第 35 卷 第 1 期 计 算 机 工 程 2009 年 1 月
Vol.35 No.1 Computer Engineering January 2009
·软件技术与数据库· 文章编号：1000—3428(2009)01—0084—03 文献标识码：A 中图分类号：TP301.6

基于矩阵的频繁项集挖掘算法

张忠平，李 岩，杨 静
(燕山大学信息科学与工程学院，秦皇岛 066004)

摘 要：如何高效地挖掘频繁项集是关联规则挖掘的主要问题。该文根据集合论和矩阵理论，提出一种基于矩阵的频繁项集挖掘算法。该算法只需扫描数据库一次，就能把所有事务转化为矩阵的行，把所有项和项集转化为矩阵的列，在对矩阵操作时能一次性产生所有频繁项集，且当支持度阈值改变时无需重新扫描数据库。实验结果表明，该算法的挖掘效率高于 Apriori 算法。

关键词：数据挖掘；频繁项集；Apriori 算法

Frequent Itemsets Mining Algorithm Based on Matrix

ZHANG Zhong-ping, LI Yan, YANG Jing
(College of Information Science & Engineering, Yanshan University, Qinhuangdao 066004)

【Abstract】 How to mine the frequent itemsets efficiently is a main problem in association rule mining. According to the theory of congregation and matrix, a frequent itemsets mining algorithm based on matrix is proposed. Through scanning database only once, all transactions are transformed to be rows of matrix and all items and itemsets are transformed to be columns of matrix. This algorithm can one-off product all frequent itemsets, and need not rescan the database when support threshold value changes. Experimental results show the mining efficiency of this algorithm is higher than Apriori algorithm.

【Key words】 data mining; frequent itemsets; Apriori algorithm

1 概述

关联规则挖掘是指从大量数据集中挖掘人们感兴趣的、相关联的项集，以指导人们作出决策。关联规则最初由美国IBM Alinaden Research Center的Agrawal R等人于 1993 年提出的[1]，关联规则挖掘的核心问题是挖掘频繁项集。传统的频繁项集挖掘算法可以分为 2 类：(1)以Apriori算法为代表的采用逐层搜索的迭代方式，即从大量候选项集中产生频繁项集的算法，需多次扫描数据库。(2)文献[2]提出的不产生候选项集的FP-growth算法，只需 2 次扫描数据库。第 1 次扫描数据库时产生频繁 1-项集，第 2 次扫描数据库时对产生的频繁 1-项集排序并构造FP-tree，然后对FP-tree进行挖掘产生频繁项集。目前，在挖掘频繁项集的研究中，很多组织和研究人员围绕这 2 个算法从不同角度对其进行深入研究。

文献[3]提出一种基于数组的关联规则挖掘算法，该算法扫描事务数据库一次，对每个事务进行分解成单个项目存放在二维数组中，同时对各个项目进行统计，找出频繁 1-项集，再利用 Apriori_gen 中的 join 算法生成临时候选项集并剪枝，该算法在对数组进行处理时，要不断对候选项目集进行删减操作。文献[4]提出一种基于矩阵的频繁项集发现算法，通过一次扫描数据库把事务数据库转化成矩阵，对矩阵的向量进行逐点乘法运算计算项集的支持数，但它仍是采用逐层搜索的思想，耗费大量时间。文献[5]提出一种基于布尔向量关系运算的关联规则挖掘算法，扫描数据库一次把事务数据库转化为布尔矩阵，直接通过对矩阵的列向量进行对位“与”运算产生频繁项集。每个频繁 k-项集的产生都要进行大量的剪枝操作，每剪枝一次就要重新计算矩阵最后一列的值，因而计算量较大。

尽管文献[3-5]在扫描事务数据库时都只扫描一次，但也存在着一些不足之处。例如，都是沿用了 Apriori 的思想，而且对数组或矩阵反复操作，浪费了大量时间。因此，本文针对以上问题，应用集合论和矩阵理论，提出一种全新的基于矩阵的频繁项集挖掘(Frequent Itemset Mining Based on Matrix, FIMM)算法，该算法只需扫描数据库一次就能把事务数据库转化成矩阵，然后对项集向量进行累加操作以实现对项集的计数，由此一次性产生所有频繁项集，有效提高了挖掘效率。

