高技术通讯 2010 年 第 20 卷 第 3 期:246 ~ 252

doi:10.3772/j.issn.1002-0470.2010.03.005

一种有效的数据流最大频繁模式挖掘算法①

毛伊敏②*** 杨路明* 李 宏* 陈志刚* 刘立新*

(*中南大学信息科学与工程学院 长沙 410083)

(**江西理工大学应用科学学院 赣州 341000)

摘 要 针对频繁项集挖掘存在数据和模式冗余的问题,对数据流最大频繁项集挖掘算法进行了研究。针对目前典型的数据流最大频繁模式挖掘算法 DSM-MFI 存在消耗大量存储空间及执行效率低等问题,提出了一种挖掘数据流界标窗口内最大频繁项集的算法 MMFI-DS,该算法首先采用 SEFI-tree 存储包含在不断增长的数据流中相关最大频繁项集的重要信息,同时删除 SEFI-tree 中大量不频繁项目,然后使用自顶向下和自底向上双向搜索策略挖掘界标窗口中一系列的最大频繁项集。理论分析与实验表明,该算法比 DSM-MFI 算法具有更高的效率,并能节省存储空间。

关键词 数据挖掘,数据流,界标窗口,频繁项集,最大频繁项集

0 引 言

数据挖掘研究的一个重要方面是关联规则,生成频繁项集是关联规则挖掘的关键技术和步骤。由于最大频繁项集隐含了所有频繁项集,故发现频繁项集的问题则转化成为挖掘最大频繁项集。此外,某些数据挖掘应用仅需挖掘最大频繁项集,而不必挖掘所有频繁项集,因而挖掘最大频繁项集对数据挖掘具有重要意义,许多最大频繁项集挖掘算法被提出[1-5]。

随着计算机网络技术的发展,一种新的数据模型——数据流得到了广泛的关注,数据流已被广泛应用于商务过程管理中的性能监测,网络流量管理中的异常检测及报警,零售业中的事务处理,银行业中的自动柜员机管理,网站访问记录的统计和分析,以及传感器网络数据的管理等应用中。由于数据流具有无限性和流动性,这使得在数据流环境下的挖掘比在静态环境下的数据挖掘要困难,在静态环境下进行多遍数据扫描的技术不能适用于数据流环境,数据流环境下的挖掘算法必须能在有限的内存空间和限定的时间内快速形成模式的归纳,对时间和空间效率的要求要比静态算法高。

数据流的处理模型分为界标窗口处理模型、衰减窗口处理模型和滑动窗口处理模型,相应地,数据流的频繁模式挖掘的研究分为三类。在界标窗口处理模型方面,2002 年,Manku 和 Motwani 提出了 Lossy Counting 算法[6],它是基于 Apriori 算法[7]思想,利用近似归纳技术实现数据一次扫描,实现频繁项目集的挖掘;2006 年,Yu 等提出了 false-negative 算法[8],它基于切尔诺夫约束,从高速的事务型的数据流中挖掘频繁项目集,使用运行错误参数剪掉不频繁项集和可靠性参数控制内存。在滑动窗口处理模型方面,2004 年,Chi 等提出了 Moment[9]算法,它采用前缀项目树存储、维护结点信息,并通过结点类型转换来挖掘闭频繁项集;2008 年,Li[10]等人提出了 MFI-TransSW 算法,此算法用二进制位序列表示每个事务的项目,当窗口滑动时,采用位移动技术移去过时的事务,当用户发出请求时,一系列频繁项集从滑动窗口中被挖掘出来。在衰减窗口处理模型方面,2003 年,Chang 等提出了 estDec 算法[11],它通过定义一个称为衰减因子的参数,使得较早到达数据流的事务的影响逐渐减弱;2003 年,Giannella 等提出了一种传统的 FP-Tree[12]改造的处理数据流的算法 FP-stream[13],它利用不同时间粒度实现不同时间段的频繁项集的生成。

