基于 Aproiri 算法的频繁项集挖掘优化方法①

吴学雁, 莫 赞

(广东工业大学 管理学院, 广州 510520)

摘 要: 为了进一步降低扫描数据库的次数和减轻内存负担, 从而更好地提高挖掘频繁项集的效率, 一种基于Apriori 的优化算法(M-Apriori)被提出. 该方法通过构建频繁状态矩阵来存放项集的频繁状态, 构建事务布尔矩阵来存放事务与项集的关系, 此算法只需在初始化阶段扫描一次数据库产生初始的频繁状态矩阵和事务布尔矩阵, 并在此基础上直接递推产生所有的频繁项集. 实验证明, 与 Apriori 算法相比, M-Apriori 算法具有更好的性能与效率.

关键词: 频繁项集挖掘; M-Apriori 算法; 关联规则挖掘

Frequent Itemsets Mining Optimization Methods Based on Aproiri Algorithm

WU Xue-Yan, MO Zan

(School of Management, Guangdong University of Technology, Guangzhou 510520, China)

Abstract: To reduce the number of database scanning and reduce the burden of memory further, also to improve the efficiency of mining frequent itemsets better, an Apriori-based optimization algorithm (M-Apriori) is proposed. The method stores frequent itemsets state by constructing the frequent state matrix and store the relationship between the transaction and itemsets by constructing the Boolean matrix. The algorithm scans the database only once and generates the initial frequent state matrix and the Boolean matrix during the initialization phase. On this basis, all frequent itemsets can be found directly without scanning the database repeatedly. Experiments show that M-Apriori algorithm has better performance and efficiency compared with the Apriori algorithm.

Keywords: frequent itemsets mining; M-Apriori algorithm; association rule mining

关联规则挖掘是用于发现大量数据中项集之间的潜在关系, 从而发现未知规律, 为决策提供依据的一项数据分析技术. 该项研究最早由Agrawal等人提出, 起初应用于购物篮分析, 其目的是发现交易数据库中不同商品之间的联系, 以帮助零售商有选择地推销, 从而引导消费. 随着研究的不断深入和扩展, 如今已成功应用于金融、通信、互联网等领域, 展现了其潜在的研究价值和巨大的应用前景.

Agrawal R等人于1994年提出了经典的Apriori算法, 该算法首先根据用户给定的最小支持度找出所有的频繁项集, 然后再由频繁项集产生满足最小置信度的强关联规则, 而频繁项集的产生是整个挖掘算法的核心. 经典的Apriori算法简单且容易操作, 但是存在着两个重大的问题: (1)在频繁项集的计算过程中需要产生大量的候选项集; (2)需要重复地扫描数据库, 每产生一次候选项集都需要扫描一次数据库. 针对以上两个问题, 研究者进行了广泛的研究, 通过各种技术(例如, 散列, 事务压缩和动态项集计数等技术)来提高Apriori算法的效率, 但是都只取得了有限的效果.

AprioriTid算法[1]仅在计算频繁1项集的时候对数据库进行一次扫描, 然后使用 C_{k-1} 来计算项集的支持度得到频繁k项集,减少了I/O操作的时间和需要扫描的数据大小. 但是, 在循环的初始阶段, AprioriTid算法产生的候选项集个数非常多, 因此在后面的求解过程

① 基金项目:国家自然科学基金(71171062);教育部人文社科青年基金(13YJCZH200);广东工业大学高教研究基金(2012ZY26)
收稿时间:2013-11-05;收到修改稿时间:2013-12-13