2 FIMM 算法

2.1 问题定义

设$I=\{i_1,i_2,\cdots,i_m\}$是项的集合，事务数据库$D=\{T_1,T_2,\cdots,T_n\}$，其中，每个事务T是项的集合，$T\subseteq I$。如果$X\subseteq T$，则称X是个项集。如果X中有k个元素，则称X为k-项集。对于一个项集X，如果其支持度大于等于用户给定的最小支持度阈值minsup，则X为频繁项集。关联规则就是类似于$X\Rightarrow Y$的一些蕴含式，其中，$X\subseteq I$, $Y\subseteq I$且$X\cap Y=\varnothing$。规则$X\Rightarrow Y$的支持度就是指事务数据库D中同时包含X和Y的事务的数量与所有事务数量的比率。

定义 1 设长度为零的项集为空项集，记为$\varnothing$。

定义 2 已知项的集合$I=\{i_1,i_2,\cdots,i_m\}$，事务数据库D，则D中的一个事务T产生的项集的集合称为T的项集集合，记为$IS(T)$，每个项集集合中都包含空项集。

基金项目：教育部科学技术研究基金资助重点项目(205014)；河北省教育厅科研计划基金资助项目(2006143)
作者简介：张忠平(1972－)，男，副教授、博士后，主研方向：数据挖掘，XML 数据库，网格技术；李 岩、杨 静，硕士研究生
收稿日期：2008-06-10 **E-mail：**shanshi1024@163.com

由集合论知，包含m个项的事务数据库，产生的所有可能的项集个数为 2^m，其中，一个事务T产生的项集的集合为$IS(T)$，产生的所有项集的个数为 $2^{|T|}$。$|T|$为事务T中项的个数，其中，$|T|\leq m$。

因为事务数据库D中有 2^m个不同的项集，所以有必要把这些项集按照一定的方式排列[6]，本文通过采用循环迭代增加项作为模式后缀的方式排列所有项集，这种排列方法更便于理解和计算。

定义 3 设事务数据库D中项的集合为$I=\{i_1,i_2,\cdots,i_m\}$，令空项集 $\varnothing$ 表示项集p_0，则项集的排列可以递归定义为

$$\begin{cases} p_0=\varnothing \\ p_k=p_{2^\rho-(k \bmod 2^\rho)-1}\oplus i_\rho \end{cases} \tag{1}$$

其中，$\rho=\lfloor \mathrm{lb}k \rfloor+1$，$0<k<2^m$；$\oplus$表示连接符号。

定理 1 设事务数据库D中项的集合为$I=\{i_1,i_2,\cdots,i_m\}$，D的所有项集都按定义 3 的方式排列，则项集 $p_{2^{\rho-1}},p_{2^\rho-2},\cdots,p_{2^\rho-1}$ 可以通过增加项i_ρ为项集 $p_0,p_1,\cdots,p_{2^{\rho-1}-1}$ 的后缀得到，其中，$1\leq\rho\leq m$。

证明：根据式(1)中的 $p_k=p_{2^\rho-(k \bmod 2^\rho)-1}\oplus i_\rho$ 可知，式(1)表示项集p_k由项集 $p_{2^\rho-(k \bmod 2^\rho)-1}$ 与项i_ρ连接构成，如果给定ρ的值，则能够得到k的值为 $2^\rho-1,2^\rho-2,\cdots$ 或 $2^{\rho-1}$，令 $k=2^\rho-1$，则得 $p_k=p_{2^\rho-1}\oplus i_\rho=p_{2^\rho-((2^\rho-1) \bmod 2^\rho)-1}\oplus i_\rho=p_0\oplus i_\rho$，即项集 $p_{2^\rho-1}$ 可以由i_ρ连接到项集p_0的后缀得到，类似地，$p_{2^\rho-2},p_{2^\rho-3},\cdots,p_{2^{\rho-1}}$ 可以由i_ρ连接到项集 $p_1,p_2,\cdots,p_{2^{\rho-1}-1}$ 的后缀得到。

