关联规则中最大频繁项目集的研究

李清峰[1,2]，杨路明[1]，张晓峰[2]

(1. 中南大学 信息科学与工程学院, 湖南 长沙 410083; 2. 湖南商学院 计算机与电子工程系, 湖南 长沙 410205)

摘　要: 研究了大型事务数据库中关联规则的频繁集问题; 提出一种高效挖掘最大频繁集的新算法 MMFI。该算法采用按事务数的层次和候选频繁集的维数处理的策略, 经数学证明和实验分析, 能大大减少判断运算量。

关键词: 数据挖掘; 关联规则; 最大频繁集; 数据库扫描法; 频繁树法

中图法分类号: TP301.6　　**文献标识码**: A　　**文章编号**: 1001-3695(2005)01-0093-03

Study for Mining Maximally Frequent Item Sets in Association Rule

LI Qing-feng[1,2], YANG Lu-ming[1], ZHANG Xiao-feng[2]

(1. *College of Information Science & Engineering, Centresouth University, Changsha Hunan* 410083, *China*; 2. *Dept. of Computer & Electronic Engineering, Business College of Hunan, Changsha Hunan* 410205, *China*)

Abstract: Study the frequent item sets problem for association rule in large business database; propose an efficient new algorithm MMFI in mining maximum frequent item sets. The idea of MMFI is to divide database by level and divide candidate frequent item sets by number. This algorithm is proved that it is efficient to reduce time in compute by mathematics and expriments.

Key words: Data Mining; Association Rules; Maximum Frequent Item Sets; Scanning Database Method; Frequent Pattern Free Method

关联规则是由 Agrawal 等人提出的一个重要的数据挖掘研究课题, 它已在分类设计、交叉购物、附加邮寄、目录设计等方面得到了广泛的应用。挖掘频繁项目集是关联规则等多种数据挖掘应用中的关键问题。然而, 在挖掘中候选项目集产生的代价是很高的, 尤其是存在大量强模式、长模式和要求阈值较低的时候。由于最大频繁项目集已经隐含了所有频繁项目集, 所以可把发现频繁项目集的问题转换为发现最大频繁集的问题。另外, 某些数据开采应用(如发现最小主键等)仅需发现最大频繁集, 而不必发现所有的频繁集[2]。因而发现最大频繁集对数据开采具有重大意义。

近年来, 在关联规则中频繁项目集算法的研究中先后出现了 Apriori[1], DMFI[2], DMFIA[3], FP-Tree[6], FPST[10], 改进的 FP-树[12], FIUA[13] 等算法。在众多算法中大致可分为两类:

(1) 对候选频繁集进行剪枝, 采用多次对数据库自底向顶和/或自顶向底搜索的策略。其思想是及早尽量多地剔除实际不可能产生频繁集的候选频繁集, 减少分析判别和运算的次数。这类算法主要有 Apriori, DMFI。

(2) 将数据库投影到一棵频繁模式树 FP-Tree 上, 仍保留项目集的关联信息, 然后通过扫描 FP-Tree 中路径节点的计数来搜索频繁项目集。这类算法以 DMFIA, FP-Tree, FPST, 改进的 FP-树, FIUA 为代表。为了介绍方便, 我们把这两大类方法分别称为数据库扫描法和频繁树法。

经分析以上两类的诸多算法都存在一定的缺陷。前类对数据库要多次扫描, 这使计算时间势必增大, 然而扫描一次可以获取数据库的全部信息。这种方法显然没有存储利用前面扫描获取的信息, 因此可以对原数据库增加一个计数属性以便存储后面判断分析所需的信息, 从而减少扫描的次数和减小范围。后类算法的主要优点是扫描数据库的次数少, 算法效率高(实验数据也显示其运算时间比前一类算法的时间短)。但当数据库很大时, 尤其是存在大量强模式和长模式时, 构造的 FP-Tree 将很庞大, 而且大量 FP 树节点链将增加数据结构的复杂度和资源的占用; 树枝伸展的无序将降低模式检索和挖掘的效率; 通过条件模式基的分析产生频繁模式仍需要进行大量的连接操作[10]。本文提出一种属于数据库扫描法的新算法 MMFI, 将原来对数据库的事务和全体候选频繁集整体处理的方法分解为按事务的层次(根据事务中项目数)和候选频繁集的维数(根据项目集中的项数)处理的策略, 经数学证明和实验分析, 能大大减少判断运算量。