中非常耗时. 崔旭等[2]利用粗糙集的特征属性约简算法对属性进行约简, 寻找核心属性数据, 然后构建了约简决策表, 并在此基础上采用改进的Apriori算法对核心数据进行数据挖掘, 最终得到频繁项集. 张敏等[3]提出了一种基于向量和矩阵的VMA算法, 该算法只扫描数据库一次, 将事务数据库转化到布尔向量中, 对频繁1-项集按支持度大小进行非递减排序, 生成一个2-项集支持度矩阵, 由频繁k-项集(k≥2)扩展生成频繁(k+1)-项集. 刘步中[4]提出了Inter-Apriori频繁项集挖掘算法, 该算法使用交集策略减少扫描数据库的次数, 从而使算法达到较高的效率. 符丽锦等[5]提出从先删减后连接的新角度来生成频繁项集, 达到减少无用连接, 进而减少剪枝步骤候选项集判断数量来改进Apriori算法. 另外, 文献[6-9]中也提出了许多有效的改进算法. 这些改进算法在提高挖掘频繁项集效率的同时也引起了一些新的问题, 例如可能增加内存负担、扫描数据库次数仍然过多等等. 为了进一步降低扫描数据库的次数和减轻内存负担, 从而更好地提高挖掘频繁项集的效率, 本文提出了一种基于Apriori的优化算法.

1 基于Aproiri的频繁项集挖掘优化算法(M-Apriori)

本文提出了一种优化的 M-Apriori 算法, 该方法构建频繁状态矩阵来存放各项集的频繁状态(频繁计数), 构建事务布尔矩阵来表示事务与项集之间的关系. 此算法在初始化阶段仅仅扫描数据库一次即可产生初始的频繁状态矩阵和事务布尔矩阵, 并在此基础上直接递推产生频繁项集, 因此, 大大提高了算法的效率.

M-Apriori 算法可分为两部分: 第一部分为初始化阶段, 根据事务数据库 D 产生初始的频繁状态矩阵 V 和事务布尔矩阵 T, 具体在 1.2 小节描述; 第二部分是递推产生频繁 k 项集阶段, 直到找到所有的频繁项集, 具体在 1.3 小节描述.

1.1相关定义

定义 1(频繁状态矩阵 V). 频繁状态矩阵 V 是一个 $m\times m$ 的矩阵, 其中 m 是项的个数, 矩阵的元素 $V(i,j)$ 表示第 i 个项和第 j 个项组成的 2 项集的频繁状态. 频繁状态矩阵 V 的性质如下:

(1) 矩阵的元素 $V(i,j)$ 满足 $1\le i\le m$, $2\le j\le m$ 且 $i<j$;

(2) 矩阵的元素 $V(i,j)$ 的取值范围为 $2\le V(i,j)\le a$, 其中 a 表示数据库 D 中事务具有的最大项集中项目的个数;

(3) $V(i,j)=k$ 表示第 i 个项和第 j 个项组成的 2 项集是某个频繁 k 项集的子集

定义 2(事务布尔矩阵 T)[10]. 事务布尔矩阵 T 是一个 $n\times m$ 的矩阵, 其中 n 是事务的个数, m 是项的个数. 矩阵 T 的元素 $T(i,j)$ 表示第 j 个项是否存在于第 i 个事务中, 如果存在, 则 $T(i,j)$ 被置为 1, 否则, $T(i,j)$ 被置为 0. 通过事务布尔矩阵 T, 每一个项集可以获得一个事务支持标志, 事务支持标志由 n 个 0、1 数值组成. 如果第 i 位取 1, 则表示第 i 个事务包含此项集, 如果取 0, 则表示第 i 个事务不包含此项集. 例如, 对于第 1 个项 I_1, 设它的事务支持标志为“100011001001”, 记为 $\{I_1\}.index=100011001001$, 表示数据库中共有 12 个事务, 其中, 第 1、5、6、9 和 12 个事务中包含了 I_1.

定理 1 若 $I_1I_2\cdots I_k$ 是频繁 k 项集, 则 $I_2I_3\cdots I_k$ 一定是频繁 $(k-1)$ 项集.

