ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2008/02(03)-0330-07
DOI:10.3778/j.issn.1673-9418.2008.03.009

E-mail:fcst@public2.bta.net.cn
http://www.ceaj.org
Tel:+86-10-51616056

数据流中基于矩阵的频繁项集挖掘

王 磊+,黄志球,朱小栋,沈国华,程 亮
WANG Lei+, HUANG Zhiqiu, ZHU Xiaodong, SHEN Guohua, CHENG Liang

南京航空航天大学 信息科学与技术学院,南京 210016
College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
+ Corresponding author: E-mail: ad82ad82@126.com

WANG Lei, HUANG Zhiqiu, ZHU Xiaodong, et al. Mining frequent itemsets over data stream by matrix. Journal of Frontiers of Computer Science and Technology, 2008,2(3):330-336.

Abstract: Mining frequent itemsets is a basic task of the data stream mining. Recently many approximate algorithms can mine frequent itemsets over data stream. However, these algorithms still can not efficiently reduce space and time cost. To improve the efficiency of mining frequent itemsets over data stream, matrix is imported as the synopsis data structure and a new algorithm of mining frequent itemsets is presented. Finally, experiments prove the efficiency of this algorithm.

Key words: data stream; data mining; frequent itemsets; matrix

摘 要:挖掘频繁项集是挖掘数据流的基本任务。许多近似算法能够有效地对数据流进行频繁项挖掘,但不能有效地控制内存资源消耗和挖掘运行时间。为了提高数据流频繁项集挖掘的时空效率,通过引入矩阵作为概要数据结构,提出了一种新的数据流频繁项集挖掘算法。最后通过实验证明了该算法的有效性。

关键词:数据流;数据挖掘;频繁模式;矩阵
文献标识码:A **中图分类号:**TP311

1 引言

数据流是一种特殊的数据类型。它在一个近似无限长的时间范围内,快速产生大量数据,如网页的点击流、传感器网络产生的数据和股票价格波动的数据,都是数据流的典型示例。数据流产生的数据无法全部保存在内存中,而访问存储在次级储存介质中的数据会严重影响算法的性能,因此基于数据流的数据挖掘算法只能按数据产生的次序访问一遍数据,且通

Received 2007-04, Accepted 2007-11.

常产生近似结果。

作为数据挖掘的一个重要课题,频繁项集挖掘在过去 10 年中得到了广泛的研究。但因为现存的频繁项集挖掘算法大多需要多次扫描数据集,无法直接将它们应用到数据流挖掘领域。基于数据流的频繁项集挖掘仍然是一个极具挑战性的研究方向。例如:Manku 等[1]提出了一个建立在频繁项挖掘算法(Lossy Counting)基础上的频繁项集挖掘算法(记为Manku),它在扫描一次数据的基础上得到基于整个数据流的频繁项集。

为了能够应对数据过期的情况,文献[2]将采样方法扩展到滑动窗口中[3],以处理数据的动态添加和删除。

针对空间效率低的问题,文献[4-6]利用前缀树的方法设计了有效的数据结构来存储频繁项集,且文献[5,6]利用项集频率估计方法,即一个项集的频率可通过其子项集的频率来估计,从而尽早过滤掉低频信息,降低了存储空间。但是,这些树型结构有时会随着项的增多而增加, 以致无法完全存储于内存中,这又将导致效率的下降。为此,Lee 等人[7]设计了能够根据内存限制进行合并的 CP-tree, 这种树型结构可根据内存的限制自动调整大小以满足实际要求。

此外,Jin 和 Agrawal[8]基于文献[9]所提出的方法设计了一个有效的数据流频繁模式挖掘算法,以适应更小型或移动设备的需要。

本文提出 MISM(Mining Frequent Itemsets over Data Stream by Matrix)算法,利用矩阵作为概要数据结构来挖掘基于整个数据流的频繁项集。

2　数据流频繁项集挖掘问题定义

数据流频繁项集挖掘可以描述如下:$I=\{x_1,x_2,\cdots,x_n\}$是给定的全项集。一个项集是全项集的一个子集。事务流 D 是一个依次到达的事务序列$(t_1,t_2,\cdots,t_N)$,其中每个事务 t_i 也是一个项集,N 是一个事先不能确定的大数。