收稿日期: 2004-01-19; 修返日期: 2004-04-04

1　问题的描述

事务是指数据库 D 的一个记录: 项目(Item)是一个文字, 在交易数据库中代表商品, 在分类时代表属性的值。项目集是指事务中项目的集合。

设 $I=\{i_1, i_2, \cdots, i_m\}$ 是 m 个不同项目的集合。给定事务数据库 D, 对于项目集 $X \in I$, X 在 D 中的支持数是指 D 中包含 X 的事务数, 记为 $X.s$。如果 X 的支持数不小于用户给定的最小支持数阈值 S, 则称 X 为 D 中的频繁集。对于项目集 $X \in I$, 如果 $X.s \geq S$, 并且对于 X 的任意超集 Y, 均有 $Y.s < S$, 则称 X 为 D 中的最大频繁集。

设数据库 D 有 m 个项, 事务中含最多的项目数为 $t(t \leq m)$ 个, 则它的各维候选项集之和 $M = Cm^1 + Cm^2 + \cdots + Cm^t$; 设 D 中总的事务数为 n, $n = n_1 + n_2 + \cdots + n_j + \cdots + n_t$, 这里 n_j 表示含 j 个项目的事务数。挖掘频繁集的原始思路是用每个候选

频繁项集来扫描 D,统计其出现次数,次数不小于最小支持数阈值 S的便是频繁集。其扫描判别数为 $M \times n$,其后众多学者根据 Apriori 性质[4],对 M进行剪枝,及早尽量多地剔除实际不可能产生频繁集的候选频繁集。然而对未知的大型数据库来说,M是一个非常庞大的数据,对 M的精简是有限。

这里我们提出的 MMFI 算法,采用对数据库分层和对候选项集分维的技术。其思想是根据 2.1 中的性质(3):只有项目数不小于 j的事务才可能对 j维频繁集作出贡献。因此先对数据库 D按事务中项目数降序排列,具有相同项目数的事务合并,然后自顶向底对 D分层扫描,先求出顶层事务(该层的所有事务都包含 t个项目)中产生的最大频繁项集的集合和非频繁项集的集合,分别存入最大频繁集数据表和候选频繁集数据表中(此两数据表是另外新建的);由于包含 t项的非频繁集可能对小于 t项的频繁集的产生作出贡献,然后将所有 t项非频繁集的 $t-1$项子集(作为 $t-1$项的候选频繁集)求出,再扫描 D中第二层事务(该层的所有事务都包含 $t-1$个项目),求出项目数为 $t-1$的最大频繁项集的集合和非频繁项集的集合;这样依此类推求得包含所有项目数的最大频繁集。其运算量最多为 $n_t \times C_m^t + n_{t-1} \times C_m^{t-1} + \cdots + n_1 \times C_m^1 << M \times n = (C_m^1 + C_m^2 + \cdots + C_m^t) \times (n_1 + n_2 + \cdots + n_t)$。经实验数据分析,比目前效率最高的算法改进的 FP-树[12]的效率提高约 30%。

2 MMFI 算法的表述与分析

2.1 有关频繁集具有如下性质

(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。

(3) 只有项目数不小于 j的事务才可能对五维频繁集作出贡献,或者说 j维频繁集的诞生与项目数小于 j的事务无关。

2.2 技术路线

(1) 对数据库 D增设一个记数属性和一个项目数属性,分别存储事务出现次数和事务中项的维数;扫描 D一次,对每个项目出现的次数统计,对每个事务中包含的项目数存入项目数属性中。

