一种结合散列与位表挖掘频繁项目集算法

任永功 宋奎勇 寇香霞
(辽宁师范大学计算机与信息技术学院 大连 116029)

摘 要 在频繁项集的挖掘中,很多算法都是基于Apriori的。这些算法有两个共同的问题:一是把整个数据库装入内存,占用大量的空间;二是在产生候选项集和计算支持度时花费了大量的时间。为了提高效率,提出了一种基于位表挖掘频繁项目集的算法Hash-BFI。按照水平和垂直的方向把数据库压缩到位表内,以大大节省内存空间。引入散列函数计算频繁二项集,完全通过AND,OR运算得到候选项集和计算候选项集支持度,并进行剪枝,从而提高了算法效率。

关键词 Apriori,频繁项集,位表,散列

Algorithm Combination of Hash and BitTable for Mining Frequent Itemsets

REN Yong-gong SONG Kui-yong KOU Xiang-xia
(School of Computer and Infomation Technology, Liaoning Normal University, Dalian 116029, China)

Abstract In the frequent itemsets mining, many algorithms are based on Apriori. These algorithms have two common problems. First, a lot of memory space are occupied by the entire database which must be loaded. Second, The processes of generating candidate itemset and computing support spend a lot of time. In order to improve efficiency, a BitTable-based form mining frequent itemsets algorithm Hash-BFI was proposed. The database was compressed into the BitTable in accordance with horizontal and vertical direction saving lots of place, used the hash function to compute the frequent two itemsets, also completely utilized AND, OR operation to generate candidate itemset and compute support for candidate itemset, and producted a pruning. All these meatures improve the efficiency of algorithm.

Keywords Apriori, Frequent itemsets, BitTable, Hash

1 引言

频繁项集挖掘研究是关联规则研究中最基本也是最重要的问题,近些年来,一直是数据挖掘领域的研究热点。Agrawal[1]提出了一种基于广度优先策略逐层搜索频繁项集的Apriori算法,该算法简单明了、容易实现,适合挖掘稀疏的数据集。Han[2]等提出FP-Growth算法,通过两次扫描将数据集压缩到频繁模式树(FP-Tree)的数据结构中,通过深度优先搜索FP-Tree来挖掘频繁项集,这既避免多次扫描数据库,减少I/O开销,又不产生候选项集,极大地提高了频繁项集挖掘效率,特别适合挖掘稠密的数据集。

然而,Apriori算法需要对数据库进行多次扫描,I/O开销较大,而且当最小支持度较小、数据集项数较多时,Apriori算法可能产生大量候选项集,极大地影响算法效率。FP-Growth算法在低支持度下处理稀疏数据库时效率也不高,并且对于海量数据库在内存中构建一个频繁模式树是不现实的。最近几年提出相关的一些新方法,CBAR[3]算法使用聚类表(一种二进制的表)把数据库事务按照项数不同分成几个表,在不同表里计算项集支持度,不用扫描整个数据库,减少了计算支持度的时间,然而它是从Apriori算法中派生出来的,并没有克服Apriori算法的根本缺点。Dong[4]提出一种基于位表的BitTableFI算法,通过位表在水平和垂直方向压缩数据库,运用位运算快速产生候选项集和计算项集支持度,此算法不但节省内存空间,而且提高了效率。但是它没有很好地解决候选二项集规模过大的问题。根据文献[5]所说,最初迭代产生候选项集是相当耗费时间的,尤其是候选二项集的产生。而且BitTableFI算法也没有对候选项集进行剪枝,在后面做了很多无用的操作,浪费了时间。

为了解决上述问题,本文提出一种基于位表挖掘频繁项目集算法Hash-BFI,首先建立位表(包含一组整数,整数的每一个二进制位代表一项),通过位表存储结构高效地压缩数据库,节省内存使用。利用哈希函数对二项集进行处理,解决候选二项集规模过大的问题。基于频繁项集位表和非频繁项集位表通过AND和OR运算产生候选项集并对候选项集剪枝。基于数据库位表完全地通过AND运算计算候选项集支持度,快速位操作大大提高了算法效率,节省了时间。实验表明本文的算法是有效可行的。

