一种基于SFP树的快速关联规则挖掘算法

李龙澍,王 永,魏博诚
(安徽大学 计算机智能与信号处理教育部重点实验室,安徽 合肥 230039)

摘 要:对于传统的FP-Growth算法而言,当事务数据库D很大时,构造基于内存的FP树可能是不现实的。针对此问题,提出了一种基于样本事务数据库的SFP算法。该方法对事务数据库D进行随机抽样,得到样本数据库S,此时以比指定的支持度min_sup小的支持度(min_sup′)在S中挖掘频繁项集L′,根据求得的频繁项集L′,在剩余的数据库D-S中求得L′中各事务的支持数,这在大多数情况下就可以求得所有的频繁项集,但是有时可能会漏掉一些。这时可以对D进行二次扫描以发现漏掉的频繁项集。该算法大多数情况下只需要对数据库进行一次扫描,最坏情况下也只需要对数据库进行二次扫描。当把效率放在首位时,比如计算密集事务数据库的频繁项集时,SFP算法尤其合适。

关键词:关联规则;频繁项集;FP树;样本事务数据库
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2011)05-0079-04

A Fast Association Rule Mining Algorithm Based on SFP Tree

LI Long-shu, WANG Yong, WEI Bo-cheng
(Ministry of Education Key Laboratory of Intelligent Computing & Signal Processing, Anhui University, Hefei 230039, China)

Abstract: For the traditional algorithm of FP growth, when database is very big, the structure of FP tree based on memory, sometimes is not realistic. According to this problem, put forward a SFP algorithm based on the sample of the transaction database. This method gets sample transaction database S from database D through random sampling, firstly in less than a specified min_sup support, digging frequent itemsets L′ from S, then calculating L′each element of the support in the rest of the data sets D-S, finally getting L by min_sup. In most cases it can be obtained all the frequent itemsets, but sometimes no. In this time omit the frequent itemsets a second scan. The algorithm needs only one scan of database in most cases, the worst case scenario also needs only second scan to database. When the efficiency is the most important, such as intensive application must be frequently calculated, SFP algorithm is the first selecting.

Key words: association rule; frequent itemsets; FP tree; sample transaction database

0 引 言

在传统的数据挖掘的研究中,关联规则挖掘方面的研究一向开展的最广泛,关联规则最初被应用于交易数据库[1]。通过对关联规则的研究,可以发现在一个超级市场中用户购买的各商品之间比较隐蔽的关系,像比较著名的"啤酒-尿布"示例:啤酒和尿布本来被认为是两个毫无关联的物品,在经过对数据的挖掘研究之后,发现它们两个有着极为密切的关系,买尿布的用户当中有很多一部分人群同时也买了啤酒。像上述这种示例还有很多,不再一一列举。商场POS机的设置以及条形码技术的发展使得超级市场存储了大量数据记录,这些记录详细记录了每个客户交易的数量、时间、商品和价格等各种信息,从而为数据挖掘的研究提供了数据基础。由此不难看出,通过对事务关联规则的研究,可以有效地为商场的决策提供依据。关联规则最初是由Rakesh Agrawal, Tomase, Imielinski和Arun Swami提出的。在众多算法中,FP-Growth[2]算法最为著名,与Apriori算法及其它各种算法相比,它可以有效地处理长和短的频繁模式,对于大规模化的事务数据库也极其有效。它比树投影算法要快,甚至比Apriori算法要快出大约一个数量级。

FP增长算法用于挖掘事务数据库中布尔型频繁模式的关联规则[3]。FP算法的挖掘过程一般比较复杂,不过可以被简单地分为三个步骤[4]:第一步,扫描事务数据库D,根据所给的支持度min_sup求出频繁项列表L;第二步,依据L表,构建FP树,递归地对该FP树进行频繁项集挖掘;第三步,根据频繁模式以及

收稿日期:2010-09-12;**修回日期**:2010-12-20
基金项目:安徽省自然科学基金项目(090412054)
作者简介:李龙澍(1956-),男,安徽亳州人,教授,博士生导师,研究方向为知识工程、软件分析与测试。

