# 基于反转矩阵的含重复项的关联规则挖掘算法*)

樊 征 柏文阳 徐洁磐

(软件新技术国家重点实验室 南京大学计算机科学技术系 南京210093)

**摘 要** 本文研究了多层关联规则挖掘中在由低概念层泛化至较高概念层后出现的含重复项规则的问题,在当前最新的高效关联规则发掘算法(IM)基础上引入了"数量"概念,提出了一种用于挖掘含重复项规则的新算法,并对其性能进行了分析。

**关键词** 数据挖掘,关联规则,反转矩阵

## Mining Association Rules with Recurrent Items Using Inverted Matrix

FAN Zheng BAI Wen-Yang XU Jie-Pan

(State Key Laboratory for Novel Software Technology, Nanjing 210093)

(Department of Computer Science & Technology, Nanjing University, Nanjing 210093)

**Abstract** This paper discusses the problem of discovering association rules with recurrent items. A novel algorithm based on inverted matrix is proposed. By introducing the conception of "quantity", this algorithm can discover association rules with recurrent items efficiently.

**Keywords** Data mining, Association rules, Inverted matrix

## 1 引言

关联规则的挖掘问题首先由 Agrawal 于1993年在文[1]中提出,在该文中 Agrawal 给出了经典的 Apriori 算法,之后出现了各种基于该算法原理的改进,如 DHP,DIC 和 FP-tree[2]等。最近,Mohammad 在文[3]中提出了一种基于反转矩阵的挖掘算法(inverted matrix algorithm,以下简称 IM 算法),该算法扫描两遍数据库,在处理大型数据库时有着优于 FP-growth 的表现,本文所提出的算法构建于该算法之上。

以上算法均用于挖掘原始层的关联规则。为避免采用低支持度在原始层中挖掘产生的开销急剧膨胀,常需要在高概念层中挖掘。此时产生一个新问题,在由低概念层向高概念层泛化时,会出现含重复项的项集。如:数据库中某事务包括"法式面包","花式面包"和"可口可乐"。泛化后的事务应该为"面包","面包"和"饮料"。但现在的许多挖掘算法会忽略事务中"面包"项出现的次数,事务被简化为"面包"和"饮料"。这样往往会遗漏一些重要的关联规则。例如:对于某疾病化验数据进行分析,当症状 $A_1A_2A_3$ 同时出现时,患疾病 B 的概率较大,而 $A_1A_2A_3$ 均被泛化为症状 A,显然规则 A(3)→B 比规则 A→B 更贴近于事实。

在本文中,我们要解决的就是以上忽略泛化后事务中的重复项可能引起的规则遗漏。我们引入了"数量"这一属性来记录一个项集的重复出现次数并提出了一种新的基于反转矩阵和 COFI-tree[3]的挖掘算法。

本文第2节中介绍算法需要用到的基本概念,第3节提出算法,第4节进行性能分析,最后对全文进行总结。

## 2 基本概念

对于事务数据库 $D$,由各事务$\langle T_i,\{A_p,\cdots,A_q\}\rangle$组成,其中 $T_i$ 为事务标志,而 $A_i\in I,(i=p,\cdots,q)$。$I$ 为数据库中所有的项(字段)的集合。

**定义1** 关联规则(模式)形如表达式:$R:A\rightarrow B$。其中 $A=A_1\wedge\cdots\wedge A_i, B=B_1\wedge\cdots\wedge B_j, A_1,\cdots,A_i,B_1,\cdots,B_j\in I$。

规则的支持度定义为 $D$ 中所有 $A\cup B$ 出现的次数,记为 $R_{sup}$;置信度定义为所有包含 $A$ 的事务中,同时包含 $B$ 的事务的比例,记为 $R_{con}$;规则为频繁的当且仅当规则的支持度和置信度均大于给定的最小支持度和置信度阈值 $S_{min}$ 和 $C_{min}$。

注意:这里的支持度定义有别于传统的支持度定义,Apriori 算法中每事务最多向模式提供1的支持,而在含重复项的情况下,每事务可能向模式提供多支持。定义如下:

$$R_{sup}=\sum_{i=0}^{|D|}\delta(X,T_i)/|D|$$

其中,$\delta(X,T_i)$为项集 $X$ 在 $T_i$ 中的支持度,定义见后文的定义4。

**定义2**[4] 给定一个项 $x$,我们定义它在事务 $T$ 或者项集 $X$ 中出现的次数(数量)为 $N(x,T)$或者 $N(x,X)$。

**定义3** 给定 COFI-tree 中两个项集 $X_i,X_j$,称其相似当且仅当它们由相同的项构成,记为 $S(X_i,X_j)$;如果 $S(X_i,X_j)$ 并且对于任意 $x(x\in X_i\wedge x\in X_j)$都有 $N(x,X_i)=N(x,X_j)$,则称 $X_i$ 和 $X_j$ 相等,记为 $E(X_i,X_j)$。