(2) 剔除出现次数小于用户给定的最小支持度阈值 S的项目,因为根据性质(2)它不可能对任何频繁集作出贡献。

(3) 对数据库 D根据事务中项目数分层,具有相同项目的事务合并,将合并数计入记数属性中,没有合并的计数值为 1(表示出现一次),然后根据数据库分层的结果降序排列。设 D中事务包含的项目数最多为 $t(t \leq m)$,事务总数 $n = n_t + n_{t-1} + \cdots + n_j + \cdots + n_2 + n_1$,这里 n_j表示第 $t-(j-1)$层含 j个项目的事务数。

(4) 增设两个二维表 Fitem 和 Citem 分别存储已判明为最大频繁集的数据,该扫描层候选频繁项集数据。

(5) 先扫描 D的第一层,这层有 n_t个事务(每个事务都包含 t个项目),如果计数值不少于 S,可判别为最大频繁集,则将该事务项加入 Fitem;其余的计数值小于 S则求取其 $t-1$维所有子集加入 Citem 中(根据性质(3)),含 j项的非频繁集可能对产生小于 j项的频繁集作出贡献,同时对 Citem 中的项目集计数,其值为它们父集的计数值;然后将 Citem 中相同项目集合并,合并后计数值为相同项目集计数值之和,再扫描 Citem,判断其中的项目集的计数值是否不小于 S,若有而且是 Fitem 的子集,则将该项目集从 Citem 中删除;如果不是 Fitem 的子集,它一定是新出现的最大频繁集,则将该项目集加入 Fitem,同时也将该项目集从 Citem 中删除;最后扫描下一层。

(6) 扫描 D的第 j层,这层有 $n_{t-(j-1)}$个事务(每个事务都包含 $t-(j-1)$个项目)。①先用每个事务扫描 Fitem,若是 Fitem 的子集,则扫描下个事务;否则再扫描 Citem,如果 Citem 中没有与事务相同的项目集,则将该事务项目集及计数值加入 Citem;若有相同项目集,则项目集计数值为两者计数值之和。②该层扫描完后,再扫描 Citem,判断其中的项目集的计数值是否不小于 S,若有而且是 Fitem 的子集,则将该项目集从 Citem 中删除;如果不是 Fitem 的子集,它一定是新的最大频繁集,则将该项目集加入 Fitem,同时也将该项目集从 Citem 中删除。③将 Citem 中所有项目集(含 $t-(j-1)$个项目)的 $t-j$维子集求出,相同项目集合并,计数值为合并项目集的计数值之和。以此 $t-j$维项目集的集合更新 Citem 中的内容。④再扫描 Citem,判断其中的项目集的计数值是否不小于 S,若有而且是 Fitem 的子集,则将该项目集从 Citem 中删除;如果不是 Fitem 的子集,它一定是新的最大频繁集,则将该项目集加入 Fitem,同时也将该项目集从 Citem 中删除,然后扫描下一层。

(7) 扫描 D的最后一层后,Fitem 中的项目集的集合便是我们所求的所有最大频繁项集。

2.3 算法分析

这种方法的思想是采用自顶向下的策略,从第一层开始依次求取下一层的频繁集和候选频繁集。我们来分析这种技术的运算量:先分析数据库 D第 j层扫描,这层有 $n_{t-(j-1)}$个事务,这时 Fitem 中存储的是项目数大于 $t-(j-1)$的所有最大频繁集,这些最大频繁集的 $t-(j-1)$项子集的数量必定不小于最大频繁集的数量;而数据表 Citem 存储的是实际可能出现的 $t-(j-1)$项候选集的集合,它的最大数量是 $C_m^{t-(j-1)}$减去所有最大频繁集的 $t-(j-1)$项子集数,也就是说 Fitem 和 Citem 中项目集的数量之和应少于 $C_m^{t-(j-1)}$。因此第 j次扫描的最大运算量应小于 $n_{t-(j-1)} \times C_m^{t-(j-1)}$。总的 t次扫描的最大运算量应小于 $n_t \times C_m^t + n_{t-1} \times C_m^{t-1} + \cdots + n_1 \times C_m^1 << M \times n = (C_m^1 + C_m^2 + \cdots + C_m^t) \times (n_1 + n_2 + \cdots + n_t)$。当然在第 j次扫描时,要从 Citem 中的 $t-(j-1)$项集中求其 $t-j$项的子集,并且这些子集要对 Fitem 扫描,这要附加一些开销。经实验数据分析,总的效率是比较高的,比目前效率最高的算法改进的 FP-树[12]的效率提高约 30%。