行业背景建立相关关联规则。不过,FP 算法是一种基于内存的算法,尤其是事务数据库比较大时,将会占用大量的内存。下面两种情况都需要占用大量的内存,影响算法的效率[5]:①对 FP 树的挖掘过程中,需要构造它的条件 FP 树,并递归地对它的各个条件 FP 树进行关联规则挖掘,这无疑是一项很耗费内存的工作;②当事务数据库很大时,比如为稠密事务数据库时,将会占用大量的内存空间以及 CPU 的处理时间。甚至,随着事务数据库的宽度和深度不断增加,将可能造成无法构建相应的 FP 树,也就无法求得 FP 树所映射的频繁一项集,最终导致算法无法正常的工作。

为了解决上述问题,不少学者根据 FP 算法的固有缺点提出了相关修改算法。文献[6]提出了最大频繁模式的快速挖掘与更新算法 DMFP。该算法使用前缀树来压缩和存放数据,通过调整前缀树中的节点链和节点信息并直接在前缀树上采用深度优先策略进行挖掘,从而避免了构建条件模式树,进而大大提高了算法的挖掘效率。文献[7]通过引入聚合链单链表结构,改进了原有 FP 树结构,此时创建的 FP 树是单向的,并且每个节点只保留指向它的父节点的指针,这样就可以节省大量的树空间。把项名相同而路径信息不同的节点压缩进聚合链单链表结构中,从而避免了生成节点链以及条件模式库。文献[8]提出了一种基于自顶向下的改进算法,运用集合运算思想以及递归的方法,通过保存前面扫描计算的结果进行最大频繁项集的查找,从而提高了频繁项集的挖掘效率。

1 传统的 FP-Growth 算法

input :一个事务数据库 D 和一个最小支持度阈值 min_sup。

output : 频繁项集 L。

第一步,扫描事务数据库 D 一次,根据所给的支持度 min_sup 求出频繁项列表 L。

第二步,依据频繁项列表 L,构建频繁模式 FP 树。一颗频繁模式 FP 树需要满足以下三种条件:①构建 FP 树的根节点(root),用"null"来标记它。扫描事务数据库,按频繁项列表 L 的次序依次处理每个事务中的项并为每个事务创建一个分枝;②项目前缀子树中的每一节点都包含 3 个部分:item_name,count,node_link,其中,item_name 代表每一个节点(项)的名称,count 代表到达该节点的路径的个数,node_link 的值代表指向下一个同名的节点的个数,当所指的同名的节点不存在时,node_link 的值为 null;③在频繁项目头表中,每一节点包含两个域:item_name、headof_nodelink,其中 headof_nodelink 代表指向 FP 树中指向相同 item_name 值的首节点的指针。当然,也可以在频繁项目头表中的每一节点增加一个域,用来记录此节点出现的频率。当满足频繁模式树 FP 的三个基本条件后,就可以来构建 FP 树了。当给定了事务数据库 DB 和给定的最小支持度 min_sup 后,FP-tree 树构建的基本步骤如下:①扫描事务数据库 D 一次,得到各项目的支持度,由最小支持度 min_sup 得到频繁项目列表 L,对频繁项目列表 L 按照支持度阈值由大到小的次序 R 排列,形成头表;②再次扫描事务数据库 DB,对每一条事务的所有频繁项按照次序 R 依次插入到 FP-tree 中。对每一频繁项生成的条件 FP-tree 进行挖掘,从而获得满足最小支持度阈值的频繁项目集。更详细内容请参见文献[9]。

