基于矩阵的频繁模式挖掘及更新算法

何海涛　张世玲
（燕山大学信息科学与工程学院　秦皇岛 066004）

摘　要　频繁模式挖掘在数据挖掘领域已经有广泛的应用。然而，对于增量更新频繁模式挖掘研究得不是很多。本文提出了一种新颖的增量更新频繁模式树结构(IUNP_Tree)，构建它只需要对数据库扫描一次。此外，提出了基于条件矩阵(conditional matrix)的频繁模式挖掘算法(FPBM_Mine)和增量更新算法 INUPA，可以有效地处理数据库的增量更新问题。实验表明，该算法是有效的，并且运行效率高于 FP-growth 算法。

关键词　数据挖掘，增量更新，频繁模式挖掘

Frequent Patterns Mining and Incremental Updating Algorithm Based on Matrix

HE Hai-Tao　ZHANG Shi-Ling
(College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004)

Abstract　Frequent patterns mining has been studied popularly in KDD research. However, little work has been done on incremental updating frequent patterns mining. A novel incremental updating pattern tree (INUP_Tree) structure is presented in this paper, which is constructed by scanning database only once. Besides, a new frequent pattern mining method (FPBM_Mine) based on conditional matrix and incremental updating algorithms INUPA are developed. The experiment result shows that the FPBM_Mine method is more efficient and faster than the FP-growth.

Keywords　Data mining, Incremental updating, Frequent pattern mining

1　引言

关联规则是由 Agrawal[1] 等人首先提出来的，目前已经成为数据挖掘的一个重要研究方向。在众多算法中，以 Agrawal[2] 等人提出的 Apriori 算法最为著名。在实时数据库中，随着时间的变化，一些新的数据会不断地添加到数据库中，因此数据库的维护问题引起了研究人员的关注。D. W. Cheung 首先研究了这个问题，并提出了 FUP 算法[3]。然而，这些算法都是基于 Apriori 思想的，大多需要产生大量的候选集，并且可能需要重复扫描数据库。Agarwal 提出 TreeProjection 算法[4]，采用字典树作为数据存储结构，主要采用了广度优先的策略来建立树，并与深度优先的策略相结合，进行事务投影和计数。虽然该算法比 Apriori 算法快了一个数量级，但是当数据库规模很大时，该算法在投影和字典树的构造上的开销是非常巨大的。Han 等人提出了 FP-Tree 和相应的 FP-Growth 算法[5]，该方法对于挖掘长的和短的频繁模式，都是有效的。但是 FP-Tree 结构存在动态维护复杂、不利于频繁项集更新等缺点。

本文提出了适合增量更新的增量更新模式树结构 INUP_Tree (Incremental Updating Tree)。构造 INUP_Tree 时只需扫描数据库一次，并且提出了基于条件矩阵的频繁模式挖掘方法 FPBM_Mine (Frequent Pattern mining method Based on Matrix)。此外，当最小支持度和数据库发生变化后，针对数据库的更新问题分别提出了算法 $INUPA_1$ 和 $INUPA_2$ (Incremental Updating Algorithm)，大大提高了频繁项集的挖掘和维护效率。

2　问题描述

2.1　频繁模式挖掘

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是一个数据项集，$DB=\{T_1,T_2,\cdots,T_n\}$ 是一个交易数据库，每条交易 T 对应于一个数据项子集，即 $T\subseteq I$。每条交易有一个 TID(transaction identifier)标识。若 DB 的总交易数为 N，DB 中包含 X 的交易数为 s，则项目集 X 的支持度为 s/N，记为 min_sup(X)。如果 min_sup(X) 大于等于某个给定的最小支持度阈值 min_sup，则称 x 为频繁模式。给出一个数据库 DB 和一个最小支持度 minimum support，找出所有频繁模式的问题称为频繁模式挖掘问题。

