

一种高效的关联规则增量式更新算法

涂 明¹, 张公让¹, 程业媛²

(1 合肥工业大学 管理学院, 安徽 合肥 230009; 2 中国移动通信集团 广西有限公司北海分公司, 广西 北海 536000)

摘 要: 提出一种基于垂直型数据格式的算法 TMFUP, 用来解决数据挖掘中数据库增加而最小支持度不变化时关联规则增量式更新问题. 该算法只须扫描原始数据库和新增数据库一遍.

关键词: 关联规则; 增量式更新; 垂直型数据格式

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2010)09-0056-05

An Efficient Incremental Updating Algorithm for Mining Association Rules

TU Ming¹, ZHANG Gong-rang¹, CHENG Ye-yuan²

(1 School of Management, Hefei University of Technology, Hefei 230009, China;

2 Guangxi Limited Company Beihai Company, China Mobile Communication Group, Beihai 536000, China)

Abstract: This paper presents a algorithm TMFUP based on the vertical data format. The paper keeps the minsupport changeless when new transaction database db is added to original database DB and generates the new association rules. This algorithm only takes one-scan on both the original database and the increased database.

Key words: association rule; incremental updating; vertical data format

1 引言

数据挖掘(Data Mining, DM)是指从大量的数据中挖掘出隐含的、未知的、用户可能感兴趣的和对决策有潜在价值的知识和规则. 关联规则作为数据挖掘方法中的一种, 在 1993 年由 Agrawal 等人首先提出. 所谓关联规则挖掘就是从大量的数据中挖掘出有价值的描述数据项之间相互联系的有关知识.

关联规则挖掘的问题可以分解成以下两个问题: (1) 找出事务数据库中所有支持度不小于最小支持度的项目集, 称为频繁项集. (2) 在频繁项集中产生所有置信度不小于最小置信度的关联规则. 由于第二个问题的求解是比较容易和直接的, 因此目前的关联规则算法都是针对第一个问题提出的^[1].

2 问题描述

由于挖掘出来的关联规则只是相对于当前的数据库有效, 而现实中数据库是经常发生变化, 因此迫切需要高效的算法来更新已挖掘出来的关联规则. 因此关联规则的更新问题又可以分为两类: (1) 给定交易数据库 DB, 假定最小支持度不变, 当一个新的交易数据集 db 添加到 DB 中去时, 如何生成 DB ∪ db 中的关联规则; (2) 当交易数据库 DB 保持不变, 而最小支持度和最小信任度发生变化时, 关联规则的更新问题. 本文研究的重点集中在第一类问题, 提出了一种高效的更新算法 TMFUP.

事实上, 当一个新的事物数据库 db 添加到 DB 中去时, 对任意一个项集可能有以下三种情况:

(1) 在 DB 中是频繁项集, 在 db 中未知;

- (2) 在 db 中是频繁项集,在 DB 中未知;
- (3) 在 DB 和 db 中均为非频繁项集.

FUP 算法能有效地处理情况(1)、(3),但对情况(2)显得无能为力.本算法对三种情况都能比较高效地进行处理,对于(1)、(2)两种情况,只须扫描 db 和 DB 的 k 阶项集的支持交易集合就能得到任意一个项集的支持数,再结合最小支持数就能得到频繁项集.对情况(3),由下面的推论 1 可知这部分项集不可能在 $DB \cup db$ 中频繁,故可以在 k 阶候选项集及 k 阶项集的支持交易集合中将其删除.

3 算法介绍

设 $I = \{i_1, i_2, \dots, i_m\}$ 是全体数据项的集合.给定一个事务数据库 DB,其中的每一个事务 T 是 I 中一组项目的集合,即 $T \subseteq I$, T 有一个唯一的标识符 TID.

定义 1 若项集 $X \subseteq I$ 且 $X \subseteq T$,则称 T 包含 X 或 T 支持 X .事务数据库 DB 中包含项集 X 的总条数,称为 X 的支持数,用 $X.count$ 表示.

定义 2 设 D 为事务数据库 DB 中的事务总条数,对于项集 X ,若 $X.count \geq D \times s$,其中 s 是最小支持度,则称 X 是频繁项集,否则称 X 是非频繁项集^[2].

本算法是一种基于垂直型数据格式的算法,下面简单介绍两种基本数据格式:

3.1 水平型数据格式

水平型数据格式是一种常见的数据格式.在这里,数据库由一系列事务构成.每个事务有一个唯一的事务标识符 TID 以及相应事务所包含的项目表列(即项目集).使用这种格式,算法的计算量主要集中在支持数的计算.如表 1 所示.

