些路径是频繁路径。

为此,对交易数据库 DB 的项目的排列次序进行相应的约束并引入前缀广义链表[12]。约束的方法由下面的(1)或(2)决定(本文仅按(1)进行讨论,(2)的讨论类似):

(1) 对交易数据库 DB 的各项目进行统计,依据项目的频繁度由大到小排列交易数据库 DB 中的每次交易;

(2) 对交易数据库 DB 的每次交易按某种字典次序排列。

定义1(前缀项) 按以上排序的项目子集中的任一项目前的项目子集为该项目的前缀项。特别地,约定项目子集的首项目的前缀项为空。

定义2(后缀项目) 按以上排序的项目子集中的尾项目被称为后缀项目。

定义3[12](前缀广义链表(GL)) 满足以下条件的广义链表称为前缀广义链表:

(1)每个结点由5部分组成:项目(item)、项目出现的次数(count)、指向孩子结点的指针(child)、指向兄弟结点的指针(sibling)、指向父结点的指针(parent);

(2)各结点以其所有祖先结点构成的项目子集为前缀项。

定义4[12](邻接表(Base)) 由头结点(item,count,first)和表结点(pointitem,next)构成的存储结构称为邻接表。其中,item 表示项目,count 表示项目 item 出现的频繁度,first 为指向表结点的指针;pointitem 为指向前缀广义链表的某结点的指针,next 为指向下一个表结点的指针。

性质1 前缀广义链表中,指向同一父结点的所有结点具

*)本课题得到国家自然科学基金(项目编号79970092)及安徽省教育厅自然科学研究基金(项目编号2001kj050)资助。杨 明 博士生,副教授,主要研究方向为知识发现与数据挖掘,神经网络理论及应用,粗集理论及应用。孙志挥 教授,博士生导师,主要研究方向为复杂系统信息集成和数据库系统及应用等。

有相同的前缀项。

性质2 前缀广义链表的深度为相应的交易数据库 DB 的频繁项目集的最大长度。

性质3 交易数据库的每次交易隐含在前缀广义链表的一条路径中。

三、关联规则挖掘算法(PGLAR 算法)[12]

基于前缀广义链表的关联规则的挖掘分为两步。首先,依据交易数据库建立前缀广义链表;其次,依据建立的前缀广义链表进行关联规则的挖掘。

3.1 前缀广义链表的生成算法[12]

算法1 CPGL 算法 //前缀广义链表的生成算法

```
input:(1)DB 为交易数据库, DB 有 m 个项目(I1,…,Im),发生了 n 次
      交易。(2)minsup^DB 为最小支持度。
output:前缀广义链表 GL^DB。
begin
(1)for(i=1;i<=m;i++)Base[i]=(Ii,0,NULL);//初始化 Base
(2)GL^DB=NULL;//头指针初始化
(3)for 任意 T∈DB do
(4)   for T 中的任一项目 I do
(5)     { i=location(I);Base[i].count++;} //location(I)表示 I 在
        Base 中的下标
(6)for 任意 T∈DB do
(7) { T←filter(T);//filter()为过滤支持度<minsup^DB的项目;
(8)   T←sort(T,Base);//依据 Base 的 count 域的统计值由大到小
      排列 T 中的项目
(9)    GL^DB=insert(GL^DB,T,NULL);//将 T 的各项目插入到前缀
       广义链表 GL^DB中
(10)   }
(11)   return(GL^DB);
end.
insert(GL,T,parent) //将处理后的交易 T 插入到前缀广义链表 GL
                    中
{ I=getfirstitem(T);if(I==NULL)return;
 if(GL!=NULL)
 {
if(GL->item==I){GL->count++;T=delete(T,I);
  insert(GL->child,T,GL);}//delete(T,I)为从 T 中删除 I
 else
  if (GL->sibling==NULL || I.count<GL->item.count)//为方
     便,简记项目.count 为该项目的频繁度
    { new(p);p->item=I;p->count=1;p->parent=parent;
      p->child=NULL;p->sibling=GL->sibling;GL->sibling=
      p;
      i=location(I);new(q);q->pointitem=p;q->next=Base[i]
      .first;Base[i].first=q;
      insert(p->child,T=delete(T,I),p);
    }
    else
     insert(Gl->sibling,T,parent);
 }
else
{ new(GL);GL->item=I;GL->count=1;GL->parent=parent;
  GL->child=NULL;GL->sibling=NUll
  i=location(I);new(q);q->pointitem=GL;q->next=Base[i]
  .first;Base[i].first=q;
  insert(GL->child,T=delete(T,I),GL);
  } }
```