2.2　INUP_Tree 的定义及性质

INUP_Tree 与频繁模式树 FP-Tree 非常类似，只不过 INUP_Tree 记录的是 DB 中的每个事务的所有项，而 FP-Tree 只记录 DB 中的每个事务中的频繁项。FP-Tree 中事务的项按支持度降序排列，而 INUP_Tree 中的事务按某种规则顺序存储。本文中采用字典顺序排序，并且头表中增加了一个 flag 域，用来标志对应的项是否是频繁项。

例 1　设事务数据库 DB 如表 1 所示，最小支持度为 40%，数据库 DB 的 INUP_Tree 如图 1 所示。

表 1　事务数据库 DB

TID	Contents
T1	{a,d,b,g,e,c}
T2	{d,f,b,a,e}
T3	{a}
T4	{d,a,b}
T5	{a,c,b}
T6	{c,b,a,e,f}
T7	{a,b,c}
T8	{a,b,c}

性质 1　INUP_Tree 的结构不会因为支持度的改变而改变。

性质 2　对于每个项 X_i，从头表的 head of node_link 链

何海涛　副教授，研究领域：数据挖掘、计算机智能控制；张世玲　硕士研究生，研究方向：数据挖掘。

出发通过遍历 node_link 可以找到所有包含 X_i 的路径。

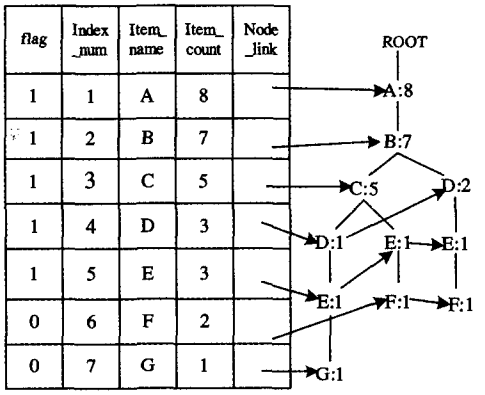

图 1　DB 的 INUP_Tree

由于 INUP_Tree 不是一棵频繁树，并且事务项按字典顺序排序，所以树的结构不会因为最小支持度的改变而改变。node_link 链接了所有具有相同 item_name 的结点，所以通过遍历 node_link 可以找到所有包含 X_i 的路径。

3　频繁模式挖掘算法(FPBM_Mine)

3.1　条件矩阵的构造

建立 INUP_Tree 后，需要扫描头表，找出频繁 1-项集，然后为每个频繁项构建条件矩阵。一个频繁项 I_i 的条件矩阵用来存储以 I_i 为后缀的所有路径。行表示 I_i 在路径中的局部支持度。第一列表示 I_i 的前缀路径中的项的索引号，第一列之外的每一列代表一条前缀路径。

定理 1　对于任何频繁项 X_i，它的条件矩阵包含所有以 X_i 为后缀的路径。

证明：根据性质 2，可以找到所有包含 X_i 的路径。对 node_link 链由左向右依次遍历，直到 node_link 为空，把遍历到的每一条路径依次投影到条件矩阵中作为一列。所以，条件矩阵包含了所有以 X_i 为后缀的路径，并且不会有遗漏和重复。

以构造项{e：3}(索引号=5)的条件矩阵为例，如图 2 所示。e 的前缀路径为 a：8-b：7-c：5-d：1-e：1 和 a：8-b：7-c：5-e：1 和 a：8-b：7-d：2-e：1。第二列的值为路径 a：8-b：7-c：5-d：1-e：1 中 e 的支持度的值。第三列的值为路径 a：8-b：7-c：5-e：1 中 e 的支持度的值。由于此路径中不包含项 d，所以第三列第四行的值为'0'。同样地，第四列的值为路径 a：8-b：7-d：2-e：1 中 e 的支持度的值，路径中不包含项 c，所以第四列第三行的值为'0'。