证明: (1) $I_2I_3\cdots I_k$ 是 $I_1I_2\cdots I_k$ 的子集;

(2)假设 $I_2I_3\cdots I_k$ 不是频繁 $(k-1)$ 项集, 则根据 Apriori 算法的性质: 任何非频繁的 $(k-1)$ 项集都不可能是频繁 k 项集的子集, 可以推出 $I_1I_2\cdots I_k$ 不是频繁 k 项集;

(3)假设与 $I_1I_2\cdots I_k$ 是频繁 k 项集相矛盾, 因此, $I_2I_3\cdots I_k$ 一定是频繁 $(k-1)$ 项集.

1.2初始化阶段

初始化阶段的主要任务为产生初始频繁状态矩阵 V 和事务布尔矩阵 T, 具体步骤如下(伪代码如图 1 所示):

(1) 为每个项集分配相应顺序号, 并将所有项集按顺序排列;

(2) 为每个事务分配相应顺序号, 并将所有事务按顺序排列;

(3)逐行扫描事务数据库 D 并产生事务布尔矩阵 T;

(4)设置最小支持度计数 min_sup, 扫描事务布尔矩阵 T 产生初始的频繁状态矩阵 V, 具体步骤如下:

1)计算事务布尔矩阵 T 的第 1 列 I_1 的事务支持标志 $\{I_1\}.index$ 与其他列 I_j $(m\ge j>1)$ 的事务支持标志 $\{I_j\}.index$ 的交集, 结果记为 $\{I_1I_j\}.index$, 若 $\{I_1I_j\}.index$ 中“1”的个数大于等于 min_sup, 则将 $V(1,j)$ 置为 2;

2)计算事务布尔矩阵 T 的第 2 列 I_2 的事务支持标志 $\{I_2\}.index$ 与其他列 I_j $(m \geq j > 2)$ 的事务支持标志 $\{I_j\}.index$ 的交集, 结果记为 $\{I_2 I_j\}.index$, 若 $\{I_2 I_j\}.index$ 中“1”的个数大于等于 *min_sup*, 则将 $V(2, j)$ 置为 2;

3)以此类推, 计算事务布尔矩阵 T 的第 i 列 I_i 的事务支持标志 $\{I_i\}.index$ 与其他列 I_j $(m \geq j > i)$ 的事务支持标志 $\{I_j\}.index$ 的交集, 结果记为 $\{I_i I_j\}.index$, 若 $\{I_i I_j\}.index$ 中“1”的个数大于等于 *min_sup*, 则将 $V(i, j)$ 置为 2;

4)执行步骤 3)直到事务布尔矩阵 T 的第(m-1)列 I_{m-1}.

```
输入:  事务数据库 D, 最小支持度 min_sup
输出:  事务布尔矩阵 T, 初始的频繁状态矩阵 V
Begin
    对所有的项集排序得到 I1,I2,…,Im
    对所有的事务排序得到 TID1,TID2,…,TIDn
    For i=1 to n        //生成事务布尔矩阵 T
        For j=1 to m
            If Ij ∈ TIDi
                Then T(i,j)=1
            EndIf
        Endfor
    Endfor
    For i=1 to m-1      //生成初始的频繁状态矩阵 V
        For j=i+1 to m
            If {IiIj}.index 中“1”的个数不小于 min_sup
                Then V(i,j)=2
            EndIf
        Endfor
    Endfor
End
```

图 1　M-Apriori 算法的初始化阶段的伪代码

1.3递推产生频繁项集阶段

递推产生频繁项集的主要任务是根据事务布尔矩阵 T 和频繁状态矩阵 V 来产生频繁 k 项集列表. 在递推过程中, 每一次都会更新相应的频繁状态矩阵 V. 以下描述在频繁(k-1)项集的基础上寻找频繁 k 项集的过程(伪代码如图 2 所示).

从频繁状态矩阵 V 的第 1 行开始进行如下操作, 直到最后一行.

对频繁状态矩阵 V 第 i 行 $(1 \leq i \leq m)$ 的所有元素进行扫描, 寻找 $V(i, j) \geq k-1$ 的元素, 设共找到 l_i 个元素, 对应的列序号为 $j_1, j_2, \cdots, j_{l_i}$.