表 1 水平型数据格式

TID	项集			
1	A	B	C	D
2	A	B	C	
3	B	C	D	
4	A	C	D	

3.2 垂直型数据格式

垂直型数据格式是把交易数据按照项集的顺序存储,数据集由一系列项目构成,每一个项目均带有其相应的 TID 表列,即包含该项目的所有事务的 TID 表,且按字典序的升序排列^[3-4].如表 2 所示.

表 2 垂直型数据格式

项集	TID			支持数
A	1	2	4	3
B	1	2	3	3
C	1	2	3	4
D	1	3	4	3

定义 3 设 X 是项集, $S(X) = \{t.TID \mid X \in t\}$ 是包含 X 的所有交易的 TID 组成的集合,称为 X 的支持交易集合, $S(X)$ 中元素的个数 $|S(X)|$ 就是 X 的支持数^[5].

以下为算法中用到的定理:

定理 1 若 k 阶项集 X 在 $DB \cup db$ 中是频繁项集,则 X 至少在 DB 或 db 中的一个频繁,即 $X.count_{DB} \geq D \times s$ 或者 $X.count_{db} \geq d \times s$.

推论 1 $DB \cup db$ 中的频繁项集必定是从 DB 或者 db 的频繁项集中产生.

定理 2 任何一个 k 阶频繁项集的非空子集一定是频繁项集,若一个 $k-1$ 阶项集是非频繁项集,则该项集的超集也一定是非频繁项集.

定理 3 $\forall C \in C_k, S(C)$ 可以由 S_{k-1} 中的两个元素 $S(X), S(Y) (X \neq Y)$ 得到,且 $S(C) = S(X) \cap S(Y)$ ^[5].

证明:设 X, Y 是 C 的任意两个 $k-1$ 阶子集,由 $X \neq Y$ 可知 $X \cup Y = C$.因此对 $\forall t \in S(C)$ 当且仅当 $t \in S(X), t \in S(Y)$.所以, $S(C) = S(X) \cap S(Y)$.又因为 $C \in C_k$,由 $C_k = \text{Apriori_gen}(L_{k-1})$ ^[6] 的定义可知, $X, Y \in L_{k-1}$,也就是说 $S(X) \in S_{k-1}, S(Y) \in S_{k-1}$.已知项集 $\{A\}, \{B\}$ 及 $\{C\}$ 的支持交易集合分别是 $\{1,2,4\}, \{1,2,3\}$ 及 $\{1,2,3,4\}$,则项集 $\{A C\}$ 的支持交易集合为 $\{1,2,4\}$,项集 $\{B C\}$ 的支持交易集合为 $\{1,2,3\}$,那么项集 $\{A B C\}$ 的支持交易集合为 $\{1,2,4\} \cap \{1,2,3\} = \{1,2\}$.

定理 4 对于所有的候选项集 $C \in C_k$,如果能由 $L_{k-1} \bowtie L_{k-1}$ 生成,并且 $\exists t \in C$,有 $t \in L_{k-1}$,那么也可以由 $L_{k-1} \bowtie L_1$ 生成^[7].

证明:如果 $\exists C \in C_k$,有 $C \in L_{k-1} \bowtie L_{k-1}$,并且 $C \in L_{k-1} \bowtie L_1$.因为对 $\exists t \in C$,有 $t \in L_{k-1}$,又因为 $C \in L_{k-1} \bowtie L_1$,那么有 $C - t \in L_1$.这明显与 $C - t$ 的长度为 1 且 $C \in C_k$ 又是由频繁项集所生成相矛盾.所以,对 $\forall C \in C_k$ 有 $C \in L_{k-1} \bowtie L_1$.

符号定义:

$L_k^{DB \cup db}$: DB $\cup$ db 的 k 阶频繁项集;

L_k^{DB} : DB 的 k 阶频繁项集; L_k^{db} : db 的 k 阶频繁项集;

C_k^{DB} : DB 的 k 阶候选项集; C_k^{db} : db 的 k 阶候选项集;

S_k^{DB} : DB 的 k 阶项集的支持交易集合; D : DB 数据库中事务总数;

S_k^{db} : db 的 k 阶项集的支持交易集合; d : db 数据库中事务总数;

$X.\text{count}_{DB \rightarrow DB}$: DB 的频繁项集在 DB 中的支持数; s : 项集支持度;

$X.\text{count}_{DB \rightarrow db}$: DB 的频繁项集在 db 中的支持数;

$X.\text{count}_{db \rightarrow db}$: db 的频繁项集在 db 中的支持数;

$X.\text{count}_{db \rightarrow DB}$: db 的频繁项集在 DB 中的支持数.