2.4 MMFI 的算法描述

输入 数据库 D,用户给定的最小支持数阈值 S。

输出 D的最大频繁集的集合。

预处理:

(1) 剔除 D中支持数小于 S的一次项目。

(2) 在 D中增设属性 D.item,统计 D的事务中项目数。

(3) 在 D中增设属性 D.count,表示 D的事务出现次数,初始值为 1(表示出现一次)。

(4) 对 D.item 进行降序排列,具有相同项目的事务合并,同时 D.count 为合并数。

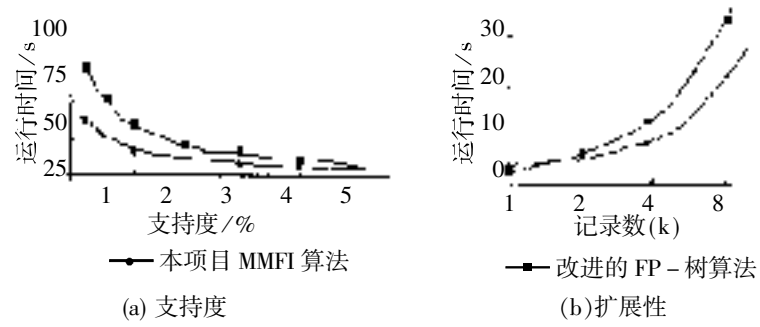
处理方法:

(1) fitem = { }, citem = { }; //设两个数据表,分别存储已判明为

```
//频繁集的数据、该扫描层候选频繁集项数据,初始值都为空
(2) for  i = t  to  1  step-1; //自顶向下扫描数据库
(3) do  while D.item = i; //对数据库每层扫描
(4) if  D.transaction ∈ fitem; //判断 D 中的当前事务是否是 fitem
//中项集的子集
(5) skip; //扫描下一条事务
(6) else
(7) if  D.transaction = citem; //citem 中是否有与当前事务相同的
//项目集
(8) citem.count = citem.count + D.count; //D 中的当前事务的计数
//值加入到 citem 中与之相同的项目集的计数中
(9) else
(10) add  D.transaction  to citem; //将 t 添加到 citem 中
(11) endif
(12) skip
(13) endif
(14) enddo
(15) citem = organ(citem); //一层扫描完调用 organ 过程
(16) citem = subtem(citem); //一层扫描完调用 subtem 过程
(17) endfor
(18) output fitem; /* 输出最大频繁集
procedure  organ(citem); /* 其功能为判断 citem 中是否出现频繁
集,若为新的最大频繁集则加入 fitem,同时在 citem 中删除该频繁集;
若频繁集在 fitem 中已存在,则在 citem 中删除该频繁集* /
(1) scan citem; //循环扫描 citem 中的每条记录
(2) if citem.count≥s; //判断 citem 当前项目集是否为频繁集
(3) if citem.items ¢fitem; //如果 citem 当前项目集没包含在 fitem
//中
(4) add citem.items to fitem;
(5) endif;
(6) delete citem.items; //删除 citem 当前记录
(7) endifl;
(8) endscan
(9) return citem;
procedure subset(citem); /* 其功能为将 citem 中 j 项目集的 j - 1
项子集及计数值更新 citem 中的内容,相同项目集合并,计数值相加,同
时判断 citem 中是否出现频繁集,若有新的最大频繁集则加入 fitem,同
时在 citem 中删除该项目集* /
(1) tempt = { }, middle = { }; //设两个临时数据表,初值为空
(2) scan citem; //循环扫描 citem 中的每条记录
(3) j = length(citem.items); //j 为 citem 中当前项目集的项目数
(4) for I = 1  to  j; 扫描 citem 中当前项目集的每个项目
(5) middle = delete  citem.item[I] from citem.items; //middle 存储
//当前 j 项目集的 j - 1 项子集
(6) if middle  ¢tempt; //j - 1 项子集没在 tempt 中
(7) add middle to tempt;
(8) else tempt.count = tempt.count + middle.count;
(9) clear middle; //清空 middle
(10) endif;
(11) endfor;
(12) endscan
(13) tempt = organ(tempt);
(14) return tempt.
```