(1)若 $l_i \geq k-1$, 则对第 $j_1, j_2, \cdots, j_{l_i}$ 个项组成的 l_i 项集进行检查. 从 l_i 项集中可以得到多个 $(k-1)$项子集, 对照已有的频繁 $(k-1)$项集列表, 如果满足以下两个条件, 则此 $(k-1)$项子集与第 i 个项构成的 k 项集为频繁项集, 将其写入频繁 k 项集, 并将此 $(k-1)$项子集中产生的所有 2 项子集对应在矩阵 V 中的元素值赋为 k.

1)此 $(k-1)$项子集是频繁项集;

2)此 $(k-1)$项子集与第 i 个项组成的 k 项集的事务支持标志中“1”的数目大于 *min_sup*.

(2)若 $l_i < k-1$, 则扫描下一行.

```
输入:  频繁 (k-1) 项集 Lk-1, 频繁状态矩阵 V, 事务布尔矩阵 T, 最小支持度 min_sup
输出:  频繁 k 项集 Lk
Begin
    For i=1 to m
    对频繁状态矩阵 V 的第 i 行统计值不小于 (k-1) 的矩阵元素个数为 li
        If li ≥ k-1
        Then 对 li 项集产生所有的 (k-1) 项子集, 得到集合 Ck-1;
        对 Ck-1 中的所有元素 ck-1;
            If ck-1 ⊂ Lk-1
            Then ck-1∞Ii 产生 k 项集 ck;
                If {ck}.index 中“1”的数目不小于 min_sup
                Then 将 ck 写入 Lk
                EndIf
            EndIf
        EndIf
    Endfor
End
```

图 2　M-Apriori 算法递推产生频繁项集阶段的伪代码

2 算法示例

2.1 示例数据库

表 1　示例数据库 D

事务标识号(TID)	项集列表
T01	I_1, I_2, I_5, I_6, I_9
T02	I_2, I_4, I_8
T03	I_2, I_3, I_7, I_9
T04	I_1, I_2, I_4
T05	I_1, I_3, I_4, I_6, I_8
T06	I_2, I_3, I_6, I_9
T07	I_1, I_3, I_6, I_9
T08	I_1, I_2, I_3, I_5, I_8
T09	I_1, I_2, I_3, I_9
T10	I_1, I_3, I_6, I_9
T11	I_2, I_5, I_7, I_9
T12	I_3, I_5, I_7
T13	I_1, I_2, I_4, I_7

T14	I_1，I_2，I_5，I_8

2.2 初始化阶段示例

(1)对数据库 D 中的项集 $I_1,I_2,\cdots,I_9$ 进行排序, 从 I_1 到 I_9 依次分配顺序号 1 到 9.

(2)对数据库 D 中的事务 T01,T02,…,T14 进行排序, 从 T01 到 T14 依次分配顺序号 1 到 14.

(3)扫描数据库 D 产生事务布尔矩阵 T, 如表 2 所示.

(4)设置最小支持度计数为 3, 扫描事务布尔矩阵 T 产生初始的频繁状态矩阵 V.

1)计算第 1 列.

$$\{I_1I_2\}.index=\{I_1\}.index\wedge\{I_2\}.index$$
$$=\{10011011110011\}\wedge\{11110101101011\}=\{10010001100011\}$$

$\{I_1I_2\}$ 的事务支持标志中有 6 个“1”, 大于最小支持度计数, 故 $V(1,2)=2$.

