文章编号：2096-1618(2016)05-0463-06

一种基于 M-Bisearch 的最大频繁项集挖掘算法研究

李宝林， 周 坤， 李仕伟
(西华师范大学计算机学院，四川 南充 637000)

摘要：大数据分析的理论核心就是数据挖掘，关联规则挖掘算法是数据挖掘的重要分支，其包含频繁项集的生成和关联规则的产生两个步骤，频繁项集的生成过程中算法开销占据很大成本。从最大频繁项集的性质入手，在改变数据存储结构的基础上采用 M-Bisearch 的思想，通过对存储空间进行压缩来减少扫描次数和降低支持度计算开销，从而达到提升算法执行效率的目的。实验表明，改进算法在处理中长模式的频繁项集挖掘问题时具有明显的优越性。

关 键 词：机器学习；数据挖掘；关联规则；频繁项集；最大频繁项集；M-Bisearch
中图分类号：TP311 **文献标志码**：A

0 引言

数据挖掘算法是大数据分析的理论核心，而关联规则挖掘算法是众多数据挖掘方法中的一个重要分支，拥有举足轻重的地位，现已成为数据挖掘领域最活跃、最成熟的研究方向。关联规则的目的旨在探寻数据库中不同属性间有用的相互依存关系，主要研究方向有基本关联规则挖掘、针对关联规则评价的研究、复杂类型关联规则挖掘、并行挖掘算法和增量挖掘算法等[1]。Agrawal 与 Srikant 在 1994 年提出的 Apriori 算法是关联规则挖掘算法中最具有代表性和权威的算法，后来的许多算法都是在 Apriori 算法的基础上进行优化和扩展而来的[2]。如 Jiawei Han 提出的 FP-Growth 算法，该算法改变了数据的存储方式，将数据集压缩在一棵 FP-tree 上，但当数据非常庞大时 FP-tree 也将非常庞大，进行数据挖掘时将产生大量的频繁模式基，严重影响算法性能。由 FP-tree 算法思想可知，提高算法效率可以从存储数据的结构出发，简化数据的 I/O 操作；关联规则算法的核心步骤在于生成全部频繁项的集合，根据关联规则的性质—最大频繁项集的子集都是频繁的和最大频繁项集的超集都是非频繁的，故得出最大频繁项集可以生成全部频繁项。论文基于存储结构和关联规则的性质两个因素进行综合考虑，在垂直数据库结构(vertical data structure，VDS)的基础上采用 M-Bisearch 的思想对 Apriori 算法进行优化改进。

1 相关工作

经典 Apriori 算法存在几个不足方面：(1)在生成候选项集时产生了大量非频繁项集；(2)连接操作 Apriori-gen 函数增加了算法的复杂度；(3)每迭代一次就扫描一次事务数据库 D。国内外学者做了大量研究，提出一系列的改进算法。文献[2]缩小待扫事务的条数，减少扫描数据库 D 的时间；文献[3]利用 Hash 技术有效地生成候选项集 C_k，同时利用只有事务长度不小于项集长度才有可能包含项集的性质减少了扫描数据库时间；文献[4]基于划分的思想将数据库逻辑地划分成几个独立块，在块的基础上调用相关算法进行频繁项集的挖掘，然后验证每个块的频繁项集，生成整个数据库的频繁项集。减少搜索数据库的开销；文献[5]改变项目集的存储结构，采用动态项集计数的方式减少扫描数据库的次数；文献[6]使用深度优先的搜索方法，首先扫描数据库构建一个称为 lexicographic 树的结构，然后利用深度优先的搜索方式遍历 lexicographic 树发现所有的最大频繁项集；文献[7]采用十字链表的方式存储事务集，同时对候选项集的生成方法进行优化，减少对数据库的扫描次数；文献[8]改变数据的存储形式，从事务包含项变为项包含事务，同时改进产生候选项集的连接方法减少重复连接；文献[9]对 2_项集使用了散列表技术，对长度大于 2 的项集采用矩阵方式存储；文献[10]在十字链表的基础上利用频繁项集的子集必是频繁的这一性质，优化了候选频繁项集的生成和数据库扫描；文献[11]采用分治策略将数据集分块处理，使用事务项数进行矩阵压缩，同时利用向量交运算和先验剪枝直接生成分块频繁 k_项集。从而得到全部频繁 k_项集。算法改进迭代流程，减轻连接运算带来的开销；文献[12]提出一种具有跳跃式前进与回退补齐的 Apriori 改进算法，通过频繁 k_项集得到候选 $2k$_项集，减少数据库扫描次