定义 1 给定事务数据库 D 和最小支持数阈值 s,对于项集 $X\subseteq I$,若 $sup(X)\geqslant s$,则称 X 为 D 中的频繁项集。

就求解频繁项集而言,数据流频繁项集挖掘的一次扫描数据集的方法与传统多次扫描数据集的方法有着本质不同,数据流一次扫描数据集的挖掘方法以牺牲挖掘结果的精确性为代价,通常只能求解出近似解。估计的近似程度用两个参数:$\varepsilon\in(0,1)$控制准确的程度,$\delta\in(0,1)$控制近似的可靠程度。这种 ε-δ 的近似模式是经常使用的。

3　MISM 相关技术与概要数据结构

3.1　Manku 算法

MISM 算法采用 Manku 算法中对项集处理的近似求解的原理,现给出 Manku 算法描述:现有数据流的长度记为 N,用户指定两个参数,支持度 s 和误差 ε,挖掘数据流频繁项集的难点在于如何处理可变的事务的长度和避免明显地列举出所有事务的子集。这里的数据结构 D 由 entry(set,f,Δ)的集合所组成,其中 set 是项集的子集,f 代表项集 e 的估计频繁度,Δ 代表 f 中最大可能的出错数。算法的最初,D 为空。把读入的事务流存储在等宽的桶(bucket)中,每个 bucket 存在 $w=\lceil 1/\varepsilon\rceil$个事务。每个 bucket 有其从 1 开始标志的 bucket id。现在时刻的 bucket id 记为 $b_{current}$,算法中并不按一个一个的事务来处理数据流,而是在主存中填入尽可能多的事务,然后以批量的形式来处理这些事务。另外用 β 来表示主存中 bucket 的数量。

Manku 算法的更新策略分为两个阶段:

(1)更新阶段 UPDATE_SET:对 D 中的每个 entry (set,f,Δ),更新现有主存中所有项集的频繁度 f。如果被更新的 entry 满足 $f+\Delta\leqslant b_{current}$,则从 D 中删除此 entry。

(2)新增阶段(NEW_SET):如果某项集的频繁度 $f\geqslant\beta$,且该项集在 D 中不存在,则创建一个新的 entry $(set,f,b_{current}-\beta)$,并加入 D 中。

由上述过程可以发现,当某个项集的真实频繁度 $f\geqslant\varepsilon N$,该项集的 entry 必定在 D 中。另外,一个 D 中的 entry(set,f,Δ)的真实频繁度 f_{set} 也满足 $f\leqslant f_{set}\leqslant f+\Delta$。当一个用户以支持度 s 请求频繁项集时,算法就输出 D 中所有满足 $f\geqslant(s-\varepsilon)N$ 的 entry 中项集。

3.2　上三角项集矩阵的定义与构造

(1)上三角项集矩阵中行 i、列 j 为数据流中数据

项的集合$\{I_1,I_2,\cdots,I_n\}$。

(2)上三角项集矩阵中的元素(i,j)的值为一个entry(e,f,Δ),其中e代表I_iI_j数据项,f代表数据项I_iI_j估计的频繁度,Δ代表f中最大可能的出错数。同理$<I_i,I_j>$表示一个2项集,$\cdots,<I_1,\cdots,I_n>$表示一个n项集。

3.3 上三角项集矩阵的构造

首先扫描数据流,对数据流中的每个二元组$T=\{Tid,Itemsets\}$(Tid是编号,$Itemsets$是项集),求出T中每个项和二项集,并把每个项和项集构成一个entry保存到概要数据结构矩阵中的相应位置。

图1给出一个由当前包含9个事务项的数据流(表1所示)而生成的上三角项集矩阵,这里图1中矩阵各个元素的值仅仅标为项或二项集的频繁数,实际存储的是entry。

	I_1	I_2	I_3	I_4	I_5
I_1	6	4	4	1	2
I_2		7	4	2	2
I_3			6	0	1
I_4				2	0
I_5					2

Fig.1 Upper triangular matrix

图1 上三角矩阵

Table 1 Data stream of transaction

表1 事务数据流

TID	项ID列表
T100	I_1,I_2,I_5
T200	I_2,I_4
T300	I_2,I_3
T400	I_1,I_2,I_4
T500	I_1,I_3
T600	I_2,I_3
T700	I_1,I_3
T800	I_1,I_2,I_3,I_5
T900	I_1,I_2,I_3
…	…