以此类推计算, 得到以下结果:

$\{I_1I_3\}.index=\{00001011110000\}$, $V(1,3)=2$

$\{I_1I_4\}.index=\{00011000000010\}$, $V(1,4)=2$

$\{I_1I_5\}.index=\{10000001000001\}$, $V(1,5)=2$

$\{I_1I_6\}.index=\{10001010010000\}$, $V(1,6)=2$

$\{I_1I_7\}.index=\{00000000000010\}$

$\{I_1I_8\}.index=\{00001001001000\}$, $V(1,8)=2$

$\{I_1I_9\}.index=\{10000010110000\}$, $V(1,9)=2$

表 2 数据库 D 的事务布尔矩阵 T

	1 (I_1)	2 (I_2)	3 (I_3)	4 (I_4)	5 (I_5)	6 (I_6)	7 (I_7)	8 (I_8)	9 (I_9)
1(T01)	1	1	0	0	1	1	0	0	1
2(T02)	0	1	0	1	0	0	0	1	0
3(T03)	0	1	1	0	0	0	1	0	1
4(T04)	1	1	0	1	0	0	0	0	0
5(T05)	1	0	1	1	0	1	0	1	0
6(T06)	0	1	1	0	0	1	0	0	1
7(T07)	1	0	1	0	0	1	0	0	1
8(T08)	1	1	1	0	1	0	0	1	0
9(T09)	1	1	1	0	0	0	0	0	1
10(T10)	1	0	1	0	0	1	0	0	1
11(T11)	0	1	0	0	1	0	1	0	1
12(T12)	0	0	1	0	1	0	1	0	0
13(T13)	1	1	0	1	0	0	1	0	0
14(T14)	1	1	0	0	1	0	0	1	0

2)参照步骤 1)计算第 2 至 8 列, 在此只列出符合条件的频繁 2 项集的结果及其对应的频繁状态矩阵 V 的取值.

$\{I_2I_3\}.index=\{00100101000000\}$, $V(2,3)=2$

$\{I_2I_4\}.index=\{01010000000010\}$, $V(2,4)=2$

$\{I_2I_5\}.index=\{10000001000001\}$, $V(2,5)=2$

$\{I_2I_7\}.index=\{00100000001010\}$, $V(2,7)=2$

$\{I_2I_8\}.index=\{01000001000001\}$, $V(2,8)=2$

$\{I_2I_9\}.index=\{10100100101000\}$, $V(2,9)=2$

$\{I_3I_6\}.index=\{00001110010000\}$, $V(3,6)=2$

$\{I_3I_9\}.index=\{00100110110000\}$, $V(3,9)=2$

$\{I_6I_9\}.index=\{10000111001000\}$, $V(6,9)=2$

3)按照以上结果填写初始的频繁状态矩阵 V, 如表 3 所示.

表 3 初始的频繁状态矩阵 V

	1 (I_1)	2 (I_2)	3 (I_3)	4 (I_4)	5 (I_5)	6 (I_6)	7 (I_7)	8 (I_8)	9 (I_9)
1(I_1)	—	2	2	2	2	2	—	2	2
2(I_2)	—	—	2	2	2	—	2	2	2
3(I_3)	—	—	—	—	—	2	—	—	2
4(I_4)	—	—	—	—	—	—	—	—	—
5(I_5)	—	—	—	—	—	—	—	—	—
6(I_6)	—	—	—	—	—	—	—	—	2
7(I_7)	—	—	—	—	—	—	—	—	—
8(I_8)	—	—	—	—	—	—	—	—	—
9(I_9)	—	—	—	—	—	—	—	—	—

同时得到所有的频繁 2 项集:

$\{I_1I_2\}$, $\{I_1I_3\}$, $\{I_1I_4\}$, $\{I_1I_5\}$, $\{I_1I_6\}$, $\{I_1I_8\}$, $\{I_1I_9\}$, $\{I_2I_3\}$, $\{I_2I_4\}$, $\{I_2I_5\}$, $\{I_2I_7\}$, $\{I_2I_8\}$, $\{I_2I_9\}$, $\{I_3I_6\}$, $\{I_3I_9\}$, $\{I_6I_9\}$