**定义4**[4] 设某事务 $T$ 支持一项集 $X$,则 $T$ 向 $X$ 提供的支持度为:

$$\delta(X,T)=\min\left(\left[\frac{N(y|S(y\in T,x_i\in X))}{N(x_i\in X)}\right]\Big|_{i=1}^{|X|}\right)$$

## 3 反转矩阵改进算法(IM')

本文在 IM[3]算法的基础上进行了改进,算法由三步组成,主要的改变在于因为定义2中为每个项集引入"数量"这一属性,新的算法则在老的 IM 基础上作了改动以适应新的数

*)基金项目:国家863计划项目(编号:2002AA141091)。樊 征 硕士研究生,研究方向为数据挖掘。柏文阳 副教授,研究方向为数据库安全,数据仓库。徐洁磐 教授,研究方向为数据库,知识库和人工智能。

据结构。以下首先给出算法的形式描述，再举一例解释该算法。

### 3.1 IM' 算法描述

步骤一：含重复项反转矩阵($M'$)的构造

输入：事务数据库 $D'$

输出：$M'$

算法流程：

1. 遍历 $D'$ 得到各原子项的支持度(定义1)
2. 将所有的原子项按升序排列
3. 根据2的结果创建 $IM'$ 的索引部分
4. 再次扫描 $D'$，对每个事务 $T$ 重复5到6；
5. 将 $T$ 中的重复项合并，按各项的支持度进行升序排列并对 $T$ 中每项重复6：
6. 在该项的 Transactional Array 的第一个空白单元中填入一个三元组$(A,B,C)$：

①$A$ 标记 $T$ 中排在该项后一项所在的行号，如果该项为 $T$ 的最后一个，则记为#；

②$B$ 标记 $A$ 所指行的第一个空白单元所在的列号，如 $A$ 为#，则 $B$ 为#；

③$C$ 标记该项在 $T$ 中出现次数；

步骤二：含重复项的 COFI-tree' 的生成

输入：$M'$

输出：各1-频繁项的 COFI-tree'

算法流程：

1. 找到满足最小支持度的项在 $M'$ 中的位置，对该项及之后的各项 $A$，重复2；
2. 将 $A$ 的 Transactional Array 中所有元组按最后一个元组项(项重复数 $N$)分为若干集合，并按 $N$ 的升序排列，对于每个集合重复3和4；
3. 为每个 $A(N)$ 创建一个根节点，将累计支持度($s$)和参与度($p$)都置为0；并对集合中的每一个三元组 $B$ 重复4到6步；
4. 找到以 $B$ 为头，以(#，#，$x$)($x$ 为任意数)为尾的链表产生的模式 $C$；
5. 创建表示 $C$ 的树分支：如果某项在原树中已存在相同的节点，则更新该节点的累计支持度 $s$；否则创建一新节点代表该项，设置 $s$ 为1，$p$ 为0；
6. 设 $C$ 的树形表示为 $H_1$，则对于本树中已存在的路径和已创建的同一项的其它 COFI-tree' 中所有路径 $P$，对于任一 $p'\in P$，若 $p'\not\subset H_1$ 且 $H_1\not\subset p'$ 重复7和8；
7. 如果 $H_1\subset_\delta p'$ 则对于所有 $h\in H_1$ $h.s=h.s+\delta(H_1,p')$；(此处的 $H_1\subset_\delta P$ 表示对于任意 $e_h\in H_1$，存在 $e_p\in P$，满足$(\delta(e_h)\leqslant\delta(e_p))\wedge S(e_h,e_p)$
8. 如果 $p'\subset_\delta H_1$，则对于所有 $q\in p'$，$q.s=q.s+\delta(p',H_1)$；

步骤三：COFI-tree' 挖掘算法

输入：某1-频繁项 A 的 COFI-tree'