2 背景

2.1 定义

设$I=\{i_1,i_2,\cdots,i_m\}$是m个不同项的集合。D是所有事

到稿日期:2010-01-08 返修日期:2010-03-18 本文受国家自然科学基金项目(60603047),辽宁省科技计划项目(2008216014),大连市优秀青年科技人才基金(2008J23JH026),教育部留学回国人员科研启动基金资助。

任永功(1972—),男,博士,教授,主要研究方向为数据挖掘、图像处理技术等,E-mail:renyg@dl.cn;**宋奎勇**(1979—),男,硕士生,主要研究方向为数据挖掘;**寇香霞**(1980—),女,硕士生,主要研究方向为数据挖掘。

务的集合，其中每个事务 T 是项的集合。T 包含在 I 中，即 $T\subseteq I$。并且每个事务都可以用一个唯一的标识符 Tid 来标识。

定义 1 设 X 为 I 中某些项目的集合，简称为项集，一个包含 k 个项的集合称为 k-项集。例如 $X=\{i_1,i_3,i_5\}$，X 是 3-项集。

定义 2 项集 X 在 D 中出现的频率(即 D 中包含 X 的事务 T 的个数)称为 X 在 D 中的支持度。

定义 3 若项集 X 的支持度不小于用户给定的最小支持度，则称 X 为频繁项目集，满足最小支持度要求的 k-项集称为频繁 k-项集，用 L_k 表示。

性质 1 任何频繁项目集的所有非空子集都是频繁的，非频繁项目集的超集是非频繁的。

2.2 Apriori 算法

Apriori 算法是 R. Agrawal 和 R. Srikant 于 1994 年提出为布尔关联规则挖掘频繁项集的原创性算法。Apriori 使用一种称为逐层搜索的迭代方法，通过($k-1$)-项集探索 k-项集。首先，找出频繁 1-项集，记作 L_1，L_1 用于探索 L_2，L_2 用于探索 L_3。如此下去，直至找到所有频繁项集。每层迭代需要对数据库扫描一次。

Apriori 算法通过 L_{k-1} 探索 L_k 主要包括两步：第一步是由 L_{k-1} 产生 C_k(候选 K 项集)。通过将所有 L_{k-1} 互相连接产生 C_k，C_k 是 L_k 的超集，所有的 L_k 都在 C_k 中。然而 C_k 可能很大，这样在计算候选项集支持度的时候，计算量会很大。为了压缩 C_k，可以用性质 1 对 C_k 进行剪枝，即如果 C_k 中候选项集的子集不在 L_{k-1} 中，则该候选项集不可能是频繁的，可以从 C_k 中删除。第二步是由 C_k 产生 L_k。扫描数据库，计算 C_k 中每个项集支持度。如果项集支持度大于等于最小支持度，则并入 L_k 中。

如图 1 所示，一个数据库 D，$I=\{a,b,c,d,e,f,g\}$，D 中有 5 个事务，设定最小支持度为 2。为了操作方便，把项集 $\{a,b,c\}$ 简化为 abc，图 1 展示了 Apriori 算法挖掘频繁项目集的过程。扫描一遍数据库 D，统计每个项出现的次数，把支持度大于等于 2 的那些项集确定为频繁 1-项集 L_1，$L_1=\{a,b,c,e\}$，其他都舍弃。使用 $L_1\infty L_1$ 产生候选 2 项集 C_2。C_2 中每个项集 1-项子集都是频繁的，剪枝在本步不起作用。扫描一遍数据库 D，计算 C_2 中每个候选项集支持度。根据最小支持度确定频繁 2-项集的集合 $L_2=\{ab,ac,ae,be\}$。使用 $L_2\infty L_2$ 产生候选 3-项集 $C_3=\{abc,abe,ace\}$，由于 bc 是 abc 的子集，而 bc 不是频繁的，剪掉 abc。ce 是 ace 的子集，ce 不是频繁的，剪掉 ace。扫描一遍数据库 D，abe 支持度等于 2，则产生频繁 3-项集 abe，扫描结束。