2.3 递推产生频繁项集示例

(1)产生频繁 3 项集

1)扫描频繁状态矩阵 V(如表 3)的第 1 行来寻找 $V(1,j)\geq 2$ 的元素, 得到 7 个元素, 列序号分别为 2, 3, 4, 5, 6, 8, 9, 得到 7 项集 $\{I_2I_3I_4I_5I_6I_8I_9\}$. 先考察 2 项集 $\{I_2I_3\}$ 是否为频繁项集. 从频繁状态矩阵可以看到 $\{I_2I_3\}$ 是频繁 2 项集, 但是由于 $\{I_1I_2I_3\}.index=\{00000001100000\}$, $\{I_1I_2I_3\}$ 不满足最小支持度计数. 同理, 考察 2 项集 $\{I_2I_5\}$, $\{I_2I_5\}$ 是频繁 2 项集, 并且 $\{I_1I_2I_5\}.index=\{10000001000001\}$, $\{I_1I_2I_5\}$ 满足最小支持度计数, 将 $\{I_1I_2I_5\}$ 写入频繁 3 项集列表, 并将 $\{I_1I_2I_5\}$ 产

生的所有 2 项集 $\{I_1I_2\}$, $\{I_2I_5\}$ 和 $\{I_1I_5\}$ 在频繁状态矩阵 V 的对应值置为 3, 即 $V(1,2)=V(2,5)=V(1,5)=3$. 以此类推, 又可以找到频繁 3 项集 $\{I_1I_3I_6\}$, $\{I_1I_3I_9\}$ 和 $\{I_1I_6I_9\}$, 在频繁状态矩阵 V 中更新 $V(1,3)=V(3,6)=V(1,6)=V(1,9)=V(3,9)=V(6,9)=3$. 更新后的频繁状态矩阵 V 如表 4 所示.

2)扫描频繁状态矩阵 V(如表 4)的第 2 行来寻找 $V(2,j)\geq 2$ 的元素, 得到 6 个元素, 列序号分别为 3, 4, 5, 7, 8, 9, 得到 6 项集 $\{I_3I_4I_5I_7I_8I_9\}$. 同理对产生的每一个 2 项集进行考察, 得到频繁 3 项集 $\{I_2I_3I_9\}$, 在频繁状态矩阵 V 中更新 $V(2,3)=V(2,9)=V(3,9)=3$. 更新后的频繁状态矩阵 V 如表 5 所示.

3)以此类推, 在扫描第 3 行的时候得到频繁 3 项集 $\{I_3I_6I_9\}$, 在频繁状态矩阵 V 中更新 $V(3,6)=V(3,9)=V(6,9)=3$. 由于这几项在前一次更新中已经更新了, 所以更新后的频繁状态矩阵 V 没有改变, 还是如表 5 所示. 由于第 4 到 9 行都不满足扫描的条件, 所以频繁 3 项集寻找完毕.

最终得到的频繁 3 项集为: $\{I_1I_2I_5\}$, $\{I_1I_3I_6\}$, $\{I_1I_3I_9\}$, $\{I_1I_6I_9\}$, $\{I_2I_3I_9\}$, $\{I_3I_6I_9\}$.

(2)产生频繁 4 项集

1)扫描频繁状态矩阵 V(如表 5)的第 1 行来寻找 $V(1,j)\geq 3$ 的元素, 得到 5 个元素, 列序号分别为 2, 3, 5, 6, 9, 得到 5 项集 $\{I_2I_3I_5I_6I_9\}$. 对产生的每一个 3 项集进行考察, 只有 $\{I_2I_3I_9\}$ 和 $\{I_3I_6I_9\}$ 是频繁 3 项集, 但是: $\{I_1I_2I_3I_9\}.index=\{000000001000000\}$,不满足最小支持度计数, $\{I_1I_3I_6I_9\}.index=\{000000100100000\}$,不满足最小支持度计数, 不更新频繁状态矩阵 V.

2)扫描频繁状态矩阵 V(如表 5)的第 2 行来寻找 $V(2,j)\geq 3$ 的元素, 得到 3 项集 $\{I_3I_5I_9\}$, 不是频繁 3 项集.

