

典型关联规则挖掘算法的分析与比较

冯 洁, 陶宏才

(西南交通大学 信息科学与技术学院, 四川 成都 610031)

摘 要: 关联规则的发现是数据挖掘的一个重要方面, 目前许多研究人员正致力于关联规则的快速开采算法的研究。文中介绍了几种典型的开采大型事务数据库中所有关联规则的算法, 特别针对算法过程中产生候选频繁项集的大小和所需扫描事务数据库的次数这两个影响关联规则挖掘效率的关键问题, 分析各个算法采用的解决策略及相应的局限性, 并比较它们的时间效率和空间效率。最后展望了关联规则挖掘算法的研究方向。

关键词: 数据挖掘; 关联规则; 频繁项集; 算法

中图分类号: TP301.6

文献标识码: A

文章编号: 1673-629X(2007)03-0121-04

Analysis and Comparison of Representative Algorithms for Mining Association Rules

FENG Jie, TAO Hong-cai

(School of Information Science & Technology, Southwest Jiaotong University, Chengdu 610031, China)

Abstract: Discovering association rules is an important data mining problem. Recently, there has been considerable research in designing algorithms for this task. Introduce some representative algorithms for discovering all significant association rules among items in large database of transactions, and it's known that there are two key problems that affect the efficiency of discovering association rules, so analyse the strategy each algorithm adopts separately, and compare their tradeoffs, finally prospects development trends of algorithms of discovering association rules.

Key words: data mining; association rule; frequent patterns; algorithm

0 引 言

数据挖掘是在大量的、不完整的、有噪声的数据中发现潜在的、有价值的模式和数据间关系(或知识)的过程。其中,从数据库中开采关联规则的算法近几年来研究较多。所谓关联规则,就是描述数据库中的数据项(属性、变量)之间所存在的(潜在)关系的规则。

关联规则的问题定义如下: 设 D 是事务集, $I = \{I_1, I_2, \dots, I_m\}$ 是 D 中全体项组成的集合, 其中 I_k ($k = 1, 2, \dots, m$) 称为项, 包含 k 个项的项集称为 k -项集。一个事务 T 是一个项集, 它是 I 的子集, 每个事务都与一个唯一标识符 TID 相联系。不同的事务一起组成事务集 D , 它构成了关联规则发现的事务数据库。若 X, Y 为项集, 且 $X \cap Y = \emptyset$, 则蕴涵式 $X \Rightarrow Y$ 称为关联规则, X 为规则的前件, Y 是结果。事务集 D 中包含

项集 X 的事务个数称为 X 的支持数, 频繁项集即指支持数不低于最小支持数的项集, 最小支持数一般由用户设定。

关联规则挖掘算法的核心任务在于识别事务集 D 中的所有频繁项集, 以便进一步构造相应的关联规则。因此, 文中主要针对频繁项集的挖掘, 对目前比较典型的 10 种关联规则挖掘算法进行了分析和比较。

1 若干典型的关联规则挖掘算法

1.1 Apriori 算法

Apriori 算法^[1]的主要理论依据是项集的两个基本性质: 频繁项集的所有非空子集必是频繁项集; 任何非频繁项集的超集也是非频繁项集。算法采用逐层搜索的迭代, 由连接和剪枝这两步来实现:

① 连接: 采用递推的连接方法求频繁 k -项集 L_k , 用 L_{k-1} 与自身连接产生候选 k -项集的集合 C_k 。

② 剪枝: C_k 是 L_k 的超集, 根据项集的基本性质删除那些具有非频繁子集的候选项集, 然后扫描数据库, 统计候选项集的支持计数, 与最小支持计数相比较, 形

收稿日期: 2006-05-23

作者简介: 冯 洁(1982-), 女, 四川泸州人, 硕士研究生, 研究方向为网络数据库、数据挖掘; 陶宏才, 副教授, 硕士生导师, 研究方向为网络与信息系统、网络安全。