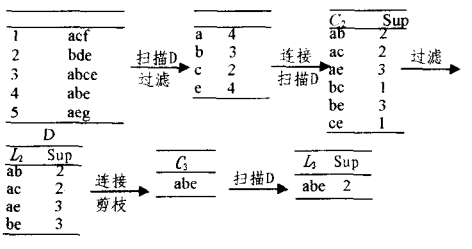

图 1 Apriori 算法的挖掘过程

3 Hash-BFI 算法的实现

3.1 建立数据库位表

位表[5]是由整数构成的一个表，整数的每一二进制位代表一项。如果某一项出现，则位表中相应位置为 1，否则为 0。位表用来压缩候选项集和数据库，对候选项集是横向压缩，对数据库是纵向压缩。表 1 对图 1 中只包含频繁项集的数据库进行了表示，数据库位表 DBT 为{23,14,20,15}。

表 1 数据库位表

Tid	a	b	c	e
1	1	0	1	0
2	0	1	0	1
3	1	1	1	1
4	1	1	0	1
5	1	0	0	1
DBT	23	14	20	15

位表对数据库的压缩是相当理想的。例如一个数据库有 500 个项，其中有 100 个是频繁的，有 100k 个事务，用 32 位整数来表示则位表大小只是源数据库的 1/160。如果位表中频繁项数或者事务总数超过了计算机表示能力，本文用一个整数数组表示位表，整数数组的大小为 $S/W+1$。其中 S 为项或事务个数；W 为整数，表示位的大小。

3.2 引入散列函数产生频繁 2-项集

在 Apriori 算法中，频繁 1-项集 L_1 连接产生候选 2-项集 C_2，C_2 中包含 $|L_1|*(|L_1|-1)/2$ 个候选 2-项集。当 $|L_1|$ 很大时，$|L_1|*(|L_1|-1)/2$ 是一个特别大的数字，$|L_1|*(|L_1|-1)/2$ 越大，产生 L_2 所花费的资源越多，它是 *Apriori* 算法的瓶颈。使用散列函数[6]可以大大提高算法的性能，尤其是产生频繁 2-项集的效率，但是并不能很好地改善 k-项集($k>2$)产生的效率，尤其是当 k 很大时很容易产生冲突。本文用在频繁 2-项集和非频繁 2-项集的产生上，引入文献[9]构造的散列函数，通过散列函数快速产生频繁 2 项集和不频繁 2 项集。

定义 4 散列函数为 $h(x,y)=|L_1|*(order(x)-1)-order(x)*(order(x)-1)/2+order(y)-order(x)$，其中 $|L_1|$ 表示频繁1-项集 L_1 的项目数，$order(x)$表示 x 在 L_1 中的索引，散列表长度为 $|L_1|*(|L_1|-1)/2$。

首先计算散列表长度，得到哈希桶大小，建立哈希桶。扫描数据库每一个事务，找到事务中包含的所有候选 2-项集，通过散列函数计算每一个候选 2-项集的桶地址。根据桶地址压入相应哈希桶中，使相应桶计数加 1。扫描整个数据库，把所有候选信息都压入哈希桶中，桶计数大于等于最小支持度的候选二项集就是频繁二项集。

例 1 如图 1 所示，频繁 1-项集 $L_1=\{a,b,c,e\}$，$|L_1|=4$，散列表长度 $|L_1|*(|L_1|-1)/2=6$，令 $order(a)=1$，$order(b)=2$，$order(c)=3$，$order(e)=4$。扫描数据库中每一个事务，事务 acf 中包含候选项集 ac，计算 $h(1,3)=2$，则 ac 的哈希桶地址为 2，把地址为 2 的桶计数加 1。计算得到其他候选 2-项集的桶地址如图 2 所示。事务 bde 包含候选项集 be，$h(2,4)=5$，则地址为 5 的桶计数加 1。事务 $abce$ 包含候选项集$\{ab,ac,ae,bc,be,ce\}$，则使相应的桶计数加 1。事务 abe 包含候选项集$\{ab,ae,be\}$，则相应的桶计数加 1。事务 aeq 包含事务 ae，则地址为 3 的桶计数加 1。最后通过桶计数和最小