3　算法实现与比较

用VC++6.0 在内存 64MB, CPU 为 Pentium Ⅲ 450MHz, 操作系统为 Windows 98 的联想奔月 2000 机上进行仿真演算。使用与文献[2, 3, 12] 同样的测试数据库, 该数据库有 8 124 条记录, 记录了蘑菇的 23 种属性, 每种属性有 2 ～12 个枚举值。用不同算法的程序计算不同的最小支持度(分七档: 5%, 4%, 3%, 2%, 1%, 0.5%, 0.25%) 的频繁项目集所用的时间, 比较不同算法的效率。图 1(a) 描绘了目前发表的效率最高改进的 FP-树[12] 算法与本项目 MMFI 算法在不同支持度下运算时间的比较。从图 1(a) 中可看出最小支持度越低, 候选集将越多, MMFI 的优越性越明显。同时为了检验不同数据量下算法的扩展性, 固定最小支持度(1%), 随机从范例数据库中抽取四个不同的记录数(1 000, 2 000, 4 000, 8 000) 进行测试。图 1(b) 是本项目 MMFI 算法与改进的 FP-树算法扩展性的比较。虽然对小数据库改进的 FP-树算法时间短(树枝和节点数少), 随着数据的增多 MMFI 的相对效率就越高。



(a) 支持度　　(b)扩展性

图 1 FP–树算法与 MMFI 算法之支持度比较

4　问题的展望

本文基于数据库扫描法提出一种求最大频繁集的高效算法 MMFI。该算法将运算量由 $M \times n$ 精简为 $n_t \times c_m^{\,t} + n_{t-1} \times c_m^{\,t-1} + \cdots + n_1 \times c_m^{\,1}$, 效率有较大幅度提高。但我们认为该问题的最理想解决方案应是 $M + n$(当然应对 M 进行最精练剪枝), 即只扫描数据库一次且不产生候选频繁集方案。其理论依据是扫描一次可以获取数据库的全部信息, 然后再分析实际出现的项目集并统计其出现次数, 次数不小于 S 的便是频繁集。但真正要达到 $M + n$ 的计算量不太现实, 因为这是个 NP 问题, 只可逼近这一目标。我们的初步想法是借助频繁树法的思想, 采用图论的知识来达到目的。其理论依据是如果只对 D 扫描一次, 那么应将扫描所得 D 的全部信息映射到一个图中, 同时保持每个事务中项目的相关性, 便于计算机的判断识别。经分析频繁树法就是这种思想, 其图便是一棵树, 每个事务中项目的相关性体现在路径中(其缺陷在前面已介绍), 即用线段(由树枝组成) 表达相关性; 为了避开其缺陷, 用面或立体图表达数据库中事务的信息和项目的相关性可能是我们探索该问题的方向之一。

参考文献:

[1] Agrawal, *et al*. Mining Association Rules Between Sets of Items in Large Databases[C]. Washington: Proceedings of the ACM SIGMOD International Conference Management of Data, 1993. 207-216.

[2] 路松峰, 卢正鼎. 快速开采最大频繁项目集[J]. 软件学报, 2001, 12(2): 293-297.

[3] 宋余庆, 朱玉全, 等. 基于 FP-Tree 的最大频繁项目集挖掘及更新算法[J]. 软件学报, 2003, 14(9): 1586-1592.

[4] Han J, Kamber M. Data Mining: Concepts and Technique[M]. Beijing: High Education Press, 2001. 149-184.

[5] J Han, J Pei B. Mortazavi-Asl: Frequent Pattern-projected Sequential pattern Mining[C]. Proc. 2000 Int. Conf. knowledge Discovery and Data Mining(KDD'00), Boston, MA, 2000.