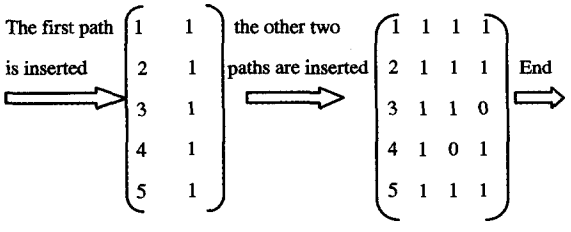

图 2　E 的条件矩阵

如果路径中某项的 index_num 不包含于条件矩阵的第一列中，则调用过程 Insert_M(X_s, j)。Function Insert_M(X_s, j)：

1. if a frequent item X_s with index_num s in path j doesn't exist in the first column
2. add a new row into M in numeral order according to s; //假设它是第 r 行
3. the value of $M[r-1][0]=s$;
4. the value of $M[r-1][1\cdots j-1]=0$;
5. the value of $M[r-1][j]=X_s$. count in path j;
6. End

3.2　频繁模式挖掘算法 FPBM_Mine

FPBM_Mine 算法的挖掘过程是在条件矩阵上进行逻辑运算，不需要占用额外的内存空间。

定理 2　在挖掘以 X_i 为后缀的频繁模式时，只需要计算条件矩阵中 X_i 所在行之上的每一行的值。

证明：假设有一个项 $X_l(i<l<n)$ 位于 X_i 行以下并同时出现在列中，由于 FPBM_Mine 算法采用由底向上的方法挖掘条件矩阵，所以以 X_l 为后缀的频繁模式已经被挖掘出。因此，只需要计算 X_i 所在行之上的行的值。由此定理可知，FPBM_Mine 算法不会重复挖掘频繁模式。

算法　FPBM_Mine(condtitional matrix $M_{n\times m}$, X_i)

输入：the condtitional matrx $M_{n\times m}$ of item X_i
输出：Frequent patterns ∂ with the suffix of X_i

1. If $m=1$ then
2. For each combination of items (registered as a_i) in the column
3. Generate frequent patterns $\partial=a_i\cup X_i$ and ∂. count$=X_i$. count
4. Else
5. Sum up the values in each column above X_i and get column L // 逻辑操作
6. For ($r=0$; $r<$the length of row in L; $r++$)
7. {if L_r. count$\geqslant$min_sup|DB| then
8. {generate frequent pattern $\partial=L_r\cup X_i$ and ∂. count$=X_i$. count}
9. Else
10. {delete the row of L_r;}//删除第 r 行
11. }
12. While ($M_{n'\times m}$! =null) do
13 调用 FPBM_Mine(conditional matrix $M_{n'\times m}$,∂);
14. End

挖掘{e：3}的频繁模式的过程如图 3 所示。

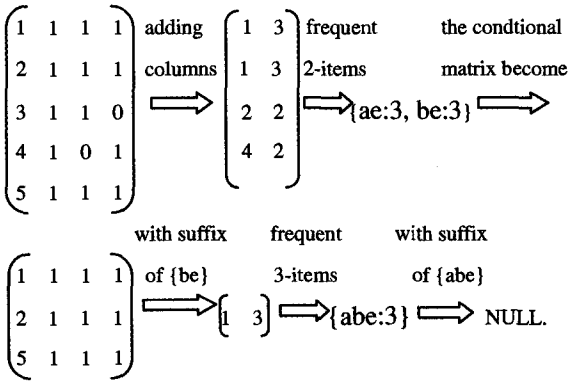

图 3　以 E 为后缀的频繁模式挖掘过程

如图 3 所示，第三行的支持度之和小于最小支持度 3，所以根据算法 FPBM_Mine，删除第三行。同理，第四行也被删除。挖掘出频繁 2-项集{ae：3，be：3}，再以{be：3}为后缀，挖掘它的频繁模式为{abe：3}，以{abe：3}为后缀的频繁模式为空。再挖掘以{ae：3}为后缀的频繁模式，为空。到此，以{e:3}为后缀的所有频繁模式挖掘完毕，即{e：3，be：3，ae：3，abe：3}。