3.2 关联规则挖掘算法[12]

算法2 PGLAR 算法//基于前缀广义链表的关联规则挖掘算法

```
input:(1)DB 为交易数据库,DB 有 m 个项目(I1,…,Im),发生了 n 次
交易,(2)minsup^DB、minconf^DB分别为最小支持度、最小信任度。
output:频繁项集 F^DB,交易数据库的关联规则 AR^DB;
begin
(1)GL=CPGL(DB, minsup^DB);//由算法1
(2)F^DB=φ;AR^DB=φ;//初始化
(3)i=1;
(4)while(i<=m);
(5){ if(Base[i].count>= minsup^DB)
(6) { B[]=NULL;//初始化
(7)   for(q=Base[i].first,j=1;q!=NULL;q=q->next,j++)
(8)    if(q->pointitem->parent)B[j]=q->p->parent;
         //记录项目 Base[i].item 对应的各路径
(9)     m=j-1;//路径数
(10)      F^DB=F^DB∪gene_F(B,m,Base[i].item,[]);//[]表示空
          集,gene_F 为生成以 Base
(11)[i].item 为后缀项目的所有频繁项目集
(12)   }}
(13) AR^DB=AR^DB∪gene_AR(F^DB);}//生成关联规则
end.
```

定理1 对某一交易数据库 DB,DB 上的所有频繁基本项 I 可由 PGLAR 算法得到。

证明:不妨设 I 的项目集为$\{x_1,x_2,\cdots,x_s\}$,且在 DB 中 $x_1.count \geqslant x_2.count \geqslant \cdots \geqslant x_s.count$,则 $x_1x_2\cdots x_s$ 必是 CPGL 算法生成的前缀广义链表的一条路径中,把 x_s 看成是 Base[i],则必由 PGLAR 算法得到频繁基本子项 I。证毕。

3.3 应用实例

对表1所示的交易数据库 DB,若取 $minsup^{DB}=0.5$,$minconf^{DB}=0.75$,则由 CPGL 算法可得前缀广义链表,如图1所示。容易看出其可以分解成两个子前缀广义链表。对建立的前缀广义链表图1应用 PGLAR 算法可得交易数据库 DB 的关联规则,如表2所示。

表1 交易数据(DB)

交易	数据项
t1	abc
t2	abd
t3	ae
t4	bc
t5	bcd
t6	e

表2 交易数据库 DB 的关联规则

关联规则	支持度	信任度
a=>b	0.5	0.75
b=>a	0.5	0.75
b=>c	0.5	0.75
c=>b	0.5	0.75

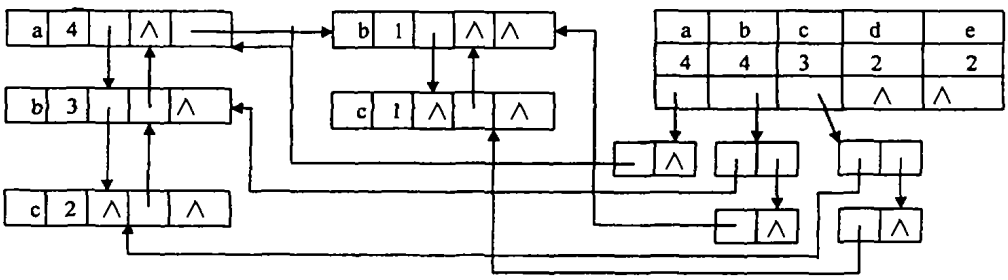

图1 前缀广义链表($minsup^{DB}=0.5$)