收稿日期：2016-09-13
基金项目：四川省科技厅支撑资助项目(2013SZ0056)

数。

上述对 Apriori 算法的改进都是着手于数据存储结构和生成候选项集的连接方式,避免多次扫描数据库和降低算法时间复杂度。但上述算法有的需要大量存储空间、有的数据结构过度复杂、有的依然迭代开销很大,在降低算法的存储复杂度和迭代上还有待改进。通过对上述文献的研究学习,现提出一种新的改进方法:使用垂直数据库结构,采用 M-BISEARCH 的思想,找出最大频繁项集,再根据最大频繁项集的性质直接生成所有频繁项。该方法通过简化数据存储方式、支持度计数方法和频繁项集生成过程来提高算法效率。经过实验论证,基于 M-Bisearch 算法对数据的挖掘效率有很大的提高。

2 相关概念

2.1 定义及性质

设 $I=\{i_1,i_2,i_3\cdots,i_k\}$ 是 k 个不同项组成的集合,则称集合 I 为项集(Itemset),其元素个数称为项集的长度,当一个项集的长度为 k 时,称该项集为 k 维项集,简称 k_项集(k_Itemset)。事务数据库 D 是事务 T 的集合,$T\subseteq I$。

定义 1 对于一个集合 X,$X\subseteq I$ 且 $X\neq\phi$,则将 D 中包含 X 的事务条数与总的事务条数之比称为 X 的支持度,记为 Support(X)。用户定义的最小支持度记为 min_sup[13-14]。

定义 2 对于项集 X,如果 Support(X)≧min_sup,则称项集 X 是频繁项集;

定义 3 对于频繁项集 X,如果其所有超集均为非频繁项集,那么称项集 X 为最大频繁项集[15-16];

性质 1 假设频繁项集为 X,若 $\forall Xi\subset X$,且 $X_i\notin\phi$,则 X_i 一定是频繁的;

性质 2 假设存在集合 super_Item,并且最大频繁项集 Max_frequent_Item⊂super_Item,则 super_Item 一定是非频繁的[17-18];

性质 3 若一个项集 X 是非频繁的,则项集 X 的超集 super_item 也是非频繁的。

2.2 VDS

垂直数据库结构(vertical data structure,VDS),是相对于水平数据库结构而言。传统的数据库是以事务为单位,用事务包含数据项,而 VDS 是以项为单位,将包含该项的事务的集合组成 TidList 列表,称之为支持事务列表。TidList 列表包含了包含该项的事务的序号,同时对项进行支持度计数,且按降序排列。VDS 能在保证事务数据完整性的基础上,减少事务数据库中数据的存储量,降低数据冗余。同时改变项集计数的方式减少扫描数据库的次数。

为更为直观地了解水平数据库结构如何转换成垂直数据库结构,用如下例子演示,见表 1、表 2。

表 1 水平数据库结构

事务序号	事务
1	AB
2	CDE
3	ACEF
4	CDE
5	ABCE
6	BDF
7	ABE
8	CE
9	EF
10	ADE

表 2 转化后的垂直数据库结构

项	支持事务列表	支持度计数
E	2、3、4、5、7、8、9、10	8
A	1、3、5、7、10	5
C	2、3、4、5、8	5
B	1、5、6、7	4
D	2、4、6、10	4
F	3、6、9	3

由上可得,VDS 具有几点性质:

性质 4 VDS 包含两个数组元素,其中,项存储了原始数据库中所有的频繁 1_项集;支持事务列表(TidList)存储了各个频繁 1_项集对应的支持事务序号 TID。