3.3 算法步骤:

(1) 第一次扫描原数据库 DB 和新增数据库 db 时就记录下每个项所对应的 TID 值, 利用垂直型数据格式来建立 DB 和 db 的一阶项集的支持交易集合.

(2) 在步骤(1)中扫描并记录的同时, 利用定理 3 求出 DB 的一阶频繁项集, 由于支持交易集合 $S(X)$ 中元素的个数 $|S(X)|$ 就是 X 的支持数, 因此我们可以很简单地得到 DB 一阶项集中各项的支持数 $X.\text{count}_{DB \rightarrow DB}$, 同理求出 db 的一阶频繁项集及其各项的支持数 $X.\text{count}_{db \rightarrow db}$.

(3) 扫描 DB 的一阶项集的支持交易集合, 利用定理 3 求出 db 的一阶频繁项集在 DB 中的支持数 $X.\text{count}_{db \rightarrow DB}$, 计算 $X.\text{count}_{db \rightarrow DB} + X.\text{count}_{db \rightarrow db}$ 得到 db 的一阶频繁项集在 $DB \cup db$ 中的支持数 $X.\text{count}_{DB \cup db}$, 其中 $X.\text{count}_{DB \cup db} \geq (D + d) \times s$ 的项集即为在 db 和 $DB \cup db$ 中均频繁的项集.

(4) 将在 L_1^{db} 和 L_1^{DB} 中均出现的项集从 L_1^{DB} 中删除, 这样将避免重复运算. 扫描 db 的一阶项集的支持交易集合, 求出 DB 的一阶频繁项集在 db 中的支持数, 把 L_1^{DB} 在 db 中的支持数加上其在 DB 中的支持数得到 L_1^{DB} 在 $DB \cup db$ 中的支持数, 其中支持数大于等于 $(D + d) \times s$ 的项集即为 L_1^{DB} 在 $DB \cup db$ 仍然频繁的项集.

(5) 将步骤(3)和(4)中得到的项集合并即为 $DB \cup db$ 的一阶频繁项集. 重复步骤(1)~(5)得到 $DB \cup db$ 所有的频繁项集. 将步骤(1)中得到的 DB

和 db 的一阶项集的支持交易集合合并保存, 可以在下一次进行数据库更新后的关联规则挖掘中利用这些保存的结果, 这样在下一次数据挖掘时就不需要在扫描原始数据库了, 因此以后的数据挖掘中就仅仅只需扫描新增数据库 db 一次, 而无须扫描原始数据库.

3.4 算法实现步骤:

输入: 事务数据库 DB, 新增数据库 db, 最小支持度 s

输出: $DB \cup db$ 中的频繁项集 $L_k^{DB \cup db}$

(1) Add_TID(DB); //调用函数

(2) Add_TID(db); //调用函数

(3) for($k = 1$; $L_k^{DB \cup db} = \emptyset$; $k++$)

(4) $C_1^{DB} = \{DB \text{ 全体项}\}; L_1^{DB} = \emptyset; X_1^{DB} = \emptyset;$

(5) sort($L_{k-1}^{DB \cup db}$) //排序

(6) $C_k^{DB} = \text{Apriori_gen}(L_{k-1}^{DB})$; //利用定理 4, 由 DB 项集的 $k-1$ 阶支持交易集合的交集求出 DB 的 k 阶候选项集

(7) for all $X \in C_k^{DB}$ do begin

(8) $X.\text{count}_{DB \rightarrow DB} = |S(X)|$; //支持交易集合 $S(X)$ 中元素的个数即为 X 的支持数

(9) if $X.\text{count}_{DB \rightarrow DB} \geq D \times s$ then

(10) $L_k^{DB} = \{X \mid X \in L_k^{DB}\}; S_k^{DB} = \{S(X) \mid X \in L_k^{DB}\};$

//求出 DB 的 k 阶频繁项集及 DB 的 k 阶项集的支持交易集合

(11) end if

(12) end for

(13) $C_k^{db} = \text{Apriori_gen}(L_{k-1}^{db})$; //和步骤(6)同理

(14) $C_1^{db} = \{db \text{ 全体项}\}; L_1^{db} = \emptyset; X_1^{db} = \emptyset;$

(15) for all $X \in C_k^{db}$ do begin

(16) $X.\text{count}_{db \rightarrow db} = |S(X)|$; //支持交易集合 $S(X)$ 中元素个数即为 X 的支持数

(17) if $X.\text{count}_{db \rightarrow db} \geq d \times s$ then

(18) $L_k^{db} = \{X \mid X \in L_k^{db}\}$; //求出 db 的 k 阶频繁项集

(19) if $X \in S_k^{DB}$ then begin

(20) $X.\text{count}_{db \rightarrow DB} = |S_k^{DB}(X)|$;