4 MISM原理与算法

4.1 MISM基本思想

本文提出的数据流频繁项集挖掘算法利用深度优先(Depth First Search)的搜索策略进行遍历上三角项集矩阵。具体过程如下:

(1)首先遍历三角项集矩阵对角线上的矩阵元素,元素entry.f的值不小于最小支持度$s-\varepsilon$,则该元素为频繁项。

(2)从上三角项集矩阵元素(1,2)作为搜索起点,当上三角项集矩阵中的任意元素$(i,j)(i\neq j)$的值不小于最小支持度$s-\varepsilon$,则$<I_i,I_j>$就是满足最小支持度为s的频繁2-项集。

(3)当矩阵中有元素$(i,j)\geqslant s-\varepsilon,(j,k)\geqslant s-\varepsilon$,且$(k,i)\geqslant s-\varepsilon$,则这3个结点$<I_i,I_j,I_k>$就是满足最小支持度为$s$的频繁3-项集的候选项集,依次类推,当矩阵中存在这样的元素链,$(i,j)\geqslant s-\varepsilon,(j,k)\geqslant s-\varepsilon,\cdots,(m,n)\geqslant s-\varepsilon,(n,i)\geqslant s-\varepsilon$,且这$n$个元素链的所有子链都能构成频繁项集的候选项集,则这n个结点$<I_i,I_j,\cdots,I_n>$就是满足最小支持度为s的频繁n-项集的候选项集。可以看出如果暂时不考虑ε,且最小支持度为2的话,可推出图1中包含的频繁1-项集的候选项集有:$L_1=\{I_1,I_2,I_3,I_4,I_5\}$,频繁2-项集的候选项集有:$L_2=\{\{I_1,I_2\},\{I_1,I_3\},\{I_1,I_5\},\{I_2,I_3\},\{I_2,I_4\},\{I_2,I_5\}\}$。频繁3-项集的候选项集有:$L_3=\{\{I_1,I_2,I_3\},\{I_1,I_2,I_5\}\}$。

(4)因为MISM在上三角矩阵的构造和维护中,按照Manku算法的误差控制方法来修剪元素,所以对于由(3)产生的候选项只需要再修剪掉数据流中实际不存在的项集,便可以得到在误差控制参数范围内的频繁项集,例如存在2项频繁项集$\{I_1,I_2\},\{I_2,I_3\},\{I_3,I_1\}$,于是按步骤(3)得到候选项集$\{I_1,I_2,I_3\}$,但是事实上项集$\{I_1,I_2,I_3\}$不存在于数据流中,因此需要去掉该候选项集,这里用一个链表来存储最大项集,来判断候选项集是否存在于事务流中。

4.2　MISM 方法描述

(1)把读入的事务流存储在等宽的桶中,每个桶存放 $w=\lceil 1/\varepsilon \rceil$个事务。每个桶有唯一的标志,初始值为 1,$N$ 为当前时间事务的数量,当前值 $b_{current}$为$\lceil N/w \rceil$。

(2)求出数据流的每个事务的所有项和 2-项集,项和 2-项集的 entry 定义为$(item,f,\Delta)$,并用项来构建矩阵,把这些项和 2-项集加入或更新矩阵,此外如果该事务的项集不是任何一个链表元素的子集,则把该事务的项集加入链表。

(3)当一个新的项或 2-项集 e 到达时,检索事务链表组中是否存在 e,如果存在 e,则 e 的频率加 1。

(4)当一个新的项或 2-项集 e 到达时,检索事务链表组中是否存在 e,如果不存在 e,则将$(e,f,b_{current}-1)$加入矩阵的对应位置中。

(5)删除操作在 N 为 w 的整数倍时发生,对任何矩阵中项或二项集(e,f,Δ),如果$f+\Delta \leqslant b_{current}$,则删除与之相关的所有矩阵元素。