成频繁项集 L_k 。两个步骤反复循环,直到不能产生新的频繁项集为止。

1.2 AprioriTid 算法

AprioriTid 算法^[1]的特点在于只在计算 1-项集的支持计数时用到了数据集 D ,当 $k > 1$ 时使用数据集 D_k^* 替代数据库 D 。如果一个事务不包含任何候选 k -项集,那么 D_k^* 中将不包含这个事务。这样, D_k^* 中元素的个数将会少于数据库中的事务数,并且随着 k 值的增大, D_k^* 中的事务数和事务的长度都将进一步减小,从而有效提高了候选项集的计数速度。

1.3 DHP 算法

DHP 算法^[2]是一种基于哈希函数产生频集的高效算法。在扫描数据库统计 1-项集支持计数的同时,将每个事务中包含的所有 2-项集分别通过哈希函数映射到不同的桶,并对每个桶分别计数。对于散列中的某个桶计数低于支持计数的桶,其包含的 2-项集不可能成为频繁 2-项集,因此删除对应桶中的项集,从而达到压缩 C_2 的目的。

1.4 FARM 算法

Apriori 算法产生的 C_2 具有两个特点: C_2 中所包含的候选 2-项集的数目 $|C_2|$ 由 $|L_1|$ 所决定,且 $|C_2| = |L_1| \times (|L_1| - 1) / 2$ 。FARM 算法^[3] 据此判定 C_2 的生成不是必需的,建立一个包含 $|L_1| \times (|L_1| - 1) / 2$ 个元素的一维数组 A ,替代哈希树统计 C_2 中各候选 2-项集的支持次数。为此,算法保证每一个候选 2-项集对应且只对应于唯一的数组元素。

1.5 StepLength 算法

基于步长(StepLength)的算法^[4]不是逐一产生 k 维频繁项集($k = 1, 2, \dots$),而是在已确定的频繁 k -项集的基础上,同时产生候选 $(k+1)$ -项集,候选 $(k+2)$ -项集, ..., 候选 $(k + \text{步长})$ -项集,再遍历数据库,确定其中真正的频繁项集,如此反复循环,直到所产生的频繁项集的最大长度小于 $(k+l)$,或 L_{k+l} 所蕴涵的频繁 1-项集的个数 n 等于 $(k+l)$,则算法结束,其中步长 L 在每个循环动态确定。

1.6 Auto-Apriori 算法

在 Apriori 算法中,当 k 较小时,随着 k 值的增大,相应 C_k 中候选项集数迅速增大,但当 k 到达某一个值后,随着 k 值的增大,候选项集个数迅速减少。根据 k 值,可将 C_k 中候选项集数量的大小分为增长期、平稳期和迅减期三个阶段。Auto-Apriori 算法^[5] 由此提出一种动态自适应的方式来减少数据库的扫描次数,即:在 k 值达到迅减期的某个 l 之后,便不再一次只生成一个候选项集的集合,而是同时生成 $C_l, C_{l+1}, C_{l+2}, \dots$,

C_{l+t} ,其中 t 值动态确定,然后再通过数据库的一次扫描求得对应的频繁项集的集合 $L_l, L_{l+1}, L_{l+2}, \dots, L_{l+t}$,直到某个频繁项集集合为空为止。

1.7 STBAR 算法

STBAR 算法^[6]将 L_k 有序存储在项集树中,使前 $(k-1)$ 项相同的频繁 k -项集记录在一起,避免在生成候选时的模式匹配。要由 k 层频繁项集生成候选 $(k+1)$ -项集,只需将每一个叶子节点和它的右兄弟节点连接即可。某个候选一经生成,其支持数立即可知,可以直接判断其是否是频繁的,无需记录候选项集。同时算法结合动态剪枝技术,节省了连接所需的时间和空间。

1.8 AVM 算法

AVM 算法^[7]基于向量和矩阵挖掘频繁项集,它赋予每个项 i 一个有序编号,并为之建立一个比特向量 BV_i ;同时定义一个 $M \times (M-1)$ 矩阵 ARM 存储频繁 2-项集,及一个长度为 M 的一维数组 ARV 为每个项目 i 存储比 i 的编号小且和项目 i 能组合成频繁 2-项集的项的个数(M 为数据库中项的个数)。算法扫描一次数据库生成 L_1 ;对 L_1 中各个项所对应的比特向量两两进行逻辑“与”运算,以生成 L_2 ,同时为矩阵 ARM 和数组 ARV 赋值;当 $k > 1$ 时,用频繁 k -项集 $\{i_1, i_2, \dots, i_k\}$ 的最后一项 i_k 来扩展项集,并结合矩阵 ARM 和数组 ARV 中存储的信息动态剪枝,以生成频繁 $(k+1)$ -项集,如此反复,直到不能产生新的频繁项集,则算法结束。