//扫描 DB 的 k 阶项集的支持交易集合, 求出 db 的 k 阶频繁项集在 DB 中支持数

(21) $X.\text{count}_{db \cup DB} = X.\text{count}_{db \rightarrow DB} + X.\text{count}_{db \rightarrow db}$;

(22) if $X.\text{count}_{db \cup DB} \geq (D + d) \times s$ then begin

(23) $X_k^{db} = X_k^{db} \cup \{X\}$; //求出 db 的频繁项集在 $DB \cup db$ 中仍然频繁的项集

(24) end if

(25) end if

(26) end if

(27) end for

(28) for all $X \in L_k^{DB}$

(29) if $X \in L_k^{db}$ then delete X from L_k^{DB}

```

(30) end if //将  $L_k^{DB}$  和  $L_k^{db}$  中相同的项从  $L_k^{DB}$  中删除
(31) if  $X \in S_k^{db}$  then begin
(32)  $X.count_{DB \rightarrow db} = S_k^{db}(X)$ ;
//扫描 db 的  $k$  阶项集的支持交易集合,得到 DB 的  $k$ 
阶频繁项集在 db 中支持数
(33)  $X.count_{DB \cup db} = X.count_{DB \rightarrow db} + X.count_{DB \rightarrow DB}$ ;
(34) if  $X.count_{DB \cup db} \geq (D+d) \times s$  then
(35)  $L_k^{DB} = X_k^{DB} \cup \{X\}$ ; //计算求出在 DB 和  $DB \cup db$ 
中均频繁的项集
(36) end if
(37)  $L_k^{DB \cup db} = L_k^{DB} \cup L_k^{db}$ 
//将 DB 和 db 的频繁项集合并得到更新后数据库  $DB \cup$ 
db 的  $k$  阶频繁项集
(38) end for
(39) end for
建立 database 的一阶项集的支持交易集合 Add_TID
(database) 的过程:
 $C_1 = \{\text{database 的全体项}\}$ 
for all item  $C \in C_1$  do  $S(C) = \emptyset$ ;
end for
for all  $t \in \text{database}$  do:
 $C_t = \text{subset}(t, C_1)$ ; //搜索  $C_1$  的各个项集  $c$ , 寻找包含
项集  $c$  的交易  $t$ 
for all  $C \in C_t$  do  $S(C) = S(C) \cup t.TID$  //建立项集的
支持交易集合
end for
end for
产生候选项集的 Apriori_gen( $L_{k-1}$ ) 的过程[6]:
for all  $l_{k-1} \in L_{k-1}$  do
for all  $l_1 \in L_1$  do
 $l_k = l_{k-1} \circ l_1$  //利用定理 4 来生成  $k$  阶项集
 $S(l_k) = S(l_{k-1}) \cap S(l_1)$  //利用定理 3, 得到  $k$  阶项集
的 TID 表
end for
end for

```

4 本算法与 FUP 算法比较

(1) ①FUP 候选项目集 C_k 的生成是通过链接 L_{k-1} 和 L_{k-1} 得到. ②本算法中 TID 表列以升序存储, 所以候选项目集 C_k 的生成与支持可以通过简单地链接 L_1 和 L_{k-1} 得到.

(2) ①FUP 算法采用复杂的 Hash 树数据结构来存储各个候选项集. 因而必须对 Hash 数据结构进行维护与搜索, 从而又增加了额外的计算负载. 而且数据的水平型格式存储使得每次循环均需扫描整个数据库或每个局部划分. ②本算法不需要复杂的 Hash 树数据结构, 不需要对 Hash 树数据结构进行

维护与搜索, 计算项集的支持数时, 无需对整个数据库进行遍历, 只需要扫描 k 阶项集的支持交易集合即可. 算法 TMFUP 在第一次进行关联规则挖掘时, 只需对原始数据库 DB 和新增数据库 db 各扫描一次, 在第二次进行关联规则挖掘时, 仅仅只需要对 db 扫描一次, 而不用对 DB 扫描.

现代企业中数据库的更新非常频繁, 而且更新的数据量有时也很大, 本算法兼顾了两种, 即新增数据库 db 远小于 DB 或者 db 和 DB 大小相差不大, 本算法都能高效率地解决关联规则的更新问题.