运用 PGLAR 算法挖掘交易数据库的关联规则仅需要扫描数据库2次;同时,第1遍扫描得到了交易数据库的1_项频

繁项目集，第2遍扫描交易数据库，使得交易或完全与前缀广义链表的一条路径重合或部分重合或新建一条路径，由于频繁项目集蕴含在相应的路径中，显著缩小了频繁项目集空间，避免了"知识的组合爆炸"问题。

四、关联规则的增量更新算法(PGLIUA 算法)

关联规则更新分两种情况：(1)为了发现用户感兴趣的未知的关联规则，在数据库不变的情况下通过最小支持度和最小信任度这两个阈值的不断调整来得到那些真正令其感兴趣的关联规则；(2)在数据库随着时间发生变化且最小支持度和最小信任度这两个阈值变化或不变的情况下更新关联规则。由于(1)可以看成(2)在数据库几乎不变情况下的特例，我们仅对(2)进行关联规则更新的讨论。

定理2 如果 X 是某记录集 D 的一个频繁项目集，将 D 分为 m 部分 $D_1, D_2, \cdots, D_m$，其中，所含记录个数分别为 $N_1, N_2, \cdots, N_m$，则 X 至少在 m 个分区之一是频繁项目集。

证明：m 个分区对应于 m 个鸽巢，项目集 X 在 D 中出现的次数对应于鸽子的个数，大于 $(N_1+N_2+\cdots+N_m)\times$minsup 次(minsup 为最小支持度)。由鸽巢原理，可知至少在一个分区 D_i 里，X 出现的次数$\geq N_i\times$minsup 次，即 X 在分区 D_i 中是频繁项目集。证毕。

推论1 如果 X 在记录集 D 的 m 个分区 $D_1, D_2, \cdots, D_m$ 上均不是频繁项目集，则 X 一定不是 D 的频繁项目集。

算法3 PGLIUA 算法//基于前缀广义链表的关联规则更新算法

input：(1)DB 为交易数据库，DB 有 m 个项目$(I_1, \cdots, I_m)$，发生了 N_1 次交易；db 为增量数据库，发生了 N_2 次交易；(2)$minsup^{DB}$为 DB 的最小支持度，$minsup^{DB\cup db}$为 DB∪db 的最小支持度。$minconf^{DB\cup db}$为 DB∪db 的最小信任度。

output：DB∪db 的前缀广义链表 $GL^{DB\cup db}$，DB∪db 的关联规则 $AR^{DB\cup db}$。

```
begin
(1)F^{DB∪db}=φ; AR^{DB∪db}=φ;
(2)PGLAR(db, minsup^{DB∪db}, minconf^{DB∪db});//得增量数据库 db 的
   频繁项目集和关联规则 F^{db}
(3)Base←Base+base;//Base 和 base 分别为 DB 和 db 的邻接表
(4)if(DB∪db 的1-频繁项目集!=DB 的1-频繁项目集)
(5)  {GL^{DB}=regene_GL(DB, GL^{DB});//重新调整生成 GL^{DB}(需扫
     描 DB 数据库一遍)
(6)   F^{DB}=regene_F(GL^{DB}, minsup^{DB∪db});//重新生成 DB 的频繁
     项目集
(7)   Base←Base+base;
(8)  }
(9)   else{
(10)    if(minsup^{DB∪db}<minsup^{DB})F^{DB}=adjust(GL^{DB});//重新调整
        DB 的频繁项目集
(11)    if(minsup^{DB∪db}>minsup^{DB}) F^{DB}=filter(F^{DB});//过滤 F^{DB}中
        的非频繁项目集
(12)    }
(13)    F'=F^{db}-F^{DB};// F'仅为 db 上的频繁项目集
(14)    for(F'中的任意频繁项目集 X)
(15)    { X.count←count(GL^{DB},X)+X.count;
(16)     if(X.count>= minsup^{DB∪db}) F^{DB∪db}= F^{DB∪db}∪{X};}
(17)  F''=F^{DB}-F^{db};// F''仅为 DB 上的频繁项目集
(18)   for(F''中的任意频繁项目集 X)
(19)   { X.count←count(GL^{db},X)+X.count;//X.count 为项目集
       X 的发生次数
(20)    if(X.count>= minsup^{DB∪db}) F^{DB∪db}=F^{DB∪db}∪{X};}
(21)    AR^{DB∪db} = AR^{DB∪db} ∪ gene_AR(F^{DB∪db});//依据 min-
        conf^{DB∪db}生成关联规则
(22)   GL^{DB∪db}=merge(GL^{DB},GL^{db});//合并 GL^{DB}与 GL^{db}
(23)  }
end.
```