1.9 FP-growth 算法

FP-growth 算法^[8]的主要思想是基于分治策略,即在第一次扫描数据库以后,将产生频繁集的数据压缩到一棵频繁模式树中,又称之为 FP-树,用模式树保留项集的关联信息,然后对模式树产生频繁集。当原始数据量很大的时候,也可结合划分的方法,使得一棵 FP-树能放入主存中。

1.10 HCS-Mine 算法

HCS-Mine 算法^[9]基于哈希链结构挖掘频繁模式,并动态构造链地址。算法针对数据库中的频繁投影逐条进行扫描,每扫描一条就产生相应的项集并记录其支持数,对于新产生的项集,利用哈希函数计算其链地址。

所有链地址相同的项集构成一条链表,链表结点中的计数域为该结点中项集累计出现的次数,各条链表头表结点的计数域为该链表中所有结点的计数之和。这样,在频繁模式挖掘过程中,只需考虑那些头表结点中计数域大于等于最小支持数的结点,从而节省挖掘时间。

2 算法的分析与比较

DHP 算法、FARM 算法、AprioriTid 算法、StepLength 算法、Auto-Apriori 算法、STBAR 算法、AVM 算法都是 Apriori 算法的改进,和 Apriori 算法合称为 Apriori 系列算法,其共同点是都需要借助于上一步产生的频繁项集来产生候选项集。FP-growth 算法、HCS-Mine 算法采用压缩的数据结构存储关联规则挖掘所需的信息,避开了产生候选项集的步骤,减少了数据交换和频繁匹配的开销。比较如下。

Apriori 算法的优点是简单易懂,但同时也存在以下两方面的不足:

① 当事务数据库中频繁 1-项集的数目 $|L_1|$ 较大时,候选 k -项集 C_k 尤其候选 2-项集将非常大,这也是 Apriori 算法的一个效率瓶颈;

② 为了由 C_k 产生 L_k ,需要重复扫描数据库中的事务并计算 C_k 中各候选项集的支持计数,特别当事务数据库中的事务个数较大时,其效率十分低下。

因此,针对 Apriori 算法的改进算法大都从以上两个方面着手对其进行优化。

DHP 算法和 FARM 算法均通过压缩候选项集和缩减数据库大小的方法优化 Apriori 算法:DHP 算法采用哈希技术压缩 C_2 ,有效解决了 Apriori 的效率瓶颈,但同时为了生成循环 2 所用的哈希表,DHP 算法不得不在循环 1 扫描数据库的过程中生成每个事务所包含的所有 2-项集,并将其哈希到哈希表中,这样 DHP 在循环 1 的效率必将低于 Apriori 算法。FARM 算法采用一维整型数组存储候选 2-项集的支持计数,有效改善了上述问题,其效率优于 Apriori 算法和 DHP 算法。

FARM 算法优于 Apriori 算法的原因主要有两个:其一是在循环 2 中使用一维数组代替 Apriori 算法中的哈希树,使得循环 2 对候选 2-项集支持计数的统计处理从原来的对哈希树的搜索处理变成了一次数学运算处理,循环 2 的处理效率得到了很大改善;其二是从循环 3 开始,FARM 算法所扫描的数据库的大小迅速缩小,而 Apriori 算法始终扫描原始数据库。FARM 算法优于 DHP 算法的主要原因是避免了在循环 1 建立哈希表的开销,另外,基于使用数组代替哈希树的同样原因,在循环 2,FARM 算法也快于 DHP 算法。但另一方面,由于 DHP 在循环 2 所处理的候选 2-项集的数量通常大大小于 FARM 算法,它比 FARM 算法能更迅速地缩小后续循环所用的数据库,因此,从循环 3 开始,DHP 算法通常要稍快于 FARM 算法,但 FARM 的总体效率仍优于 DHP 算法。