2 基于样本的 SFP 算法

2.1 SFP 算法的基本思想

对给定事务数据库 D,在事务数据库 D[10] 中进行随机抽样得到样本数据库 S,然后在样本数据库 S 而不是 D 计算频繁项集。用这种方法,就会牺牲了一些精度,但换取了有效性。原本的事务数据库 D 可能非常大,不适合在内存中搜索频繁项集,而选取了合理的样本数据库 S 后,就会占用更小的内存却得到更高的效率。这样当扫描一次事务数据库后,由于是在样本数据库 S 而不是在事务数据库 D 中搜索频繁项集,就可能造成一些全局频繁项的丢失。针对此种情况,使用比最小支持度阈值 min_sup 低的 min_sup' 在样本数据库 S 中搜索频繁项列表 L'。然后,在数据库的其余部分 $D-S$ 中计算频繁项列表 L' 中每个项集的实际频率。使用一种判断机制来判定频繁项集是否全部在频繁项列表 L' 中,如果频繁项集全部在 L' 中,则只需要扫描一次事务数据库;否则,需要扫描两次事务数据库,以求得所有的频繁项集。当把效率放在首位时,比如计算密集事务数据库的频繁项集时,SFP 算法尤其合适。

2.2 SFP 算法的相关定义及性质

定义 1:项(item)是一个文字,在交易数据库中,它可以代表商品。在分类时,它可以代表属性的值。设 $I=\{i_1, i_2, \cdots, i_m\}$ 为项的全集,$D=\{T_1, T_2, \cdots, T_n\}$ 为事务数据库,其中每个事务 $T_i(i \in [1, \cdots, n])$ 包含事务 ID 号 TID 和一个 I 中项的子集 itemset[11]。

定义 2:事务数据库 D 是一个可以用来存储交易记录的数据库,当然,事务数据库 D 中的每一天记录都应该有唯一的标识。一条事务记录 TID 包括涉及到的项 item 的有序序列。表 1 是一个比较简单的事务数据库 D,其中 $T001$ 代表一条事务记录,$I1$,$I2$,$I5$ 是一条事务记录 $T001$ 中有序的项序列[11]。

定义3: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 $X<I$, X 在 D 中的支持数是指 D 中包含 X 的事务数,记为 $X.$ coun - tD。X 在 D 中的支持度是指 D 中包含 X 事务的百分比,记为 $X.$ supD 。如果 X 的支持度不小于用户给定的最小支持度阈值min_sup,则称 X 为 D 中的频繁项目集,项目集中项目的个数称为项目集的维数或长度,频繁12项目集简称频繁项目。

表1　事务数据库

TID	List of item ID's
T001	I1,I2,I5
T002	I2,I4
T003	I2,I3
T004	I1,I2,I4
T005	I1,I3
T006	I2,I3
T007	I1,I2
T008	I1,I2,I3,I5
T009	I1,I2,I3

定义4:I 的子集 X 称做项集或模式,项集 X 的支持度计数 sup _count (X) 为 D 中包含项集 X 的事务数; X 的支持度support (X) = sup _count (X) / | D |,其中,| D | 是 D 中事务的个数。显然,项集 X 的支持度是项集 X 在事务数据库中出现的频率。任意的项集并不提供有意义的知识。然而,当项集 X 的支持度大于一定值时,则呈现一定的统计规律。

定义5[12]:如果项集(模式) X 是频繁的,则对于预先指定的最小支持度阈值 min _sup, support(X) ≥ min _sup。通常,最小支持度阈值 min _sup 由用户或领域专家指定,而最小支持度计数 min _ count = [min _ sup ×| D |]。在事务数据库中挖掘频繁模式的问题可以描述为:给定事务数据库 D 和最小支持度阈值 min _sup,挖掘所有的频繁模式。

性质1:对于一个项集 IS,如果它在所有的样本数据库 S 上是不频繁的,则它不可能在所有的事务数据库 D 中是频繁的。

性质2:若项集 X 不是样本数据库 S 的强频繁项集,则 X 的超频繁项集一定不是事务数据库 D 的强频繁项集。

性质3:样本数据库 S 的强频繁项集一定是事务数据库 D 的局部频繁项目集,但事务数据库 D 的局部频繁项集未必是样本数据库 S 的强频繁项集。

2.3　SFP 算法的伪代码

算法:SFP 算法。使用 SFP 树,通过模式段增长挖掘频繁模式。

输入:事务数据库 D;

最小支持度计数阈值 min_sup;