由于 GL 可以灵活地进行合并和分解，PGLIUA 算法可以方便应用于分布式及并行环境中，同时也可解决内存不足问题。

五、实验结果

我们用 Visual C++6.0在64M 内存、CPU 为 C466实现了 PGLIUA 算法并进行了性能测试。利用 http://www.ics.uci.edu/~mlearn/MLSummary.html 上所提供的蘑菇数据库(mushroom database)来进行实验，该数据库(共有蘑菇的23种属性)有8124条记录。随机抽取8000个记录，并分成两部分 DB(6000个记录)和 db(2000)个记录，$minsup^{DB}=0.03$；对 PGLIUA 算法进行测试，实验表明(如图2所示)PGLIUA 算法是有效可行的。

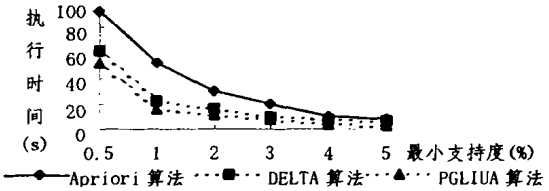

图2 算法的执行时间

结束语 针对文[12]提出的前缀广义链表及基于此结构的快速关联规则挖掘，本文给出了相应的增量更新算法。限于篇幅，有关增量更新时前缀广义链表的调整、与 FP-Tree[11]的比较及实验的进一步验证等将另文发表。理论分析与实验结果都表明，文中提出的算法不仅有效可行且避免了"知识的组合爆炸"问题。本文算法已经用于企业风险决策管理的 CRM 项目中，并取得了良好的效果。

参 考 文 献

1 Agrawal R, ImielinSki T, Swami A. Mining association rules between sets of items in large database . Proceeding of the ACM SIGMOD International Conference On Management of Data, 1993 (2): 207～216

2 冯玉才，冯剑琳. 关联规则的增量式更新算法. 软件学报，1998，9(4)：301～306

3 马元元，孙志挥，高红梅. 时态数据库中增量关联规则的挖掘. 计算机研究与发展，2000，37(12)：1446～1451

4 Bayardo R J. Efficiently mining long patterns from database. In SIGMOD '98, 85～93

5 Han J, Pei J, and Yin Y. Mining partial periodicity using frequent pattern tree. In CS Tech. Rep, 99～100, Simon Fraser University, July 1999

6 Vikram P, Haritsas J R. Quantify the utility of the past in mining large database . Information Systems ,2000,25(5):323～343

7 Cheung D, Han J, Ng V, Wong C. Maintenance of discovered association rules in large databases: an incremental updating technique. In: Proc. of the 12th Intl. Conf. on Data Engineering (ICDE), New Orleans ,Louisiana, IEEE Computer Society, 1996. 106～114

8 Cheung D, LEE S, Kao B. A general incremental technique for maintaining discovered association rules. In: Proceedings of the 5th Intl. Conf. on Database Systems for Advanced Applications (DASFAA), Melbourne, Australia, World Scientific, 1997. 185～194

9 Feldman R, Aumann Y, Amir A, Mannila H. Efficient algorithms for discovering frequent sets in incremental databases. In: Proc. of SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Tucson, Arizona, 1997

10 Pudi V Haritsa J R. Quantifying the utility of the past in mining large database. Information Systems , 2000, 25(5): 323～343

11 Han J, et al. Mining Frequent Patterns without Candidate Generation, (Slide). In: Proc. 2000 ACM-SIGMOD Int. Conf. On Management of Data ,Dallas. Tx, 2000(5)

12 杨明，孙志挥. 基于前缀广义链表的快速关联规则挖掘算法. 小型微型计算机系统(已录用)