AprioriTid 算法、StepLength 算法和 Auto-Apriori

算法主要从减少扫描数据库的开销这个角度优化 Apriori 算法:

AprioriTid 算法有效缩减了数据库的大小。然而当 k 值较小时,由于大部分事务会包含几乎所有的频繁 k -项集, D_k^* 中的数据量仍会很大,特别在 D_k^* 不能放进内存时,AprioriTid 的效率甚至不及 Apriori 算法。因此,R. Agrawal 等学者后来又提出了 AprioriHybrid 算法^[10]。作为上述两种算法的折衷:AprioriHybrid 在初期扫描时采用 Apriori 算法,当 D_k^* 能放进内存时就转向 AprioriTid 算法。

StepLength 算法和 Auto-Apriori 算法由于放宽了对候选项集的剪枝条件,均能有效减少数据库的扫描次数,但也相应增大了候选项集的数目。对 StepLength 算法而言,步长 L 的值是影响算法效率的关键:如果 L 取值过小,必然违背算法减少数据库扫描次数的初衷,而 L 若取值过大,则会产生较大规模的候选项集。Auto-Apriori 算法动态确定从迅减期开始一次所求的候选项集集合的个数,灵活性强,但当总交易项目数较小时,Auto-Apriori 产生的候选项集不够精确,算法效率低于 Apriori 算法。

STBAR 算法和 AVM 算法高效生成候选,且无需记录候选项集,优化了 Apriori 的时空效率。

与 Apriori 算法相比,STBAR 算法在连接步不需要作任何比较运算,直接进行拼接即可生成候选;同时结合动态剪枝技术,有效改进了 Apriori 循环的两个步骤,在项集树能一次放入内存的前提下,其时空效率优于 Apriori,尤其适合事务平均长度小、项目数较大的情况。与 STBAR 算法相比,AVM 算法引用的是矩阵和整型数组,矩阵可以压缩存储,它占用的空间比项集树小得多,并且矩阵也易于扩展。与 Apriori 算法相比,AVM 算法基于矩阵和向量存储关联规则挖掘所需的信息,仅扫描一次数据库,且无需记录候选项集。但由于向量长度相当于数据库中事务的长度,因此,当数据库中事务数目很大,且项目在事务中的分布比较分散时,算法效率较低。

FP-growth 算法和 HCS-Mine 算法由于采用压缩的数据结构存储关联规则挖掘所需的信息,无需存储候选项集,省去了 Apriori 算法中哈希树的建立;同时也大大减少了数据库的扫描次数。然而这种对 Apriori 算法的有效改进必须基于其采用的数据结构能一次放入内存为前提,当数据库中项目数相对较大时,算法将面临空间瓶颈。尽管如此,上述两种算法的总体效率仍优于 Apriori 算法。

FP-growth 算法与 Apriori 算法比较,有以下两个好处:

①它用一种高度压缩的结构存储数据库中所有有意义的信息,仅需扫描两次数据库;

②它将长的频集分割成多个长度为 1 的频繁项,一步一步用模式生长的方法产生较长的频集,避免了大量候选项集的产生和测试过程,直接产生频繁模式。

HCS-Mine 算法同样无需产生候选项集,且仅扫描一次数据库。和 FP-growth 算法相比,HCS-Mine 算法采用链表结构:一方面,当数据库更新时,算法更易于扩展;另外,算法不需其他辅助数据结构,尤其对项数相对较小,数据库又很大的情况,算法效率占优。

3 结束语

文中分析比较了几种典型的关联规则挖掘算法,此外,发现最大频繁项集也是关联规则挖掘的一个重要算法思想^[11]:最大频繁项集集合中隐含了所有频繁项集,因此可把发现频繁项集的问题转化为发现最大频繁项集的问题,该方面的研究工作尚不太充分。与此同时,在对关联规则挖掘问题进行大量研究工作的基础之上,研究人员还提出了一些相关的变体算法,如泛化的关联规则、周期关联规则等。

参考文献:

[1] Agrawal R, Srikant S. Fast Algorithms for Mining Association

Rules[C]//VLDB'94. Santiago, Chile: [s. n.], 1994: 487 - 499.