输出:频繁项集 L。

(1) Input(D,min_sup);/ * 输入数据库 D 和最小支持度阈值 min_sup * /

(2) L←Φ ;

(3) Sampling_DB; / * 随机抽取样本数据库 * /

(4) for i = 1 to n do begin

(5) min_sup' = min_sup||S||/||D||; / * 计算样本事务数据库的最小支持度 * /

(6) Init_SFPTree/ * 利用样本数据库构造 FP 树 * /

(7) L' = Generate_LocalFreqent (SFP_Tree, min_sup') / * 利用 FP 算法生成样本的局部频繁项集 * /

(8) for item in L' begin

(9) select $\min\_sup_i$ from D-S / * 在剩余的数据集 D-S 中计算 L' 中每个项集的实际支持度 * /

(10) if($\min\_sup_i >= \min\_sup$); L←$item_i$

(11) if(L' 实际包含了 D 中的所有频繁项集) END

(12) else 扫描数据库 D,找出遗漏的频繁项集

(13) END

(14) Generate_globle(min_sup,L) / * 产生全局频繁项集 * /

(15) ReturnL;

其中,

第(3)步样本的抽取是把事务数据库 D 分成 n 等份,随机地抽取一个样本进行求局部频繁项集;

第(7)步求样本的局部频繁项集是根据传统的 FP - Growth 算法求得;当求得样本的局部频繁项集 L' 后,在剩余的数据库 $D-S$ 中求 L' 中每个项集的实际频率;

第(11)步采用一种判断机制来确定是否所有的频繁项集都包含在 L' 中,在此,采用的判断机制为:依然采用抽样的方法,从剩余的 $D-S$ 数据库中随机地抽取 n 个样本数据库 S 中没有出现的项集进行频繁判断,如果这些项集都不是频繁项集,就判定已找到了所有的频繁项集。否则,执行第(12)步;

在第12步中,仅仅对样本中从未出现的项集进行频繁测试。

第(6)步中 SFP 树的构造过程为:

(1) 扫描样本事务数据库 S 一次。收集频繁项集合 FI' 和它们的支持度计数。对 FI 按支持度计数降序排列,结果为频繁项集合 L';

(2) 创建 SFP 树的根节点,用"null" 来标记它。

对于 S 中的每个事务 T,执行:选择 T 中的频繁项,并按 L' 中的次序排序。设排序后的 T 中频繁项列表为

[p | P],其中p是第一个元素,而P是剩余元素的列表。调用inser_tree([p | P],T)。该过程执行情况如下:如果T有一个子女N是对N. item - name = p. item - name,则N的计数加1;否则创建一个新节点,将其计数设置为1,链接到它的父节点T,并且通过节点链结构将其链接到具有相同item - name的节点。如果P非空,递归地调用insert_tree(P,N)。更具体的内容请参照文献[1]。

3 实验结果及分析

为了测试SFP算法的实际运行效果,在同样的软硬件环境下,选取了6组不同的事务数据库,按事务数据库的大小分别用SFP算法和FP- Growth算法进行了挖掘测试。实验环境为Intel酷睿双核2.80CPU,2GRAM,操作系统为Windows XP,算法用C#编程语言实现。