性质 5 若 2_项集 $I_2=\{m,n\}$ 是由 VDS 中数组元素项的任意两项构成,那么 $\{m,n\}$ 的支持事务列表是 m、n 的支持事务列表的交集。即 $\{m,n\}$. TidList = m. TidList∩n. TidList。

性质 6 若 k_项集 $L_k=\{i_{m1},i_{m2},i_{m3},\cdots,i_{mk}\}$,则 L_k 的支持事务列表为所有项 $i_{m1},i_{m2},i_{m3},\cdots,i_{mk}$ 的支持事务列表的交集。即 L_k. TidList = i_{m1}. TidList∩i_{m2}. TidList ∩i_{m3}. TidList∩⋯∩i_{mk}. TidList。

3　算法改进

3.1　改进思路

首先,同文献[8]类似,先将原始事务数据库转化为 VDS 的数据库,且按项的支持事务列表中事务的计数降序排列。在此基础上同时生成事务长度计数表,该表包括事务长度与达到该长度事务的个数。数据库结构的转化如表 1、表 2 所示,沿用上节例子,事务长度计数表如表 3 所示。

表 3　事务长度统计表

事务长度	事务计数
2	3
3	5
4	2

然后,从大到小计算各事务长度区间内事务的个数,找出一个长度值 M,使 M 与大于 M 的事务长度值的事务计数和满足最小支持度阈值,同时大于 M 的事务长度值的计数和要小于最小支持度阈值。

M 值的确定,如下所示:

$$\min\_sup \leq \frac{\sum_{M}^{N} T:N\_\text{Items} \subseteq T, T \subseteq D}{D}, N \geq M$$

$$\text{且} \frac{\sum_{M+1}^{N} T:N\_\text{Items} \subseteq T, T \subseteq D}{D} \leq \min\_sup, N \geq M+1 \quad (1)$$

最后,根据二分查找的思想寻找到全部最大频繁项集,根据性质 1 生成全部频繁项集。按照 M 值与最短频繁集长度 1 向上取整数平均值,设 $(M+1)/2=i$,根据频繁 1_项集生成 i_项集,对相应项的支持事务列表进行相交操作,得出支持度计数表同 min_sup 进行比较,设 x 属于 i_项集,若 count(x) ≥min_sup,则计算 $(i+M)/2$ 向上取整,若 count(x)<min_sup,则计算 $(1+i)/2$ 向上取整,按照这样二分查找的方法继续进行支持度比较,直到下一个长度与当前项集长度相差为一时,直接采用另一个长度生成候选项集,最后使得长度为 k 的项集 I_k 为频繁项集,长度为 $k+1$ 的项集 $I_{k+}1$ 为非频繁项集,则将 I_k 加入最大频繁项集集合 MF。最终找到所有的最大频繁项集,然后根据最大频繁项集的性质,生成所有频繁项集 L_k。

与 Apriori 算法类似,改进算法同样要进行连接和剪枝。文中算法通过频繁 1_项集的连接操作,得出 k_项集 P_k,接着通过 TidList 的相交运算得出 P_k 的支持度,若 P_k 为非频繁项集,则将其归入剪枝集合 P-set,并向下进行二分查找;当 P_k 为频繁项集时,向上进行二分查找,得到的新的候选项集 P_{new} 与 P-set 进行比较,并进行剪枝操作。

3.2　示例说明

沿用表 1 的数据,设 min_sup = 0.4,则 VDS 数据库如表 4 所示,事务长度统计仍如表 3 所示。

表 4　VDS 数据库

项	支持事务列表	支持度计数
E	2、3、4、5、7、8、9、10	
A	1、3、5、7、10	5
C	2、3、4、5、8	5
B	1、5、6、7	4
D	2、4、6、10	4
F	3、6、9	3

因为 min_sup = 0.4,事务长度的支持度分别为 support(4) = 2/10, support(3) = 7/10, support(2) = 10/10,所以最大频繁项集长度的最大值 M 为 3。根据二分查找思想在长度区间[1,3]上查找最大频繁项集。(1+3) = 2,所以从 2_项集开始挖掘。扫描 VDS 数据库得到频繁 2_项集如表 5 所示。