一个事务 T 产生的项集与事务数据库 D 产生的所有项集有着密切的联系。

定理 2 设事务数据库D中项的集合为$I=\{i_1,i_2,\cdots,i_m\}$，其中，事务T产生的项集集合为$IS(T)$，如果项i_ρ不包含于事务T，则项集 $p_{2^\rho-1},p_{2^\rho-2},\cdots,p_{2^{\rho-1}}$ 不包含于$IS(T)$，否则项集 $p_{2^\rho-1},p_{2^\rho-2},\cdots,p_{2^{\rho-1}}$ 是否包含于$IS(T)$分别依赖于项集 $p_0,p_1,\cdots,p_{2^{\rho-1}-1}$ 是否包含于$IS(T)$，其中，$1\leq\rho\leq m$。

证明 根据式(1) $p_k=p_{2^\rho-(k \bmod 2^\rho)-1}\oplus i_\rho$，其中，$\rho=\lfloor \mathrm{lb}k \rfloor+1$，可知式(1)表示项集$p_k$由项集 $p_{2^\rho-(k \bmod 2^\rho)-1}$ 与项i_ρ连接组成。

先证明i_ρ不在事务T中的情况。如果给定ρ的值，则能够得到k的值为 $2^\rho-1,2^\rho-2,\cdots$ 或 $2^{\rho-1}$，根据定义 2，$IS(T)$中的每个项集都是由事务T中的项构成，因此，如果项i_ρ不在T中时，由项i_ρ组成的项集p_k即 $p_{2^\rho-1},p_{2^\rho-2},\cdots$ 或 $p_{2^{\rho-1}}$ 不在$IS(T)$之中。

再证明i_ρ在事务T中的情况。项集p_k是否在$IS(T)$中与项集 $p_{2^\rho-(k \bmod 2^\rho)-1}$ 是否在$IS(T)$中有关，根据上面的证明，k值为 $2^\rho-1,2^\rho-2,\cdots$ 或 $2^{\rho-1}$，令 $k=2^\rho-1$，则 $p_k=p_{2^\rho-1}\oplus i_\rho=p_{2^\rho-((2^\rho-1) \bmod 2^\rho)-1}\oplus i_\rho=p_0\oplus i_\rho$，因此，如果项$i_\rho$在事务$T$中，则项集 $p_{2^\rho-1}$ 是否在$IS(T)$中依赖于项集p_0是否在$IS(T)$中，类似地可得，项集 $p_{2^\rho-2},p_{2^\rho-3},\cdots,p_{2^{\rho-1}}$ 是否在$IS(T)$中依赖于项集 $p_1,p_2,\cdots,p_{2^{\rho-1}-1}$ 是否在$IS(T)$中。

2.2 算法描述

FIMM 算法的基本思想是：根据式(1)项集的排列方式，把事务数据库产生的所有项集按顺序存放在矩阵中，以作为矩阵的列，行用来存储所有的事务。为便于找到事务产生的项集与所有项集之间的关系，把所有的项也按照字母表顺序存放于矩阵的前 m 列。如果该项属于该事务则置交叉位为 0，否则为 1，如果该项集包含于事务产生的项集，则置交叉位为 1，否则为 0。最后累加存储项集的列中 1 的个数，存放于矩阵的最后一行，计数值大于等于最小支持数阈值的项集则为频繁项集。

FIMM算法先扫描数据库，把事务数据库转化为矩阵，用来存储事务数据库中所有事务和项集。对于包含n个事务，m个项的事务数据库D，项目集$I=\{i_1,i_2,\cdots,i_m\}$，并建立一个n+1行$m+2^m$列的矩阵$\boldsymbol{M}$，前n行用来存储所有的事务，最后一行用来存储累加结果，前m列用来存储所有的项(按字母表顺序排列)，后 2^m列用来存储所有可能的项集(按定义 3 项集的排列方式)。对于前m列，当项$I_j\in T_i$时，其中，$1\leq j\leq m,1\leq i\leq n$，则$M_{ij}$=1，否则，$M_{ij}$=0；对于后 2^m列，当项集$p_k\subseteq IS(T_i)$时，其中，$0\leq k\leq 2^m$-1，则$M_{i(k+m+1)}$=1，否则，$M_{i(k+m+1)}$=0。由于每个事务中都包含空项集，因此矩阵的第m+1 列总是为 1，为节省时间，不累加这列 1 的个数。