目前,在数据流环境下挖掘最大频繁模式的算法还很少。Li 等人提出的 DSM-MFI[14](data stream

① 国家自然科学基金(60573127)资助项目。

② 女,1970 年生,博士,副教授;研究方向:数据挖掘;联系人,E-mail:mymlyc@163.com

(收稿日期:2008-11-28)

mining for maximal frequent itemsets)算法采用概要频繁项目森林(summary frequent itemset forest, SFI-forest)来存储当前流进的数据流信息,当每个基本窗口的数据流过时,从概要频繁项目森林中剪掉大量不频繁的项目集,内存中存储有用的频繁项目集,因此内存空间的效率得到了很大的提高。但是 DSM-MFI 算法用投影的方式把每个事务的投影子集存储到概要频繁项目森林结构中,这样概要频繁项目森林树中存储了大量的事务投影子集,浪费了大量的存储空间,同时若这些事务投影子集是非频繁项集,还要把它们从概要频繁项目森林树中删除,消耗了大量的时间,而且此算法只采用自顶向下的搜索方法发现最大频繁项目集,这样需要计算大量的项目集支持度,而项目集支持度的计算是发现频繁项目集最耗时的工作。针对这种情况,本文提出了一个 DSM-MFI 改进算法——MMFI-DS(mining maximal frequent itemsets over data stream),该算法使用类似 FP-tree 树的概要数据结构存储数据流信息,可节省大量的存储空间和时间开销,而且采用自顶向下和自底向上双向搜索策略,可尽早剪掉较短非频繁项集的超集和较长最大频繁项集的子集,减少项目集支持度计算,降低算法开销。

1 问题定义和描述

定义 1 一个数据流是一个连续的、无穷的基本窗口序列,$DS=[w_1,w_2,\cdots,w_n]$,w_i,$\forall i=1,2,\ldots,N$,i 是每一个基本窗口的标识,N 是最后一个基本窗口的标识。每个基本窗口含有固定数目的事务,$\langle T_1,T_2,\cdots,T_k,\cdots\rangle$,$T_k=\{i_1,i_2,\cdots,i_m,\cdots\}$,$T_k(k=1,2,\cdots)$称为事务,$i_m(m=1,2,\cdots,p)$称为项目。

定义 2 $x.t\mathrm{sup}$ 是项目 x 的真实支持度,其值为项目 x 在事务中所包含的数目,$x.e\mathrm{sup}$ 是项目 x 的估计支持度,其值为项目 x 在概要数据结构中所包含的数目,并且 $1\leqslant x.e\mathrm{sup}\leqslant x.t\mathrm{sup}$。

定义 3 设给定的支持度 s 和允许偏差 $\varepsilon(\varepsilon<s)$,$|w|$表示基本窗口的长度,$CL$ 为 $|w_1|+|w_2|+\cdots+|w_N|$,N 为当前窗口标识,$x.CL$ 为 $|w_j|+|w_{j+1}|+\cdots+|w_N|$,其中 w_j 为项目 x 在数据流中出现的第一个窗口。项目 x 被分为 3 种类型,如果 $x.e\mathrm{sup}\geqslant s\cdot x.CL$,则称频繁项集;如果 $s\cdot x.CL>x.e\mathrm{sup}\geqslant\varepsilon\cdot x.CL$,则称潜在频繁项集;如果 $x.e\mathrm{sup}<\varepsilon\cdot x.CL$,则称不频繁项集。

定义 4 设给定的支持度 s 和允许偏差 $\varepsilon(\varepsilon<s)$,$N$ 是当前数据流已经到来的数据项个数,在任意时刻,用户发出查询,算法满足:(1)所有真实频率超过 sN 的数据项均被输出;(2)所有真实频率低于 $(s-\varepsilon)N$ 的数据项均不输出;(3)算法真实频率值与算法估计频率值的误差小于 εN,则称算法满足 ε-近似要求。

定义 5 若频繁项目集 x 的所有超集都是非频繁项目集,则称 x 为最大频繁项目集;将所有最大频繁项目集组成的集合称为最大频繁集(maximum frequent sets, MFS)。