表 5　频繁 2_项集

项	支持事务列表	支持度计数
EA	3、5、7、10	4
EC	2、3、4、5、8	5

因为是 2_项集,所以没满足支持度阈值的项集直接采用 1_项集作为最大频繁项集。而 EA,EC 的支持度满足 min_sup=0.4 的条件,又因为此时候选项集长度为 2 与上限长度 3 比邻,所以不用向上取整数平均值,直接计算另一个长度值的候选项集,即 EAC,EAD,ECD,根据非频繁项集的超集必为非频繁项集的性质进行剪枝操作,则候选项集变为 EAC,此时 EAC 的支持度计数通过如下相交运算得出:E. TidList ∩ A. TidList ∩ C. TidList = 2,Support(EAC) <min_sup,所以 EAC 为非频繁项集. 因为 EA,EC 本身为频繁项集,且他们的直接超集均为非频繁项集。可得 EA、EC 为最大频繁项集。因此得到 D、B,EA,EC 4 个最大频繁项集,根据最大频繁项集的子集是所有的频繁项集集合的性质,求出有 E,A,C,D,B,EA,EC 7 个频繁项集。

根据 Apriori 算法,将得到 A,B,C,D,E,AE,CE 7

个频繁项集。与改进算法得到的频繁项集完全一样。通过例子分析,可知改进算法是有效的和准确的。后面将进一步用实验对改进算法的性能进行验证。

3.3 算法描述

输入:D-原始事务数据库,min_sup-最小支持度阈值

输出:频繁项集 *Fk*

Generate_VDSandTable(D);//扫描原始数据库生成垂直数据库 VD 和事务长度统计表 Table

Max_length (Table);

Right_length=Max_length;

Left_length=1;

Generate_VDS(VD,min_sup);

For (i=0;i<Item. size-1;i++){

添加频繁项集到 L_1

}

For (k=2;k! =null; k) {

P_k= join (L_1, k);

C_k= Cut (P_k);

L_k=Count (C_k);

if(k = Right_length 且 k_项集是非频繁项 or k = Left_length且 k_项集是频繁项){

then 就把此时的 k_项集写入最大频繁项集集合(Max_frequent_itemsets);

}

If (Length. C_k ≥min_sup){

If (Right_length-k>1)

k=(k+Right_length)/2 向上取整;

Else k= Right_length;

Else if (k-Left_length>1)

k=(k-Left_length) /2 向上取整;

Else k= Left_length;

}

P_k=ϕ;

C_k=ϕ;

}。

函数说明:

Generate_VDandTable(D):扫描原始数据库 D,生成垂直数据库 VD 和事务长度统计表 Table;

Max_length (Table):从事务长度统计表中计算出频繁项集的最大长度值;

Generate_VDS(VD,min_sup):根据垂直数据库和 min_sup,删除非频繁的 1_项集;

join(L_1, k):通过频繁 1_项集的相交运算,得出 k_项集 P_k;

Cut(P_k):根据非频繁项集的超集必是非频繁的这一性质进行剪枝操作,生成候选 k_项集 C_k;

Count(C_k):对候选项集 C_k进行支持度计数,设 $C_k=\{i_{m1},i_{m2},\cdots,i_{mk}\}$,则其计数是各个支持事务列表的交集,即 L_k. TidList = i_{m1}. TidList ∩ i_{m2}. TidList ∩ i_{m3}。TidList∩…∩i_{mk}. TidList。同时和 min_sup 相比较,生成频繁项集 L_k。

4 实验结果与分析

为验证算法的性能,将改进算法与 Apriori 算法进行对比分析的同时也和文献[7]中的 Apriori_BL 算法进行对比。

实验环境:Intel(R) Core(TM) i-5 3470 CPU @ 3.20 GHz 3.20 GHz;内存4.00 GB;650 G 硬盘;Windows7 * 64 位操作系统。