(6)当收到对挖掘结果的请求后,输出$f \geqslant (s-\varepsilon)N$的那些项,$s$、$\varepsilon$ 分别为支持度和误差。输出方法按 3.1 节的深度优先的方法遍历矩阵,如果满足条件,且该项集在链表中存在或者为最大项集链表中某一项集的子集,则输出该项集。

4.3　相关定理及证明

定理 1　无论什么时候当删除操作部分发生时,$b_{current} \leqslant \varepsilon N$。

证明　因为$\lceil 1/\varepsilon \rceil * b_{current} \leqslant N$,可得结论。　□

定理 2　无论什么时候当一个 entry(e,f,Δ)被删除,$f_e \leqslant b_{current}$。

证明　这里通过归纳的方法给予证明。当 $b_{current}=1$。一个 entry(e,f,Δ)只有当其 f=1 时才被删除,此时 f 也就是说它的真实频率 f_e,这样 $f_e \leqslant b_{current}$。假定当 $b_{current}>1$ 时,一个 entry(e,f,Δ)被删除。可知当 $\Delta+1$ 个 bucket 处理时,这个 entry 被插入。一个 entry 可能被删除,最迟当 $bucket\Delta$ 填满后。通过归纳,e 真实的频繁度计数 f_e 不可能超过 Δ。进一步,当 e 被插入时起,f 是 e 的真实频繁度,由此得出 f_e 从 $bucket_1$ 到 $b_{current}$最多为 $f+\Delta$。与删除规则$f+\Delta \leqslant b_{current}$结合,得出结论 $f_e \leqslant b_{current}$。　□

定理 3　如果项集 e 不在概要数据结构 D 中出现,则 $f_e \leqslant \varepsilon N$。

证明　无论什么时候 e 被删除时,如果该定理成立,也就是说当 $N \equiv 0 \mod w$,这对于其他的值 N 也都成立。从定理 1 和定理 2,可以推出无论什么时候 e 被删除时,$f_e \leqslant \varepsilon N$。　□

定理 4　如果$(e,f,\Delta) \in D$,那么$f \leqslant f_e \leqslant f+\varepsilon N$。

证明　如果 $\Delta=0$,则 $f=f_e$。另外,有时候在前 Δ 个 bucket,e 有可能被删除,从定理 2 可以推断出当最后一个这样的删除发生时,e 的精确频率最大为 Δ。因此,$f_e \leqslant f+\Delta$。既然 $\Delta \leqslant b_{current}-1 \leqslant \varepsilon N$,可以得出结论$f \leqslant f_e \leqslant f+\varepsilon N$。　□

4.4　MISM 算法描述

下面给出内存中概要数据结构矩阵维护的伪代码(e_{sub}指 e 中的二项集,$\boldsymbol{D}$ 为矩阵,而 $list$ 为保存最大项集的链表):

```
1. D←Ø;N←0
2. w←⌈1/ε⌉;b_current←1
3. e←next itemset;N←N+1
4.    update(e,f,Δ)
5. if N mod w=0 do
6.    prune(D,b_current);
7.    b_current←b_current+1
8. endif
9. Goto 3;
```

```
1. function prune(D,b_current)
2. for each entry(e,f,Δ) in D do
3.    if f+Δ≤b_current do
4.       remove the entry from D
5.    endif
```

```
1. function update(e,f,Δ)
```

```
2.  if e is a subitem of list element
3.    add e to list
4.  endif
5.  for each e_sub in e
6.    if(e_sub exists in D)
7.      e_sub.f ←e_sub.f+1 in D
8.    else
9.      e.f ←1 and e.Δ←b_current-β
10.     D←e
11.   endif
```

下面给出当用户请求输出频繁项集时的伪代码(*L* 保存频繁项集,*s* 为最小支持度,*numlist* 保存项在矩阵中的位置编号):

```
1. for i←1 to n
2. if ((I_i,I_i)≥(s-ε)N)
3.   L←I_i
4.   call DFS(i,L,i)
5. endif

6. function DFS(i,L,k)
7. add k to numlist
8. for j←i+1 to n
9.   if ((I_i,I_j)≥(s-ε)N and (I_k,I_j)≥(s-ε)N)
10.    add j to numlist
11.    tmp←I_num1∪I_num2…in numlist
12.    if (tmp exsiting in list or tmp is
13.        a subitem of list element)
14.      L←tmp
15.    endif
16.    DFS(j,L,k)
17.  endif
```