算法 基于矩阵的频繁项集挖掘(FIMM)算法

输入 事务数据库 D，最小支持数 minss

输出 所有频繁项集

Begin

扫描数据库D，把所有事务置为矩阵的行，把所有的项置为矩阵的前m列，把所有的项集置为矩阵的后 2^m列。

```
for(i=1;i<=n;i++)
  {for(j=1;j<=m;j++)      /* 为矩阵前 m 列赋值 */
    if(项I_j in 事务T_i) then
        M_ij=1;
    else
        M_ij=0;
   for(k=0;k<=2^m-1; k++)
    if(项集p_k in事务T_i 的项集集合IS(T_i)) then
        M_i(k+m+1)=1;
   else
       M_i(k+m+1)=0;
  }
for(i=1;i<=n;i++)
  {for(j=m+2;j<=m+2^m-1;j++)
    if M_ij=1 then    /*累加矩阵后 2^m-1 列中每列 1 的个数，结果放于第n+1 行相应的列。*/
        count[n+1][j]++；
  }
for(k=m+2;k<=m+2^m-1;k++)
if count[n+1][k]>=minss then
/*比较这些计数值与最小支持数 minss 的大小，大于等于 minss 的元素就是求得的频繁项集。*/
      p_k-m是频繁项集；
End
```

2.3 算法分析

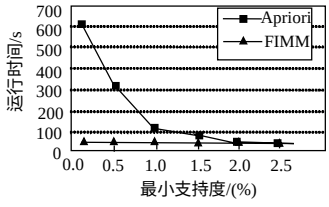
FIMM 算法是根据矩阵理论提出的，应用集合论，令矩阵的行存储所有事务，列存储所有项和项集，所以，该算法在理论上是正确的。而且由于矩阵存储的主要数据类型是布尔值，可以按位方式进行存储，因此在计算上是可行的。

挖掘频繁项集算法的效率取决于扫描事务数据库的次数和产生候选集的迭代次数，而FIMM算法只需扫描事务数据库一次，整个过程分为 2 个阶段：(1)扫描事务数据库把事务数据库转化为矩阵并为矩阵赋值阶段，为矩阵赋值的计算次数为$n\times(m+2^m)$。(2)累计矩阵后 2^m-1 列的值和求频繁项集阶段，累加操作的次数为$n\times(2^m-2)$，求频繁项集的计算次数为

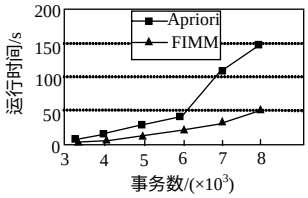
2^m-2，所以，FIMM算法的时间复杂度为$O(n\times 2^m)$。

3 实验结果

为验证 FIMM 算法的有效性，本文采用标准数据 Mushroom(117 个不同项，8 124 个事务，平均每个事务均有 23 项，文件大小 558 KB)进行频繁项集挖掘实验，与 Apriori 算法比较的实验结果如图 1 所示。



(a)随最小支持度变化的运行时间



(b)随事务数量变化的运行时间

图 1 随支持度和事务数量变化的相应时间

图 1(a)为 2 种算法对给定不同最小支持度的比较结果，它显示 FIMM 算法不受最小支持度大小的影响，而 Apriori 算法当支持度减小时，耗费的时间会增大。图 1(b)为 2 种算法在固定最小支持度为 1%而事务数不同情况下的比较结果，它显示当支持度不变而事务数增多时，2 种方法花费时间都会呈上升趋势，但 FIMM 算法上升的趋势非常慢。

4 结束语

本文提出的 FIMM 算法是种频繁项集挖掘算法。它只需扫描事务数据库一次，并把事务数据库转化为矩阵，矩阵的列向量按照一定规则排列，这样在对矩阵赋值时省去了大量工作，节省了时间。在对矩阵挖掘频繁项集时，无需进行连接和剪枝操作，而是根据每列的项集计数过滤掉小于支持度阈值的项集，从而发现所有频繁项集。当支持度阈值发生变化时，也不用重新扫描数据库和重新构造矩阵，就能很容易地挖掘频繁项集，因此，该算法也适合增量挖掘频繁项集。

参考文献

[1] Agrawal R, Imielinski T, Swami A. Mining Association Rules Between Sets of Items in Large Databases[C]//Proc. of ACM-SIGMOD Int'l Conf. on Management of Data. Washington D. C., USA: [s. n.], 1993.