输出：含该节点的所有频繁模式

算法流程：

1. 对于以 $A$ 为根的所有 COFI-tree'，设 $T$ 为其中一棵，重复2到6；
2. 对于 $T$ 的每一个叶子节点 $J$，重复3到6；
3. 以 $J$ 到根作为一条路径构建一个模式 $P$，模式的支持度为 $J.s$-$J.p$，并将路径上所有节点的 $p$ 加上 $J.s$-$J.p$；
4. 将 $P$ 和 $P$ 的各子模式，加入候选集 $C$，若 $C$ 中已存在某模式，则更新其支持度，加上 $J.s$-$J.p$；
5. 从 $J$ 往上依次检查到根节点，如果有节点 $J'$ 的支持度不等于参与度，则以 $J'$ 到根作为一条路径构建一个模式 $P'$，模式的支持度为 $J'.s$-$J'.p$，并将路径上所有节点的 $p$ 加上 $J'.s$-$J'.p$；
6. 将 $P'$ 和 $P'$ 的各子模式，加入候选集 $C$，若 $C$ 中已存在某模式，则更新其支持度，加上 $J'.s$-$J'.p$；
7. 在 $C$ 中由最小支持度得到频繁模式集。

### 3.2 算法示例

假设有事务数据库 $D$ 如表1所示。

表1 含重复项的事务数据库 D'

| TID | 包含项 | | |
|---|---|---|---|
| 1 | $B(3)$ | $A(2)$ | $E(1)$ |
| 2 | $B(1)$ | $E(1)$ | |
| 3 | $B(3)$ | $E(1)$ | |
| 4 | $B(9)$ | $E(5)$ | |
| 5 | $B(6)$ | $A(4)$ | |
| 6 | $B(9)$ | $F(3)$ | |
| 7 | $B(6)$ | $A(4)$ | |

表1中项名后的数字为该项的出现次数。我们在表2中构造新的反转矩阵 $M'$，因为项的数量这一属性的引入，我们将原来的二元组改成三元组，新增的末元则指明了该项的出现次数，而 $M'$ 中各项按总出现次数(定义1中支持度)进行升序排列。

表2 含重复项的反转矩阵 M'

| Loc | Index | Transaction Array | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ($F$,3) | (4,1,3) | | | | | | |
| 2 | ($E$,8) | (3,1,1) | (4,3,1) | (4,4,1) | (4,5,5) | | | |
| 3 | ($A$,10) | (4,2,2) | (4,6,4) | (4,7,4) | | | | |
| 4 | ($B$,37) | (#,#,9) | (#,#,3) | (#,#,1) | (#,#,3) | (#,#,9) | (#,#,6) | (#,#,6) |

接下来以表2为例，设最小支持度为5，则 COFI-tree' 的构造从 $E$ 开始，因为 $E$ 在事务中按出现次数划分有两种形式，单个出现和五个一起出现，为此我们分别构造两棵 COFI-tree' 在图1中。COFI-tree' 的元素数据结构较原来有两点变化，首先是项后多了数字表明该项的重复数；其次，原来的支持度内涵有所变化，因为需要提取的频繁模式较以往不同，形如 $AB(2)$ 和 $A(2)B$ 相似但不相同(见定义3)的模式会被分别对待，而以往都化成 $AB$。所以 COFI-tree' 中引入“累计支持度”的概念，意指该项为所有相似或不相似的模式提供的支持度之和(以下记为 $S$)。下面说明图1中 COFI-tree' 的生成过程：

由包含 $E$ 的第一个事务 $E(1)A(2)B(3)$ 可以得到树1，以 $E(1)$ 为根节点，若后续项不在当前项的子节点中，则新构造一子节点表示该项，$S$ 置为0；然后将包含 $E$ 的第2个事务 $E$

(1)$B(1)$加入，首先节点$E(1)$的$S$更新为2，并新加入子节点$B(1)$，但我们发现，树的已存分支$E(1)A(2)B(3)$中包含了$E(1)B(1)$，即模式$E(1)B(1)$的$S$被前一事务更新为2，为此我们再次更新了$E(1)$和$B(1)$的$S$得到树2；在树3的生成过程中，我们没有考虑$E(1)A(2)B(3)$对$E(1)B(3)$的支持累加，因为稍后的步骤中会将类似的情况考虑进去；最后我们为$E(5)$生成新的COFI-tree'，当$B(9)$被加入树中时，$E(1)$树中的$E(1)B(1)$的$S$增加了5，而$E(1)B(3)$的$S$增加了3，因此这两条路径上节点的$S$被更新，得到最终结果。项目$A$的COFI-tree'构造方法类似。

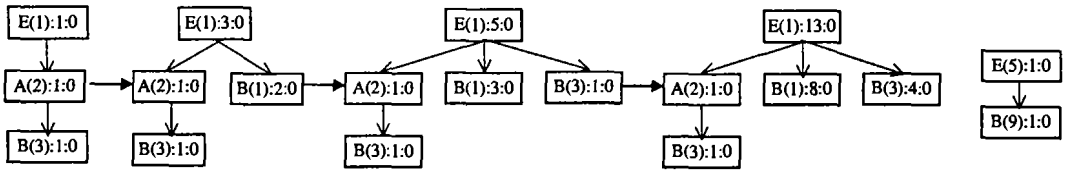


图1　由$M'$构造$E$的COFI-tree'