5 实验与算法分析

5.1 空间复杂性分析

Manku 的算法中,随最小支持度减小和频繁项集长度增加,它维护树型结构所需的空间复杂度大大增加。MISM 算法在空间上的代价主要是存储上三角项集矩阵,而矩阵的大小主要由数据流中数据项的个数 *n* 来决定。由于矩阵只需要存储约 $n^2/2$ 个二项集出现的次数信息,*n* 代表唯一数据项的个数,所以 MISM 所需空间代价比 Manku 算法低。

5.2 时间复杂性分析

Manku 算法中,算法在时间上的代价主要在前缀树的维护中对前缀树的遍历访问操作,随着前缀树的不断增大,这个开销所造成的时间复杂度是比较大的。而 MISM 算法在时间上的代价主要取决于搜索矩阵时 DFS 的执行次数。对矩阵进行搜索时,DFS 的执行次数则是由矩阵中结点的个数以及频繁项集的长度决定,如果唯一的数据项的个数确定的话,所需 DFS 所需时间也可以确定,因此时间复杂度要小于 Manku 算法。

5.3 实验结果与比较

这里使用一台 Pentium 4 1.8 GHz/768 M 微机(OS 为 Windows XP)作为服务器端,一台 Pentium 4 3.0 GHz/768 M 微机(OS 为 Windows XP)作为客户端,服务器端是自行开发的一个数据流的产生器,客户端对服务器产生的数据流进行挖掘。本算法的实现语言是 Java 5.0,开发工具是 Eclipse,对 MISM 算法与 Manku 算法随事务数变化和最小支持度变化在挖掘时间和内存消耗上进行了对比实验,另外平均每事务包含项为 4 个,数据流中唯一项有 10 个,误差参数取 0.01。图 3、图 5 给出了当最小支持度为 0.1 时,MISM 算法与 Manku 算法随事务数变化的挖掘时间和内存消耗结果比较,图 2、图 4 给出了在挖掘 100 000 条事务的前提下,MISM 算法与 Manku 算法随最小支持度变化的挖掘时间和内存消耗结果比较。

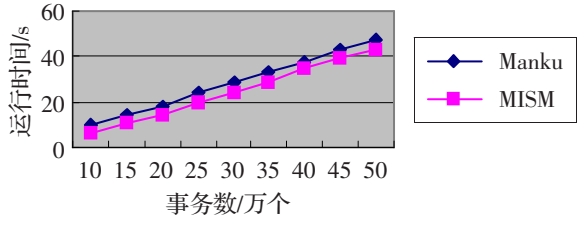

Fig.2 The comparison of time cost between Manku and MISM when transaction number changes

图 2 Manku 与 MISM 随事务数变化运行时间比较

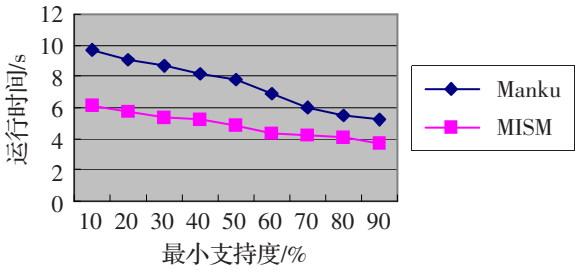

Fig.3　The comparison of time cost between Manku and MISM when minimum support changes

图 3　Manku 与 MISM 随最小支持度变化的运行时间比较

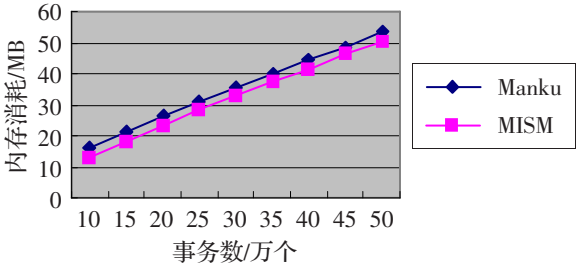

Fig.4　The comparison of memory cost between Manku and MISM when transaction number changes