支持度比较，得到频繁二项集 $L_2=\{ab,ac,ae,be\}$ 和非频繁二项集 $UL_2=\{bc,ce\}$。

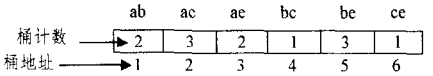

图 2 哈希桶

3.3 快速产生候选项集并对候选项集剪枝

定义 5 给定 $I=\{i_1,i_2,\cdots,i_m\}$ 及所对应的二进制数 b，称 b 中 1 的个数为 b 的长度，记为 Length(b)。

性质 2 对于频繁($k-1$)-项集 X 和 Y，X 对应二进制数为 X_b，Y 对应二进制数为 Y_b，如果 length(X_b or Y_b)$=k$，则项集 $X\cup Y$ 是候选 k-项集。

性质 3 对于任意非频繁($k-1$)-项集 X 和任意候选 k-项集 Y，如果 $X\cap Y=X$，则 Y 是非频繁的。

得到频繁 2-项集和非频繁 2-项集后，开始构建频繁 k-项集位表 BL_k 和非频繁 k-项集位表 UBL_k(k 值从 2 开始)。把 BL_k 中频繁项集相互间进行 OR 运算，并统计结果项集长度。如果项集长度等于 $k+1$，则根据性质 2 此项集作为一个候选($k+1$)-项集，否则舍弃此项集。把该候选($k+1$)-项集与位表 UBL_k 中每一项进行 AND 运算，根据性质 3 位表 UBL_k 中如果存在候选($k+1$)-项集的子集，就可以剪掉这个不频繁($k+1$)-项集。如果不存在，把此候选($k+1$)-项集存入位表 BC_{k+1} 中。通过对所有候选($k+1$)-项集进行剪枝，最后得到包含所有候选($k+1$)项集位表 BC_{k+1}。

算法 1 候选项集的产生

输入：频繁 k 项集位表 BL_k 和非频繁 k 项集位表 UBL_k
输出：剪枝后的候选 $k+1$ 项集位表 BC_{k+1}
pro_condidate(BL_k,UBL_k){
// 从 BL_k 中取出一个频繁 k 项集。
1 for each itemset $I_1\in BL_k$ {
// 从 BL_k 中取出另一个频繁 k 项集。
2 for each itemset $I_2\in BL_k$ {
// 把两个频繁 k 项集按位或运算。
3 $c=I_1\mid I_2$;
// 判断 c 的长度是否等于 $k+1$。
4 If(length(c)=$k+1$) then {
// 取出 UBL_k 中的每一项。
5 for each itemset $u\in UBL_k$
// 如果 UBL_k 没有 c 的子集，则把 c 并入 BC_{k+1} 中。
6 if(c&u! =u) $BC_{k+1}=BC_{k+1}\cup c$;
7 else delete(c);
8 }
9 else delete(c);
10 }}
11 return BC_{k+1}
12 }

例 2 从例 1 中可知，频繁二项集位表的二进制代码为{1100,1010,1001,0101}，相应 BL_2 为{12,10,9,5}。非频繁二项集位表的二进制代码为{0110,0011}，相应 UBL_2 为{6,3}。在 BL_2 上不同整数之间进行 OR 运算，得到整数二进制表示为{1110,1101,1011,1111}，长度不等于 3 的整数被舍弃，15 被舍弃，$BC_3=\{14,13,11\}$。把 BC_3 中每一个整数与 UBL_2 中每一个整数进行 AND 运算，因为 14 AND 6=6，11 AND 3=3，14 和 11 从 BC_3 中被删去。得到剪枝后的 BC_3，$BC_3=\{13\}$。快速产生候选项集并对候选项集剪枝的过程如图 3 所示。