算法实现方式:实验涉及的所有算法都通过 Java 实现。

实验 1:数据采用文献[11]中的 Extended BAKERY 数据集,该数据集共有 75000 个事务,每条事务包括 8 种属性,每种属性有 0 至 50 个属性值。随机取 50000 条数据进行实验,结果如图 1 和表 6 所示。

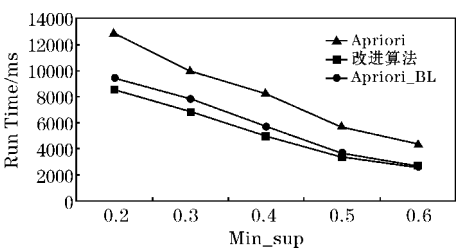

图 1 在 Extended BAKERY 上的挖掘时间

表 6 在 Extended BAKERY 上的挖掘结果比较

支持度	频繁项集个数	
	Apriori 算法	改进算法
0.2	5217	5217
0.3	3346	3346
0.4	1834	1834
0.5	1029	1029

由图 1 可知,针对 Extended BAKERY 数据集这种短模式频繁项集挖掘问题,改进算法较之文献[8]中只是单独的改进存储结构的 Apriori_BL 算法,当支持度阈值较低时,算法效率有一定的提高,但是差距不是很大。随着支持度阈值的增大,两者的差距逐渐缩小,当 Min_sup=0.6时,因为文中算法的复杂性导致其运

行时间大于 Apriori_BL 算法的运行时间。

由表 6 可知,改进算法的挖掘结果和 Aprior 算法的挖掘结果一致无遗漏,进一步说明改进算法的正确性和完整性。

实验 2:仍然使用 Extended BAKERY 数据集,测试 3 种算法随着数据集大小的改变(事务个数从 5000 至 75000 个),其运行时间的变化情况,设置支持的阈值 Min_sup=0.2。实验结果如图 2 所示。

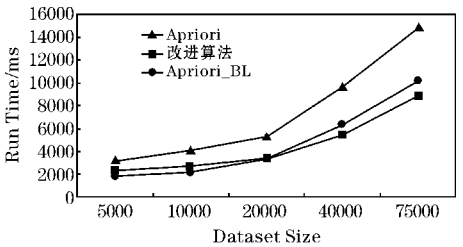

图 2　3 种算法对不同事务集大小的运行时间

由图 2 可知,当事务集较大时,改进算法的运行时间明显低于其他两种算法的运行时间,但随着事务集逐渐减小,文中算法运行时间的减少速度是没有其他两种算法的减小幅度大,当事务集大小降为 20000 时,Aprior_BL 算法的运行时间已经低于改进算法。

实验 3:数据集采用 SuperMarket 数据集,该数据超市 4627 条交易记录,包括 217 样商品的购买情况。实验结果如图 3 所示。

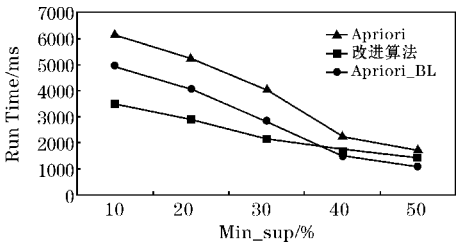

图 3　在 SuperMarke 上的挖掘时间

由图 3 可知,当针对 SuperMarket 数据集这种长模式的频繁项集挖掘时,改进算法在 Min_sup 小于30 % 时,运行效率较 Apriori 算法和 Aprior_BL 算法缩短显著,但随着 Min_sup 增大时,改进算法与其他两种算法运行时间的差距减少明显,特别是当 Min_sup=40 % 时,Aprior_BL 算法的运行效率已经高于改进算法的运行效率。这是因为改进算法是根据最大频繁项集的长度进行最大频繁项集的查找,着手于项集的长度,在较小事务集长模式的频繁项集挖掘中支持度阈值对其影响很小。由图 3 可以看出,文中改进算法随支持度阈值的增加,算法的运行时间减少不是很快速,而其他两者算法时间减少速度明显。

参考文献:

[1] 吴喜萍. 基于关联规则数据挖掘技术的高校学生学习成绩分析[D]. 成都:西南交通大学,2010.