4　增量更新算法

给定事务数据库 D，针对实际应用需求，关联规则的更新问题可以分为如下两种情况，下面我们分别讨论两种情况。

情况 1：如果事务数据库 D 不变，最小支持度 s 改变，设新的最小支持度为 s'，那么分为两种可能：(1)$s>s'$；(2)$s<s'$。对于 $s<s'$，表明原频繁模式中的部分频繁模式是不符合最小支持度的，那么直接删除最小支持度小于 s' 的频繁模式即可。这种情况比较简单，在此不再赘述。我们来具体分析 $s>s'$ 的情况。

根据性质 1，INUP_Tree 的结构不会因为最小支持度的

改变而改变，所以不需要重新建立 INUP_Tree，只需要扫描头表的 flag 域。如果 flag 为 0，则比较该项的支持度是否大于 s'；如果大于，则构建它的条件矩阵；如果小于，则继续扫描 flag。对于 flag 为 1 的项，不用再次挖掘。

算法：增量更新算法 $INUPA_1$

输入：INUP_Tree 和 s'
输出：新的频繁项集
1. For each item X_i in Header Table
2. if X_i. flag=1 then
3. continue; //如果 flag=1 说明 X_i 已经挖掘过
4. else
5. { if X'_i count>s' then
6. flag=1 and construct its conditional matrix}
7. End for

改写上例，假设 $s'=2$。扫描头表找到新的频繁项为{f：2}。需要构建它的条件矩阵并挖掘以 f 为后缀的频繁模式。挖掘结果为{f：2，ef：2，bf：2，af：2，bef：2，aef：2，abef：2，abf：2}。

情况 2：由于篇幅所限，我们只考虑当最小支持度不变，一个事务数据集 db 添加到事务数据库 D 中，如何生成事务数据库 D∪db 中的频繁项目集。我们将其分解为以下三个子问题：(1)找出 D 中不再生效的频繁项目集；(2)找出仍然生效的频繁项目集；(3)找出 D∪db 中新的频繁项目集。

我们只需扫描 db，将新数据集插入到 INUP_Tree 中，而不需要重新建立 INUP_Tree。通过调用增量更新算法 $INUPA_2$，来解决以上的三个子问题。

算法：增量更新算法 $INUPA_2$

输入：INUP_Tree，DB，db，s
输出：事务数据库 DB∪db 中的所有频繁项目集
1. 调用函数 insert([t| T_i],T)把新数据集插入到 INUP_Tree 中
2. For each item X_i in Header Table
3. if X_i. flag=1&&X_i. item_count>s then
4. { if X_i has conditional matrix then
5. update X_i's conditional matrix
6. else
7. construct X_i's conditional matrix
8. if X_i. flag=1&&X_i. item_count<s then
9. flag=0 && delete its frequent patterns
10. End for

5 实验结果和性能分析

我们用 VC ++6.0 在内存 512M、CPU 为 Pentium4 2.4GHZ、操作系统为 Windows XP 的环境下实现了 $INUPA_1$ 和 $INUPA_2$ 算法并进行了性能测试。利用蘑菇数据库(mushroom database)来进行实验。该数据库有 8124 条记录，记录了蘑菇的 23 种属性。图 4(a)显示了在不同的最小支持度下算法的性能比较。由于 INUP_Tree 的构建只需扫描数据库一次，而且在最小支持度发生变化时不需要改变 INUP_Tree，不需要任何插入、删除操作，flag 标志的设置缩小了搜索范围，提高了发掘新频繁项集的效率。因此 $INUPA_1$ 算法的执行时间比 FP-Growth 算法要少，图 4(a)说明了这一点。