实验结果如图1所示。

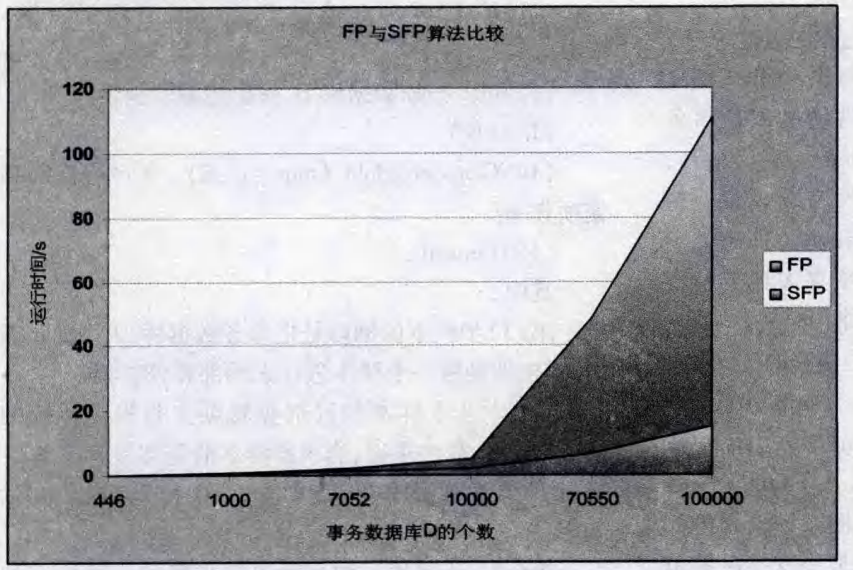

图1 实验结果

其中,min_sup = 8%,由结果可以看出,当事务的个数很大时,FP算法的效率下降的非常快,从趋势图可以看出,当数据库D进一步扩大时,FP算法的运行时间长度将变得不可想象,甚至是无法实现的。而SFP算法则能很好地解决这一课题,当数据库很大时,采用基于样本的SFP算法无疑可以大大提高算法效率。如果把效率放在首要位置,如计算密集的应用频繁运行时,抽样方法特别合适。

4 结束语

由于传统FP树增长算法是基于内存的,当事务数据库D非常大时,构建相应的FP - tree将占用大量的内存及CPU,降低了算法的运行效率。

基于此问题,文中提出了SFP算法,在事务数据库D抽取样本数据库S,在样本数据库S而不是D上构建FP - tree。根据创建的FP树挖掘出样本的局部频繁项列表L',然后根据L'挖掘出全局频繁项集,但是有时可能会漏掉一些。这时可以对事务数据库D进行二次扫描以挖掘出漏掉的频繁项集。该算法大多数情况下只需要对数据库进行一次扫描,最坏情况下也只需要对数据库进行二次扫描。不过,由于SFP算法固有的缺陷,不一定能得到最完整的频繁项集。用这种方法,牺牲了一些精度换取了有效性。由此,当采用SFP算法进行挖掘频繁项集时,合理的样本选取以及样本事务数据库的最小支持度阈值的取值将会变得尤为重要。

参考文献:

[1] 韩家炜,堪 博. 数据挖掘概念与技术[M]. 第2版. 范明,孟小峰,译. 北京: 机械工业出版社,2007.

[2] Goebel M, Gruenwald L. A Survey of Data Mining and Knowledge Discovery Software Tools[J]. SIGKDD Explorations,1999,1(5):20-23.

[3] Han J, Pei J. Freespan: Frequent pattern-projected sequential pattern Mining [R]. Vancouver: Si mon Fraser University,2000:6-12.

[4] 蒋良孝. 一种基于FP-增长的决策规则挖掘算法[J]. 计算机科学,2003,32(6):23-25.

[5] 黄学平,薛安荣. 基于数据库划分的关联规则算法[J]. 计算机工程与设计,2008,29(12):3005-3007.

[6] 幼 林,李庆华,刘 干. 最大频繁模式的快速挖掘与更新算法[J]. 计算机工程与应用,2005,41(24):23-26.

[7] 焦明海,姜慧研,唐加福. 一种基于聚合链的改进FP-Growth算法[J]. 东北大学学报,2006,27(2):153-156.

[8] 刘军峰,李景文,陈大克,等. 一种改进的关联规则自顶向下算法[J]. 计算机技术与发展,2008,18(2):136-138.

[9] 毛国君,段立娟. 数据挖掘原理与算法[M]. 北京:清华大学出版社,2005.

[10] Hong J R. AE1: An extension matrix approximate method for general covering problem[J]. International Journal of Computer and Information Science,1985,14(6):421-437.

[11] Mohammed J. Mining non-redundant association rules[J]. Data Mining and Knowledge Discovery, 2004,9(11):223-248.

[12] 李志云, 周国祥. 一种基于MFP树的快速关联规则挖掘算法[J]. 计算机技术与发展,2007,17(6):94-100.