我们关注在界标窗口模型中输出所有的最大频繁集。在界标窗口模型中,知识的发现是在一个确定的窗口与当前窗口之间进行,我们的目标是设计一个算法,该算法可以在允许偏差 ε 范围内,在尽可能小的空间复杂性和时间复杂性的基础上,输出界标窗口中所有最大频繁项集。

2 MMFI-DS 算法描述及分析

为了有效地计算界标窗口中的最大频繁项集,我们提出了一种 MMFI-DS 算法:从内存中读取每一个窗口的事务,并把它们存储在一个新的概要数据结构——概要扩展频繁项目树(summary extended frequent itemset tree, SEFI-tree)中;删除 SEFI-tree 中不频繁项;利用 SEFI-tree 有效地挖掘界标窗口中所有最大频繁项集。

2.1 SEFI-tree 的结构

DSM-MFI 算法根据 Apriori 思想,对每个事务做投影,投影规则是[13]:设一个事务 $T=(x_1,x_2,\cdots,x_m)$,把它转换成 m 个子集,即$(x_1,x_2,\cdots x_m)$,$(x_2,x_3,\cdots,x_m)$,$\cdots$,(x_m),并把这些子集存储到 SFI-forest 中,由于 SFI-forest 存储了每个事务的投影子集,它重复存储了大量的不频繁项,读完一个基本窗口事务做删除操作时,需遍历整个 SFI-forest 才能把重复的大量不频繁项从 SFI-forest 中删除,这样浪费了大量的存储空间并会消耗大量的时间。因此,我们利用 FP-tree 的优点,根据数据流具有快速和持续的特点,要求算法只能单遍扫描数据库的特性,设计了类似 FP-tree 的概要数据结构 SEFI-tree,SEFI-tree 存储数据流中基本窗口事务的每个项,当一个基本窗口的所有事务读完时,从 SEFI-tree 中删除不频繁的项,这样 SEFI-tree 存储了从开始窗口到当前窗口所有 $x.e\mathrm{sup}>\varepsilon\cdot x.CL$ 的项,为了有效地挖掘最大频

繁项目集，我们采用 DSM-MFI 算法的思想，使用 OFI-list[13](opposite frequent item list)存储每个事务的投影子集。SEFI-tree 的结构特点如下：

(1) 它由一个 EIS-tree(extended item suffix tree)以及 FI-list[13](frequent item list)组成；

(2) EIS-tree 的每一个结点由 5 个域组成：itemname，node-count，window-id，brother 和 link。itemname 代表项目名；node-count 记录能到达该结点的路径所表示的事务的数目；window-id 是目前界标窗口的标识；brother 指向 EIS-tree 中具有相同 itemname 值的结点，当此结点不存在时，brother 为 null；link 为指向 EIS-tree 中具有相同 itemname 值的下一个结点，当下一个结点不存在时，next 为 null；

(3) FI-list 的每一表项包含 4 个域：itemname，item-count，window-id，和 node-link，其中 item-count 为 itemname 对应项目的频度；node-link 为指向 EIS-tree 中具有相同的 itemname 值的首结点的指针；

(4) OFI-list 的每一表项包含 3 个域：itemname，item-count 和 window-id，表示为 x_i.OFI-list。

2.2 SEFI-tree 的构造和维护

当每一个数据流到达时，把每个项插入到 SEFI-tree 中，再把每个事务投影子集的项插入到 OFI-list 中，读完一个基本窗口的数据，先删除 FI-list 指向 EIS-tree 具有相同的 itemname 所有不频繁的结点，接着删除 FI-list 中那个不频繁的结点，最后删除那个项的 OFI-list 表。

算法1：BuildMaintainSFI-tree. 构造和维护 SEFI-tree

输入： 数据流 $DS=[w_1,w_2,\cdots w_n]$，支持度 s，用户允许的最大误差率 $\varepsilon\in(0,s)$；

输出： 目前所产生的 SEFI-tree；

for each 基本窗口 w_j，$j=1,2,\cdots,N$，do
  for each 事务 $T=(x_1,x_2,\cdots,x_m)\in W_j$ 中的项 $x_i\in T$ do
    if $x_i\notin$ FI-list then
      构造一新实体(x_i,1,j,node-link)并插入到 FI-list 中；
    else
      $x_i.esup=x_i.esup+1$；
    endif
  if EIS-tree 具有一个项目名为 y 而且 y.itemname = x_i.itemname
    $y.esup=y.esup+1$；
  else
    构造一新实体(x_i,1,j,brother,link)并插入 EIS-tree 中；
  endif
  call Transaction _ Projection(T,j)；
endfor
call SEFI-tree _ pruning(SEFI-tree,ε,N)；
endfor