最后，由各节点的COFI-tree'生成频繁模式较简单，我们用图2展示了图1中以E(1)为根的树的频繁模式生成过程：

首先从树的最底层叶子节点($B(3):1:0$)开始，该节点的参与度(以下记为$P$)和$S$相差1，将以该节点为末节点，($E(1):13:0$)为首节点的整条路径上的所有节点的$P$加上末节点的($P$-$S$)1，并将该路径上的所有节点串接得到一模式($E(1)A(2)B(3)$)，$S$为1，我们在树2的说明框中描述了该分支生成的候选集$E(1)A(2)B(3)$和它的子集及相应支持度；然后找到下一个未处理的叶节点$B(1):8:0$，重复以上过程，新增一个候选模式$B(1)E(1)$，$S$为8，相应路径上所有节点的$P$也同时加上8；最后，因为由节点$B(3):4:0$得到的候选$B(3)E(1)$在第一步中已经生成，我们需要更新$B(3)E(1)$的$S$为5。此外，由$E(5)$为根的树可生成唯一的候选模式$E(5)B(9):1$。

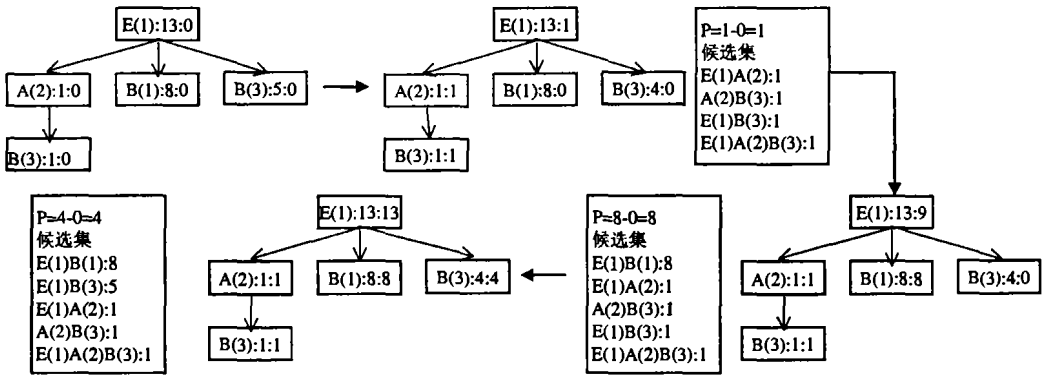


图2　由COFI-tree'生成频繁模式的过程

## 4　性能分析

算法和Apriori算法相比具有如下优点：(1)采用了COFI-tree'保存候选集，该树充分利用的模式间的包含关系，有效地压缩了候选集的占用空间；(2)整个算法仅需扫描两遍数据库，避免了多次扫描数据库带来的I/O瓶颈问题。

该算法和较新的FP-tree算法相比同样具有优点：当数据库的项很多时，FP-tree生成的频繁模式树将大到无法一次放入内存，如果置于二级存储器中，FP-tree算法的I/O开销将难以接受；而本算法在由COFI-tree'生成频繁模式时每次只产生一个频繁项的COFI-tree'，当1-频繁项较多时，该树所占的平均空间低于整个FP-tree所占空间的1/10。因此，若挖掘的初始支持度阈值较小时，该算法因为减小了I/O代价，时间开销比FP-tree算法有显著的下降。

**总结**　多层关联规则的挖掘近年来日益受到关注，随之出现的一个问题是模式在根据概念树泛化后会出现重复项，确保这些重复项所蕴含的信息被有效地发掘出来成为一个新的研究课题。本文采用了当前最新的高效关联规则发掘算法(IM)作为基础，提出了新算法以解决上述问题。该算法通过减少中间结果的大小有效地降低了挖掘的时间开销。

## 参考文献


1　Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items of large database. In: Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data, Washington, D. C, May 1993. 207～216

2　Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: ACM-SIGMOD Dallas, 2000

3　Mohammad E, Zaiane R. Inverted matrix: efficient discovery of frequent items in large datasets in the context of interactive mining. In: SIGKDD'03, Washington, D. C, Aug. 2003

4　Kok-Lenng, Wee-Keong, Ee-Peng. Mining multi-level rules with recurrent items using FP'-tree. In: Proc. of the 3rd Int. Conf. on Information, Communications and Signal Processing, Singapore, Oct. 2001