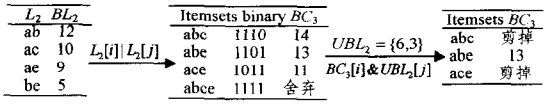

图 3 产生候选项集并剪枝

3.4 候选项集支持度的计算

源数据库被纵向压缩到位表内，在算法执行过程中，整个数据库位表被装进内存，候选项集支持度可以直接在位表中进行计算。对于 BC_{k+1} 中整数二进制表示的每一位，在数据库位表 DBT 中都有一列与其相对应。找到二进制值为 1 所对应 DBT 中的那些列，把这些列的整数值进行 AND 运算，得到整数二进制的长度就是该候选项集支持度。如果长度大于等于最小支持度，则它是频繁的，存入 BL_{k+1} 中。这种候选项集支持度计算方法简单而且效率高，减少了扫描数据库的次数，节省了时间。

算法 2 候选项集支持度的计算

输入：数据库位表 DBT 和候选项集位表 BC_{k+1}
输出：频繁 $k+1$ 项集位表 BL_{k+1}
count_support(DBT,BC_{k+1}){
// 取出 BC_{k+1} 中的每一候选项集。
1 For each itemset $c\in BC_{k+1}$ {
// 把 c 中二进制数等于 1 的位置存入数组 arr 中。
2 arr=position(c);
// 计算 c 在数据库中的支持度 count，arr[k]为 DBT 的第 k 列。
3 count =length(arr[1]&arr[2]&…arr[k]);
// 如果支持度大于最小支持度，则并入 BL_{k+1} 中。
4 If(count>=minsup) $BL_{k+1}=BL_{k+1}\cup c$;
5 else delete(c);
6 }
7 return BL_{k+1};
8 }

例 3 从例 2 中得到 $BC_3=\{13\}$，13 的二进制为 1101，值为 1 的位对应 DBT 列的值为 23，14 和 15，23&14&15=6，6 的二进制为 00110，00110 的长度为 2 等于最小支持度，即 abe 是频繁 3 项集。只有一个频繁 3 项集，循环结束，找到所有的频繁项集{a,b,c,e,ab,ac,ae,be,abe}。频繁 3-项集的产生如图 4 所示。

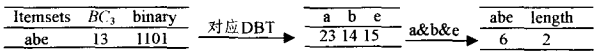

图 4 候选项集支持度的计算

4 实验结果与分析

用 Java 实现了 Apriori，BitTableFI 和 Hash-BFI 算法，所有的实验都在相同环境下进行。实验环境为：Windows XP 系统，2.66GHz 奔腾四处理器，512MB 内存。

使用文献[4]中测试频繁项集的两个标准测试集 Mashroom 和 Pumsh 进行测试。这两个测试集项集是稠密的，即使是在支持度较高的情况下也会产生较长的频繁项集。两个测试集特征如表 2 所列。

表 2　Mashroom 和 Pumsh 的特征

数据库	项数	事务总数	平均长度
Mashroom	120	8124	23
Pumsb	7117	49046	50

图 5 显示 Mashroom 在较低支持度下各算法效率的变化情况。支持度从 14%到 4%，随着支持度的降低，满足最小支持度的频繁项集的长度和个数快速增加，各算法花费的时间也快速增加。其中 Apriori 算法变化最大，花费时间最多，Hash-BFI 较 BitTableFI 所花费的时间要少。

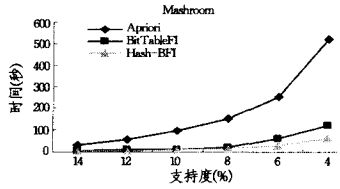

图 5　低支持度下算法性能比较

如图 6 所示，对包含大量项数测试集的 Pumsb 在较高支持度下进行测试，支持度从 50%到 30%。随着支持度的降低，Hash-BFI 算法所花费的时间缓慢增长，花费的时间明显比 BitTableFI 和 Apriori 要少。通过实验和理论分析，本文算法引入散列函数，在处理像 Pumsb 这样包含大量项数的测试集时，大大减少了候选 2-项集产生的个数，突破了 BitTableFI 和 Apriori 算法瓶颈，充分显示本文算法的时间优越性。然而我们也发现，随着测试集的增大和支持度的降低，这几个算法时间花费明显增大。