5 实验结果

为了进一步验证算法 TMFUP 的性能, 用 VC++ 6.0 在使用 Windows XP 操作系统、CPU 为 P4 2.8GHz、内存为 384MB 的计算机上对 TMFUP 和 FUP 这两种算法进行测试, 使用了与文献[8]的数据生成程序来合成所需要的测试数据, 实验结果如图 1 所示, 图中执行时间为当 DB 中的所有频繁项目集都已经得到时求 $DB \cup db$ 中的所有频繁项目集所需时间 (其中实验中使用的新增数据库 db 条数为 3 000 条).

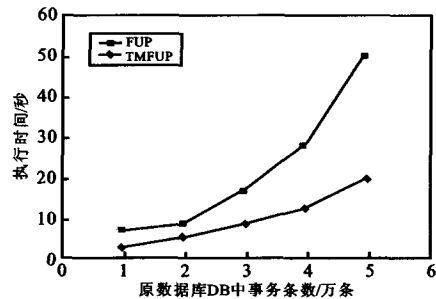


图 1 测试结果

6 结束语

对“在给定的最小支持度, 当新的数据库 db 添加到原始数据库 DB 中时, 如何生成 $db \cup DB$ 中的大项集”问题进行了研究, 提出了一个高效的增量式更新算法 TMFUP. 与算法 FUP 相比, 充分利用以往挖掘过程中的结果, 无需再次扫描原数据集, 对新增数据集也只扫描一次, 减少了候选集的数量, 在算法的运行时耗上具有很明显的优势. 算法的测试结果可知算法是可行且有效的. 本算法利用的垂直型数据格式需要更多的存储空间来存储前面挖掘的相关信息, 但在各种存储备份方案优化的今天, 用空间来换取时间还是划算的. 本算法的下一步改进将集中在当数据库中的数据删除一部分后的关联规则的挖

掘问题.

参考文献:

- [1] Agrawal R, Imielinski T, Swami A. Database mining: a performance perspective[J]. IEEE Transactions on Knowledge and Data Engineering, 1993, 5(6): 914 - 925.
- [2] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases[C]//Proc. ACM SIGMOD International Conference. Management of Data. Washington, DC, 1993: 207 - 216.
- [3] Mohammed J Zaki, Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure[J]. IEEE Transactions On Knowledge And Data Engineering, 2005, 17(4): 462 - 478.
- [4] Jiawei Han, Micheline Kanber. 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, 2008: 146 - 177.
- [5] 陈安, 陈宁, 周龙骧, 等. 数据挖掘技术及应用[M]. 北京: 科学出版社, 2006: 50 - 104.
- [6] Agrawal R, Srikant R. Fast algorithm for mining association rules [C]//The 20th International Conference on VLDB. Santiago, Chile, 1994: 487 - 499.
- [7] 刘华婷, 郭仁祥, 姜浩. 关联规则挖掘 Apriori 算法的研究与改进[J]. 计算机应用与软件, 2009, 26(1): 146 - 149.
- [8] IBM. Computer science: intelligent information systems [EB/OL]. [2005 - 12 - 15]. [http://www. ibm.com/](http://www.ibm.com/)

作者简介:

涂明男, (1984 -), 硕士研究生. 研究方向为数据挖掘、数据库.

(上接第 55 页)

- [2] Leeson D. A simple model of feedback oscillator noises spectrum[J]. Letters IEEE, 1966, 54(2): 329 - 330.
- [3] Hoshino K, Hegazi E, Rael J, et al. A 1.5V, 1.7mA 700Mz CMOS LC oscillator with no up-converted flicker noise[C]// IEEE ESSCC. San Francisco, 2001: 337 - 340.
- [4] Qiuting H. Phase noise to carrier ratio in Lc oscillators[J]. IEEE Trans on Circuits and Systems-I: Fundamental Theory and Application, 2000, 47(7): 965 - 979.
- [5] Samori C, Lacaita A, Zanchi A, et al. Phase noise degradation at high oscillation amplitudes in LC-tuned VCO's [J]. IEEE Journal of Solid-State Circuits, 2000, 35(1): 96 - 99.
- [6] Demir A, Mehrotra A, Roychowdhury J. Phase noise in oscillators: a unifying theory and numerical methods for characterization [J]. IEEE Trans Circuits and System - I: 2000, 47(5): 655 - 674.
- [7] Ali Hajimiri, Thomas Lee. A general theory of phase noise in electrical oscillators[J]. IEEE Journal of Solid - State Circuits, 1998, 33(2): 179 - 194.
- [8] Andreani P, Wang X. On the phase noise and phase error performances of multiphase LC CMOS VCOs[J]. IEEE Journal of Solid - State Circuits, 2004, 39(11): 1883 - 1893.

作者简介:

吴秀龙男, (1979 -), 博士, 副教授. 研究方向为数模混合集成电路设计.