为了验证增量更新算法 $INUPA_2$ 的有效性，随机抽取 8000 个记录，并分为两部分：原始 DB(4000 个记录)和新增 db(4000 个记录)。最小支持度为 2.5%，从 db 中抽取不同的记录数(500，1000，2000，4000)作为 db 的不同增量情况，对更新算法 $INUPA_2$ 进行测试，结果如图 4(b)所示。由于 $INUPA_2$ 算法只需扫描新增数据 db 一次插入到 INUP_Tree 中，而不需要重新建立 INUP_Tree，并且可以充分利用已有的条件矩阵来加速本次的挖掘，因此其效率极大提高。

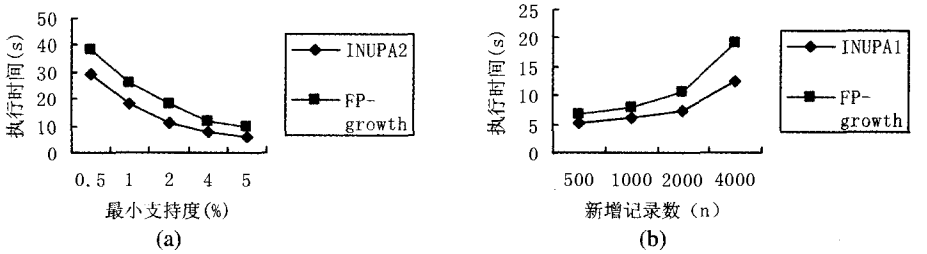

图 4 更新算法的执行时间对比

结束语 本文提出了一种新的增量更新频繁挖掘方法。使用 INUP_Tree 压缩存储数据库中的事务，它的建立只需要扫描数据库一次。同时提出了频繁模式挖掘方法 FPBM_Mine，以条件矩阵为基础进行挖掘，减少了搜索范围，加快了挖掘速度。针对不同的数据库增量更新问题，提出了 INUPA 算法，实验证明这种增量挖掘方法比 FP-growth 方法在处理增量问题时效率更高。

参 考 文 献

1 Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. In: Proc. of the ACM SIGMOD Conference, 1993. 207～216
2 Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proc. of VLDB Conf, 1994. 487～499
3 Cheung D W, Han J, Ng V T, el al. Maintenance of discovered association rules in large databases: An incremental updating approach. In: The 12th IEEE International Conference on Data Engineering, 1996. 106～114
4 Agarwal R, Aggarwal C, Prassad V V V. A tree projection algorithm for generation of frequent itemsets. Journal of Parallel and Distributed Computing, 2001. 61: 350～371
5 Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Proceeding of the ACM SIGMOD Conference, 2000. 1～12

(上接第 169 页)

和更小的内存需求的优点。特别是对于高维小样本集的文本分类问题，基于近似支持向量机能够适用于需要实时重构文本分类器的应用环境。

参 考 文 献

1 Fischer G , Otswald J. Knowledge management: problems, promises, realities, and challenges[J]. IEEE Intelligent Systems and Their Applications, 2001,16(1) :60 ～ 72
2 Kim S-B. Some Effective Techniques for Naive Bayes Text Classification[J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(11) :1457 ～ 1466
3 Vapnik V. The Nature of Statistical Learning Theory[M]. New York: Springer, 2000
4 庄东，陈英. 基于加权近似支持向量机的文本分类. 清华大学学报(自然科学版)，2005，45 (S1)：1787～1790
5 Lewis D D, Yang Y, Rose T G, et al. RCV 1: A new benchmark collection for text categorization research [J]. Journal of Machine Learning Research, 2004, 5(2): 361～397
6 Fung G, Mangasarian O L. Proximal support vector machine classifiers [A]. In: Proc. 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining [C]. San Francisco, CA , USA: ACM , 2001
7 温罗生，李泽民. 含有线性和非线性等式约束非线性规划问题的一种降维乘子法. 见：第七届中国运筹学大会论文集. 2005
8 http://www.icl.pku.edu.cn/icl_res/