图 4　Manku 与 MISM 随事务数变化的内存消耗比较

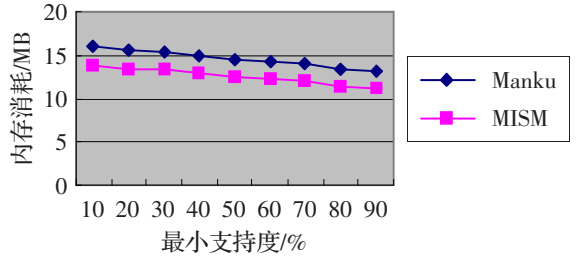

Fig.5　The comparison of memory cost between Manku and MISM when minimum support changes

图 5　Manku 与 MISM 随最小支持度变化的内存消耗比较

6　结束语

本文提出了一种基于矩阵的数据流频繁项集挖掘算法 MISM,它将数据流中有关项集的信息保存在一个上三角矩阵中,通过对上三角项集矩阵进行搜索来发现其中的频繁项集,并且当事务数据流和最小支持度发生变化时，该算法与 Manku 算法的比较中体现了更好的时间复杂度和空间复杂度。

References:

[1] Manku G S, Motwani R. Approximate frequency counts over data streams[C]//Proceedings of the 28th International Conference on Very Large Databases, 2002:346-357.

[2] Arasu A, Manku G S. Approximate counts and quantiles over sliding windows[C]//Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Paris, France: ACM Press, 2004:286-296.

[3] Datar M, Gionis A, Indyk P, et al. Maintaining stream statistics over sliding windows[C]//Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms. San Francisco, USA: ACM Press, 2002:635-644.

[4] Li H, Lee S, Shan M. An efficient algorithm for mining frequent itemsets over the entire history of data streams[C]//Proceedings of the First International Workshop on Knowledge Discovery in Data Streams, held in conjunction with the 15th European Conference on Machine Learning (ECML 2004) and the 8th European Conference on the Principles and Practice of Knowledge Discovery in Databases (PKDD 2004), Pisa, Italy, 2004.

[5] Chang J, Lee W S. Finding recent frequent itemsets adaptively over online data streams[C]//Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington, USA: ACM Press, 2003:487-492.

[6] Chang J, Lee W S. EstWin: adaptively monitoring the recent change of frequent itemsets over online data streams[C]//Proceedings of the 12th International Conference on Information and Knowledge Management. New Orleans, USA: ACM Press, 2003:536-539.

[7] Lee D, Lee W. Finding maximal frequent itemsets over online data streams adaptively[C]//Proceedings of the Fifth IEEE International Conference on Data Mining. Houston,

USA: IEEE Press, 2005:266-273.

[8] Jin R, Agrawal G. An algorithm for in-core frequent itemset mining on streaming data[C]//Proceedings of the Fifth IEEE International Conference on Data Mining. Houston, USA: IEEE Press, 2005:210-217.

[9] Karp R M, Shenker S, Papadimitriou C H. A simple algorithm for finding frequent elements in streams and bags[J]. ACM Transactions on Database Systems, 2003,28(1):51-55.

WANG Lei was born in 1982. He is a graduate student. His research interest includes data mining.
王磊(1982-),男,浙江嘉兴人,硕士研究生,主要研究领域为数据挖掘。

HUANG Zhiqiu was born in 1965. He is a professor. His research interests include data warehouse, data mining, software engineering and database.
黄志球(1965-),男,江苏南京人,教授,博士生导师,主要研究领域为软件工程与数据库、数据仓库、数据挖掘。

ZHU Xiaodong was born in 1981. He is a Ph.D. candidate. His research interest includes data mining.
朱小栋(1981-),男,安徽太湖人,博士研究生,主要研究领域为数据挖掘。

SHEN Guohua was born in 1976. He is a Ph.D. candidate. His research interests include data warehouse, software engineering and database.
沈国华(1976-),男,江苏丹阳人,博士研究生,主要研究方向为软件工程与数据库、数据仓库。

CHENG Liang was born in 1982. He is a graduate student. His research interest includes data mining.
程亮(1982-),男,江苏宜兴人,硕士研究生,主要研究领域为数据挖掘。