[6] Han J, Jian P, Yiwen Y. Mining Frequent Patterns Without Candidate Generation[C]. Proceedings of the 2000 ACM SIGMOD International Conference Management of Data. Dallas, 2000. 1-12.

[7] Agrawal R, Srikant R. Fast Algorithm for Mining Association Rules [C]. Proceedings of the 20th Internation Conference on VLDB, Santiago, 1994. 487-499.

[8] 袁鼎荣, 张师超. 基于频繁链表的频繁集的挖掘算法[J]. 计算机科学, 2003, 30(7): 165-166.

(下转第 98 页)

```
    For each HTptr 的儿子节点 HTSanptr do
      HTptr = HTSanptr
      If Tptr 的儿子节点中有标号为 HTptr. pid 的节点 then
          Tptr 指向该儿子节点
          Tptr. count = Tptr. count + HTptr. count
          Tptr. coe = Tptr. coe + HTptr. coe
      Else
        为 Tptr 创建儿子节点, Tptr 指向该儿子节点
        Tptr. pid = HTptr. pid
        Tptr. count = HTptr. count
        Tptr. coe = HTptr. coe
      Endif
      MergeTree( Tptr, HTptr)
  Loop
 Endif
End function
```

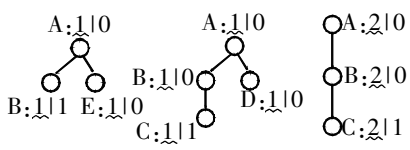


图 3 THAP – 树

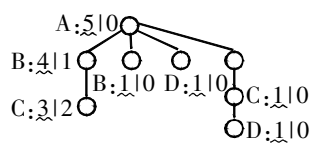


图 4 HAP – 树

2.4 SSP-Output 算法

根据 Make-HAP-Tree 算法, 得出关于 HAP-树的如下两个性质:

性质 1 在一棵挖掘 ξH^n-SSP 的 HAP-树上, 如果节点 N 的子孙中没有 H 节点, 则从根节点到 N 节点的路径是一个 H^n-SSP, 且该 H^n-SSP 在 WTDB 中的计数是 N 节点的 Count 值。

性质 2 在一棵挖掘 ξH^n-SSP 的 HAP-树上, 如果从根节点到节点 N 的路径构成了一个 ξH^n-SSP, 则只有当 N. coe$\geq \xi$ 时, 该 ξH^n-SSP 才是 Seed。

由性质 1 和 Seed 的定义, 性质 2 的结论是显然的。以图 4 的 HAP-树为例, 假设规定 ξ= 2, 则输出到 L 中的 ξA-SSP 为 {A∶5}, {AB∶4}, {ABC∶3}, 而可以作为 Seed 的只有 {ABC}。算法 SSP-Output 的内容如下:

```
输入  HAP-树
输出  ξH^n-SSP 和 Seed-Set
Seed-Set = Φ      //清空 Seed-Set
For each 节点 N do
  if 由根节点到 N 节点的路径构成 H^n-SSP and N.count∈ ξ then
    将该 H^n-SSP 加入 L
    if N. coe≥ ξ then
      将该 H^n-SSP 加入 Seed-Set
    end
end
```

按照 PS2 算法的上述规定, 每一次循环扫描只为 L_1 中页面各生成一棵 HAP-树, 实际上经过两次循环后, 搜索空间可能已经大大缩小, 可以有选择地越过一些循环, 如拆分 AP-树后, 同时为 H^2-SSP, H^3-SSP, H^4-SSP…构造 HAP-树。

3 PS2 算法的性能分析

PS2 算法的性能好于现有的基于 Apriori 的算法, 这是因为 PS2 算法是基于内存的, 只需一次扫描数据库, 而 Apriori 的算法需多次扫描数据库, 且 Apriori 算法每次扫描数据库只使发现的频繁访问路径增加一个页面长度, 而 PS2 算法则使其增加一个扩展子路径的长度。