[2] Han Jiawei, Pei Jian, Yin Yiwei. Mining Frequent Patterns Without Candidate Generation[C]//Proc. of the 2000 ACM-SIGMOD Int'l Conf. on Management of Data. Dallas, TX, USA: [s. n.], 2000.

[3] 孟祥萍, 钱 进, 刘大有. 基于数组的关联规则挖掘算法[J]. 计算机工程, 2003, 29(15): 98-99.

[4] 焦学磊, 王新庄. 基于矩阵的频繁项集发现算法[J]. 江汉大学学报: 自然科学版, 2007, 35(1): 43-46.

[5] 王柏盛, 刘寒冰, 靳书和, 等. 基于矩阵的关联规则挖掘算法[J]. 微计算机信息, 2007, 24(5): 143-145.

[6] Wu Fan. A New Approach to Mine Frequent Patterns Using Item-transformation Methods[J]. Information Systems, 2007, 32(7): 1056-1072.

（上接第 83 页）

其中，c_{ij}是一个界于 0 和 1 之间的值(当上式的分母为 0 时，c_{ij}应该取 0)。通过正规化可以防止信誉作弊，即防止一个用户通过给一个很高的评价值来不正当地帮另一个用户提高信誉。c_{ij}仅仅表示一个局部的信任关系，如果c_{ij}的值较大，并不意味着用户j可以信任，它只表示对于用户i来说，用户j是可信任的。

现实中信任关系是传递的，即如果i信任j，并且j信任k，那么i也会信任k。利用这种传递关系，信任管理系统可以为每个用户计算出一个全局的可信任度。令$\boldsymbol{C}$表示矩阵$[c_{ij}]$，向量$\boldsymbol{t}$是$\boldsymbol{C}$的左主特征向量(left principal eigenvector)，则$\boldsymbol{t}i$表示用户i的全局可信任度。

信任管理服务器的任务是接收用户的反馈信息，并接收用户关于全局可信任度的查询。TrustFs 还可以根据应用的需要采取其他的模型和算法来计算可信任度，具体可参考信任管理相关的文献。上文描述的模型具有代表性，并且可以使用非集中式的方式来实现，因此非常适合于分布式文件系统。

4 结束语

TrustFs 采用了信任管理机制来对用户的可信任程度进行评价，从而帮助用户识别潜在的不可信任的文件；TrustFs 使用数字签名这种强安全机制保证文件的发布者不可伪造，这些技术的结合大大增强了分布式文件系统的安全性。TrustFs 以可堆叠文件系统的形式实现，继承了其灵活性与可移植性。

参考文献

[1] Zadok E, Iyer R, Joukov N, et al. On Incremental File System Development[J]. ACM Transactions on Storage(TOS) Magazine, 2006, 2(2): 161-196.

[2] Brondsema D, Schamp A. Konfidi: Trust Networks Using PGP and RDF[C]//Proc. of the WWW'06 Workshop on Models of Trust for the Web. Edinburgh, Scotland, UK: [s. n.], 2006.

[3] Kamvar S D, Schlosser M T, Hector G M. The Eigen Trust Algorithm for Reputation Management in P2P Networks[C]//Proc. of the 12th International World Wide Web Conference. Budapest, Hungary: [s. n.], 2003.

[4] Patil S, Kashyap A, Sivathanu G, et al. I3FS: An In-kernel Integrity Checker and Intrusion Detection File System[C]//Proc. of the 18th USENIX Large Installation System Administration Conference. Atlanta, GA, USA: [s. n.], 2004: 69-79.

[5] Joukov N, Rai A, Zadok E. Increasing Distributed Storage Survivability with a Stackable Raid-like File System[C]//Proc. of the 5th IEEE International Symposium on Cluster Computing and the Grid. Cardiff, UK: [s. n.], 2005: 82-89.