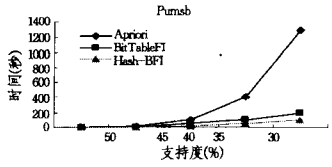

图 6　高支持度下算法性能比较

结束语　针对 BitTableFI 算法的不足提出 Hash-BFI 算法，在处理非常花费资源的二项集上引入散列函数，BitTableFI 算法没有对候选项集进行剪枝。本文算法对候选项集进行有效的剪枝，并且完全通过 AND，OR 运算产生候选项集和计算候选项集支持度。然而，实验过程中也发现了一些问题，由于 Hash-BFI 是基于宽度比较算法，虽然在稀疏测试集上有较大优势，但是在低支持度下处理稠密测试集用时还是比较多。文献[7,8]把位表和深度探索方法相结合，取得了很好的效果，我们下一步要在这方面做一些工作。

参 考 文 献

[1] Agrawal R, Imilienski T, Swami A. Mining association rules between sets of items in large databases[C]// Proceedings of the ACMSIGMOD International Conference on Management of Data. Washington, DC, 1993: 207-216

[2] Han J W, Pei J, Yin Y W, et al. Mining frequent patterns without candidate generation: a frequent-pattern tree approach[J]. Data Mining and Knowledge Discovery, 2004, 8(1): 53-87

[3] Tsay Y J, Chiang J Y. CBAR: an efficient method for mining association rules[J]. Knowledge-Based Systems, 2005, 18(2/3): 99-105

[4] Dong Jie, Han Min. BitTableFI: An efficient mining frequent itemsets algorithm[J]. Knowledge-Based Systems, 2007, 20: 329-335

[5] Burdick D, Calimlim M, Gehrke J, et al. MAFIA: a maximal frequent itemset algorithm[J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(11): 1490-1504

[6] Park J S, Chen M S, Yu P S. Using a hash-based method with transaction trimming for mining association rules[J]. IEEE Transactions on Knowledge and Data Engineering, 1997, 9(5): 813-825

[7] Song Wei, Yang Bing-ru, Xu Zhang-yan. Index-BitTableFI: An improved algorithm for mining frequent itemsets[J]. Knowledge-Based Systems, 2008, 21: 507-513

[8] Duemong F, Preechaveerakul L, Vanichayobon S. FIAST: A novel algorithm for mining frequent itemsets[C]// International Conference on Future Computer and Communication. 2009: 140-144

[9] 柴华昕，王勇. Apriori 挖掘频繁项目集算法的改进[J]. 计算机工程与应用，2007，43(24)：158-171

[10] 朱玉全，杨鹤标，孙蕾. 数据挖掘技术[M]. 南京：东南大学出版社，2006

(上接第 129 页)

[23] Beigel R, Tanin E. The geometry of browsing[C]// Proceedings of the Third Latin Ameriacn Symposium on Theoretical Informatics. 1998: 331-340

[24] Mamoulis N, Papadias D. Selectivity estimation of complex spatial queries[C]// Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Database. 2001: 155-174

[25] Aref W, Samet H. A Cost Model for Query Optimization Using R-Trees[C]// Proceedings of the Second ACM Workshop on Advances in Geographic Information Systems (ACM-GIS). 1994: 60-67

[26] Shekhar S, Chawla S. Spatial Databse: A Tour [M]. Prentice Hall, 2003

[27] Date C J. An introduction to database systems[M]. New York: Addison-Wesley Publishing Company, 1994

[28] Egenhofer M J, Herring J R. Categorizing binary topological relations between regions, lines, and points in geographic databases[C]// Egenhofer M J, Mark D M, Herring J R. editors. The 9-Intersection: Formalism and Its Use for Natural-Language Spatial Predicates. National Center for Geographic Information and Analysis, Report 94-1, 1994: 13-17

[29] Brinkhoff T, Horn H, Kriegel H P, et al. A Storage and Access Architecture for Efficient Query Processing in Spatial Databse Ssystems[C]// Proceedings of 3rd International Symposium on Advaces in Spatial Databases. 1993: 357-376

[30] 陈琳，杜友福，王元珍. MRR：基于 MBR 的空间关系模型[J]. 计算机工程与应用，2002：76-78

[31] Sun C, Agrawal D, Abbadi A El. Selectivity Estimation for Spatial Joins with Geometric Selections[C]// Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology. 2002: 609-626