为了验证 PS2 算法的性能, 对 PS2 算法和类 Apriori 算法的 GSP 算法[2]进行了对比试验, 用 VC + + 实现算法, 试验环境为 Intel P 4 1. 8GHz, 256MB RAM, Windows 2000 Professional, 数据集取自东北林业大学校园网站 http: //www. nefu. edu. cn 的 Weblog, 将 WTDB 中的事务数由 100k 条逐渐增至 500k 条。两种算法的性能曲线如图 5 所示。

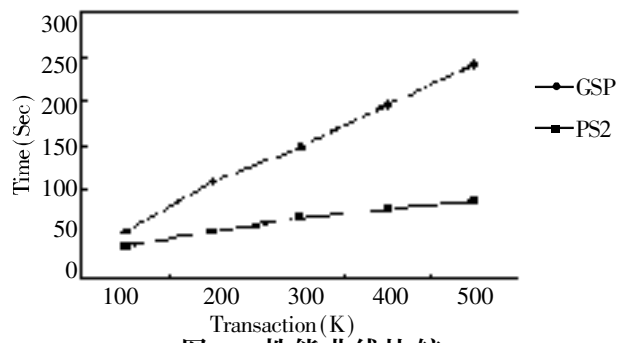


图 5 性能曲线比较

4 结论

PS2 算法和现有的类 Apriori 算法相比具有速度快、效率高的优点, 缺点是可扩展性差, 原因是当数据量很大时, 内存可能无法容纳整棵 AP-树, 从而使该算法的性能变坏。该算法也可用于其他类似问题, 如连续频繁字符串发现等。

参考文献:

[1] Agrawal, R Srikant. Mining Sequential Patterns[C]. Proceedings International Conference on Data Engineering, 1995. 3-14.

[2] M S Chen, J S Park, P S Yu. Efficient Data Mining for Path Traversal Patterns[J]. IEEE Transactions on Knowledge and Data Engineering, 1998, 10(2): 209-221.

[3] J Borges, M Levene. Mining Association Rules in Hypertext Databases [C]. Proceedings Conference on Knowledge Discovery and Data Mining, 1998. 149-153.

[4] J Pei, J Han, *et al*. Mining Access Patterns Efficiently from Web Logs [C]. Proceedings Pacific-Asia Conference on Knowledge Discovery and Data Mining, Kyoto, Japan, 2000. 396-407.

[5] Cooley R, *et al*. Discovery of Interesting Usage Patterns from Web Data[C]. Berlin: International WEBKDD'99 Workshop Springer Verlag, 2000. 163-182.

[6] J Han, *et al*. Mining Frequent Patterns Without Candidate Generation [C]. Proc. ACM-SIGMOD Int. Conf. Management of Data, Dallas, TX, 2000. 1-12.

作者简介:

战立强(1966-), 男, 博士研究生, 研究方向为数据库与知识库、数据挖掘; 刘大昕(1941-), 男, 博士生导师, 研究方向为数据库与知识库。

(上接第 95 页)

[9] 朱玉全, 孙志挥, 赵传申. 快速更换频繁项集[J]. 计算机研究与发展, 2003, 40(1): 94-99.

[10] 惠晓滨, 张凤鸣, 虞健飞, 等. 一种基于栈变换的高效关联规则挖掘算法[J]. 计算机研究与发展, 2003, 40(2): 330-335.

[11] 邹晓峰, 陆建江, 宋自林. 基于模糊分类关联规则的分类系统[J]. 计算机研究与发展, 2003, 40(5): 651-656.

[12] 范明, 李川. 在 FP-树中挖掘频繁模式而不生成条件 FP-树[J]. 计算机研究与发展, 2003, 40(8): 1216-1222.

[13] 朱玉全, 孙志挥, 季小俊. 基于频繁模式树的关联规则增量式更新算法[J]. 计算机学报, 2003, 26(1): 91-96.

作者简介:

李清峰(1975-), 男, 硕士研究生, 主要研究方向为数据挖掘、图像处理; 杨路明(1947-), 男, 教授, 博士生导师, 主要研究方向为计算机网络、数据库技术。