3)第 3 至 9 行不满足扫描条件, 寻找频繁 4 项集完毕, 最终未得到任何频繁 4 项集.

表 4　第 1 次更新后的频繁状态矩阵 V

	1 (I_1)	2 (I_2)	3 (I_3)	4 (I_4)	5 (I_5)	6 (I_6)	7 (I_7)	8 (I_8)	9 (I_9)
1(I_1)	—	3	3	2	3	3	—	2	3
2(I_2)	—	—	2	2	3	—	2	2	2
3(I_3)	—	—	—	—	—	3	—	—	3
4(I_4)	—	—	—	—	—	—	—	—	—
5(I_5)	—	—	—	—	—	—	—	—	—
6(I_6)	—	—	—	—	—	—	—	—	3
7(I_7)	—	—	—	—	—	—	—	—	—
8(I_8)	—	—	—	—	—	—	—	—	—
9(I_9)	—	—	—	—	—	—	—	—	—

表 5　第 2 次更新后的频繁状态矩阵 V

	1 (I_1)	2 (I_2)	3 (I_3)	4 (I_4)	5 (I_5)	6 (I_6)	7 (I_7)	8 (I_8)	9 (I_9)
1(I_1)	—	3	3	2	3	3	—	2	3
2(I_2)	—	—	3	2	3	—	2	2	3
3(I_3)	—	—	—	—	—	3	—	—	3
4(I_4)	—	—	—	—	—	—	—	—	—
5(I_5)	—	—	—	—	—	—	—	—	—
6(I_6)	—	—	—	—	—	—	—	—	3
7(I_7)	—	—	—	—	—	—	—	—	—
8(I_8)	—	—	—	—	—	—	—	—	—
9(I_9)	—	—	—	—	—	—	—	—	—

3　实验分析

3.1　实验框架

(1)数据集

Mushroom 数据集是网络上公开发布的用于测试频繁项集挖掘的数据集, 来源于 http://fimi.cs.helsinki.fi/data/. 本文采用此数据集进行算法测试.

(2)实验方案

使用数据集来比较 Apriori 算法和 M-Apriori 算法, 主要比较两种算法寻找频繁项集的效率与效果.

3.2　实验结果分析

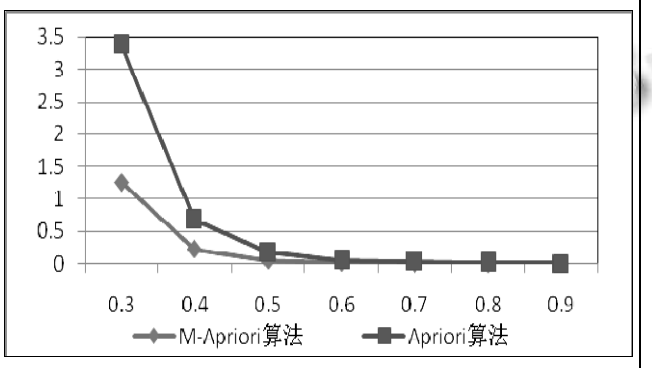
使用 Apriori 算法和 M-Apriori 算法对 Mushroom 数据集进行计算得到的频繁项集结果如表 6 所示, 可以发现, 两种算法得到的频繁项集结果是一样的. 另外, 图 3 展示了两种算法对数据集进行频繁项集查找的运算时间对比, 横轴代表最小支持度的不同取值, 纵轴表示算法的运算时间. 其中, 图 3(b)专门显示了最小支持度取值为 0.6 至 0.9 之间时, 两种算法运算时间的对比. 从图 3 中可以看出, M-Apriori 算法的效率是明显高于 Apriori 算法的. 因此, 通过实验可以得到以下结论: Apriori 算法和 M-Apriori 算法在计算频繁项集的效果上是一致的, 但是在同等的效果下, M-Apriori 算法具有更好的运算效率.