[2] Agrawal R, Srikant S. Fast Algorithms for Mining Association Rules [C]. VLDB'94. Santiago, Chile,1994:487-499.

[3] Park J S, Chen M, Yu P S. An Effective Hash-Based Algorithm for Mining Association Rules [C]. SIGMOD'95. Sanjose, CA,1995: 175-186.

[4] Savasere A,omieeinski E, Navathe S. An efficient algorithm for mining association rules in large databases[C]. Proceedings of the 21st International Conference on Very large Database, 1995.

[5] Brin S,Motwani R,Ullman J,et al. Dynamic Itemset Counting and Implication Rules for Market Basket Data[C]. Proc. of 1997 ACMSIGMOD Int'1 Conf. on Management of Data. ACM Press,1997: 255-264.

[6] Charu C. Aggarwal,towards long pattern generation in dense databases[J]. ACM SIGKDD Explorations Newsletter,2001,3(1).

[7] 黄建明,赵文静,王星星. 基于十字链表的 Apriori 改进算法[J]. 计算机工程,2009,(2):37-38.

[8] 刘华婷,郭仁祥,姜浩. 关联规则挖 Apriori 算法的研究改进[J]. 计算机应用与软件,2009,26(1):146-148.

[9] 栗晓聪,滕少华. 频繁项集挖掘的 Apriori 改进算法研究[J]. 江西师范大学学报:自然科学版,2011,35(5):498-501.

[10] 刘玉文. 基于十字链表的 Apriori 算法的研究与改进[J]. 计算机应用与软件,2012,29(5):267-369.

[11] 郑麟. 一种直接生成频繁项集的分治 Apriori 算法[J]. 计算机应用与软件, 2014,(4):297-301.

[12] 陈方健,张明新,杨昆. 一种具有跳跃式前进的 Apriori 算法[J]. 计算机应用与软件,2015,(3):34-36,92.

[13] 宋余庆,朱玉全,孙志挥,等. 一种基于频繁模式树的约束最大频繁项目集挖掘及其更新算法[J]计算机研究与发展,2005,42(5):777-783.

[14] 颜跃进,李舟军,陈火旺. 基于 FP-Tree 有效挖掘最大频繁项集[J]. 软件学报,2005,(2): 215-222.

[15] 林佳雄,黄战. 基于数组向量的 Apriori 算法改

进[J]. 计算机应用与软件,2011,28(5):268-271.
[16] 付沙,宋丹. 基于矩阵的 Apriori 改进算法研究[J]. 微电子学与计算机,2012,29(5):156-160.
[17] 刘红星,王崇骏,谢俊元. 基于图的最大频繁项集的生成算法[J]. 南京大学学报:自然科学版,2008,44(5):520-526.
[18] 陈向华,刘可昂. 基于 FP-Tree 的最大频繁项目集挖掘算法[J]. 软件,2015,12:98-102.
[19] 杨鹏坤,彭慧,周晓锋,等. 改进的基于频繁模式树的最大频繁项集挖掘算法——FP-MFIA[J]. 计算机应用,2015,(3):775-778.

Research on Mining Algorithm of Maximal Frequent Itemsets based on M-blsearch

LI Bao-lin, ZHOU Kun, LI Shi-wei
(1. College of Computer Science, China West Normal University, Nanchong 637000, China)

Abstract: Data mining is the core of big data analysis, and association rule mining algorithm is an important branch of data miningwhich contains two steps: the generation of frequent itemsets and the generation of association rules. The process of generating frequent itemsets in overhead occupies a large cost. This paper starts with the nature of the maximal frequent itemsets, adopts the idea of M-bisearch on the basis of hanging data storage structure, reduces computation cost of the scanning times and the support degree though compressing storage space, so as to achieve the goal of improving the efficiency of the algorithm.

Key words: machine learning; data mining; association rules; frequent itemsets; maximum frequent itemsets; M-bisearch