[2] Park J S, Chen M S, Yu P S. An Effective Hash - Based Algorithm for Mining Association Rules[C]//SIGMOD'95. San Jose, CA: [s. n.], 1995: 175 - 186.

[3] 杜孝平, 马秀莉, 唐世渭, 等. 快速关联规则挖掘算法[J]. 计算机工程与应用, 2002(11): 1 - 4.

[4] 郭景峰, 路 燕. 一种数据挖掘关联规则的高效算法[J]. 燕山大学学报, 2001, 25(3): 213 - 216.

[5] 顾泽元, 吕宗宝, 刘兴丽. 频繁项目集发现算法 Apriori 的研究[J]. 黑龙江科技学院学报, 2005, 15(5): 319 - 322.

[6] DE-CHANG P I. STBAR: A More Efficient Algorithm for Association Rules Mining[C]//MLC'05. Guangzhou: [s. n.], 2005: 18 - 21.

[7] 牛小飞, 石 兵. 基于向量和矩阵的挖掘关联规则的高效算法[J]. 计算机工程与应用, 2004(12): 170 - 173.

[8] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]//SIGMOD'00. Dallas, TX: [s. n.], 2000: 1 - 12.

[9] 叶飞跃, 王建东, 陈慧萍, 等. 基于哈希链结构的频繁模式挖掘[J]. 计算机工程与应用, 2004(11): 174 - 176.

[10] 邵峰晶, 于忠清. 数据挖掘原理与算法[M]. 北京: 中国水利水电出版社, 2003: 158 - 162.

[11] 李 超, 余昭平. 基于最大模式的关联规则挖掘研究[J]. 微计算机信息, 2006, 22(2-3): 164 - 165.

(上接第 120 页)

算, $f(q_i)$ 的值发生了变化, 可是频率的相对顺序没有发生改变, 频率较高的查询词排在前面, 频率低的排在列表后面。

表 2 运用文中的方法排序后的结果

查询词	文档数 n_i	文档数 n_i	$P(q_i)$	$f(q_i)$	L_2
办公	19	11	0.039	0.073	6
商务	22	6	0.037	0.051	7
计算机	47	14	0.08	0.19	3
工具	115	92	0.273	0.273	2
财务	8	16	0.03	0.074	5
教学	177	8	0.245	0.132	4
游戏	248	48	0.39	0.746	1

$L_2 = \{\text{游戏, 工具, 计算机, 教学, 财务, 办公, 商务}\}$

比较两个表, 可以看出经公式计算后, 列表 L_1 与 L_2 中查询词的排序相对顺序是吻合的。即可以用部分文档中的查询词得出整个文档中查询词频率的规律。

4 结束语

查询词的选择是 Deep Web 搜索的一个关键问题, 能有效地找到适合的查询词来填入查询接口, 可以提高 Deep Web 的效率。文中将 Zipf Estimator 应用于根

据查询词的频率选择词条的方法中, 提出了用部分文档中的查询词的排序来得出整个文档中查询词的排序的方法, 对提高 Deep Web 的搜索效率有积极作用。今后的工作重点是在更大范围中进一步研究这种方法, 并不断地完善。

参考文献:

[1] Chang K Chen - Chuan, He Bin, Li Chengkai, et al. Structured database on the web: Observations and Implications[J]. SIGMOD Record, 2004, 33(3): 61 - 70.

[2] Raghavan S, Garcia - Molina H. Crawling the hidden web [C]//In VLDB. Roma, Italy: [s. n.], 2001.

[3] Bergman M K. The deep web: Surfacing hidden value[EB/OL]. 2001 - 07. <http://www.press.umich.edu/jep/07-01/bergman.html>.

[4] Ipeirotis P G, Gravano L. Distributed search over the hidden web: Hierarchical database sampling and selection [C]//In VLDB. Hong Kong, China: [s. n.], 2002.

[5] Cormen T H, Leiserson C E, Rivest R L. Introduction to Algorithms[M]. 2nd Edition. [s. l.]: MIT Press/McGraw Hill, 2001.

[6] Zipf G K. Human Behavior and the Principle of Least - Effort [M]. Cambridge, MA: Addison - Wesley, 1949.