表 6 Apriori 和 M-Apriori 算法在频繁项集上的对比

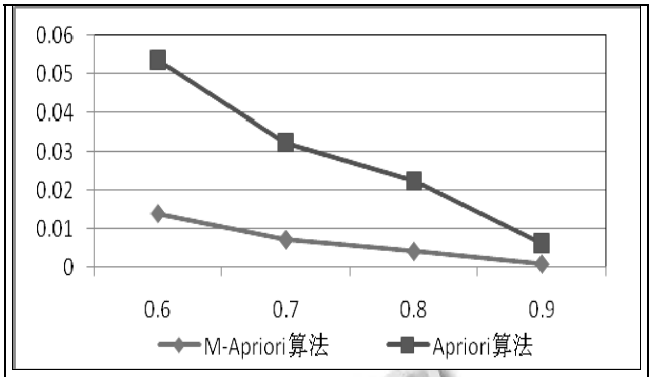
最小支持度	算法	2 项集	3 项集	4 项集	5 项集	6 项集	7 项集	8 项集	9 项集
0.3	Apriori	163	455	725	712	441	169	38	4
	M-Apriori	163	455	725	712	441	169	38	4
0.4	Apriori	97	185	170	76	15	1	—	—
	M-Apriori	97	185	170	76	15	1	—	—
0.5	Apriori	41	56	35	8	—	—	—	—
	M-Apriori	41	56	35	8	—	—	—	—
0.6	Apriori	18	17	7	1	—	—	—	—
	M-Apriori	18	17	7	1	—	—	—	—
0.7	Apriori	10	10	5	1	—	—	—	—
	M-Apriori	10	10	5	1	—	—	—	—
0.8	Apriori	9	7	2	—	—	—	—	—
	M-Apriori	9	7	2	—	—	—	—	—
0.9	Apriori	4	1	—	—	—	—	—	—
	M-Apriori	4	1	—	—	—	—	—	—

4 结 语

本文在 Apriori 算法的基础上提出了一种优化的 M-Apriori 算法, 该方法通过构建频繁状态矩阵来存放项集的频繁状态, 构建事务布尔矩阵来存放事务与项集的关系. M-Apriori 算法只需在初始化阶段扫描一次数据库产生初始的频繁状态矩阵和事务布尔矩阵, 并在此基础上直接递推产生所有频繁项集. 实验证明, M-Apriori 算法具有较好的效果和效率.



a)



b)

图 3 Apriori 和 M-Apriori 算法在运算时间上的比较

参考文献

1 何云峰.关联规则算法研究及在股市中的应用[学位论文].成都:西南交通大学,2006

2 崔旭,刘小丽.基于粗糙集的改进Apriori算法研究.计算机仿真,2013,30(1):329–332,385

3 张敏,姚良威,侯宇.基于向量和矩阵的频繁项集挖掘算法研究.计算机工程与设计,2013,34(3):939–943

4 刘步中.基于频繁项集挖掘算法的改进与研究.计算机应用研究,2012,29(2):475–477

5 符丽锦,覃华,邓海等.一种改进的Apriori算法.广西科学院学报,2013,29(1):1–3

6 Chen C, Shen J, Chen B, et a1. An improvement Apriori arithmetic based on rough set theory. 2011 Third Pacific Asia Conference on circuits, Communications and System. 2011. 1–3.

7 Chen Z, Cai S, Song Q, et a1. An improved Apriori algorithm based on pruning optimization and transaction reduction. 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce. 2011. 1908–1911.

8 吕桃霞,刘培玉.一种基于矩阵的强关联规则生成算法.计算机应用研究,2011,28(4):1031–1033.

9 张文东,尹金焕,贾晓飞,等.基于向量的频繁项集挖掘算法研究.山东大学学报(理学版),2011,46(3):31–34.

10 Burdick D, Calimlim M, Flannick J, et al. Mafia: A maximal frequent itemset algorithm. IEEE Trans. on Knowledge and Data Engineering, 2005, 17(11): 1490–1504.