Subroutine Transaction _ Projection

输入： 一个事务 $T=(x_1,x_2,\cdots x_m)$ 和目前窗口标识 j；

输出： x_i.OFI-list；

for each 项目 x_i，$i=1,2,\cdots m$，do
  OFI-list _ maintenance($[x_i|X]$,x_i.OFI-list,j)；
/* $X=x_1,x_2,\cdots,x_m$，$[x_i|X]$是前缀项目 x_i 的前缀事务 */
endfor

Subroutine OFI-list _ maintenance

输入： 一个项目前缀事务 $(x_{i,},x_{i+1},\cdots x_m)$ 和目前窗口标记 j；

输出： 一个修改过的 x_i.OFI-list，$i=1,2,\cdots m$；

for each 项目 x_l，do /* $l=i+1$，$i=2,\cdots m$ */
  if $x_l\notin x_i$.OFI-list then
    构造一新实体 (x_l,1,j)并插入到 x_i.OFI-list 中；
  else
    $x_l.esup=x_l.esup+1$；
  endif
endfor

Subroutine SEFI-tree _ pruning

输入： 一个 EIS-tree，用户允许的最大误差率 ε 和当前窗口标识 j；

输出： 一个包含所有频繁项集和潜在频繁项集的 EIS-tree；

for each 项目 $x_i(i=1,2,\cdots d)\in$ FI-list，$d=|$FI-list$|$ do
  if $x_i.esup<\varepsilon\cdot x_j.CL$ then
    通过遍历，从 EIS-tree 中删除 itemname = x_i 的所有结点；
    从 FI-list 表中删除实体 x_i；
    删除 x_i.OFI-list；
    如果 x_i 在 x_j.OFI-list($j=1,2,\cdots d;j\neq i$)中存在，从 X_j.OFI-list 中删除 x_i；
  endif

endfor

2.3 最大频繁项集的挖掘算法

若 FI-list 中包含目前产生的所有潜在频繁项及频繁项,根据 Apriori 原理,我们设计 tobo-botoMFI (top-bottom and bottom-top selection of maximal frequent)算法,从 SEFI-tree 中有效地挖掘所有最大频繁项集。

假设 FI-list 中有 k 个 1 项集$(e_1, e_2, \cdots, e_k)$,每个项 $e_i(i=1,2,\cdots,k)$都有一个 e_i.OFI-list,其包含 1 项集的数目为$|e_i.\text{OFI-list}|$,e_i.OFI-list 中的每个项表示为 $e_i.o_1, e_i.o_2, \cdots, e_i.o_j(j=|e_i.\text{OFI-list}|)$。根据 Apriori 原理,FI-list 中的每个实体 e_i 都有 e_i. OFI-list,它的每个项通过组合产生 $c_j^1, c_j^2, \ldots c_j^j$ 项目集,每个项目集分别与 e_i 合并,采用自底向上及自顶向下双向搜索策略,剪去非频繁集的超集,剪去较长最大频繁项集的子集,减少大量的项目集支持度计算。

算法 2: tobo-botoMFI.最大频繁项集的发现

输入:当前 SEFI-tree,当前窗口标识 N,最小支持度 s,用户允许的最大误差率 ε;

输出:一系列的最大频繁项集;

for each FI-list 中的 e_i do

枚举 c_j^{top} 项集,每个项集与 e_i 合并形成最大频繁候选集 E1 /* top = $|e_i.\text{OFI-list}|, \cdots, \lfloor |e_i.\text{OFI-list}|/2 \rfloor$ */

枚举 c_j^{bottom} 项集,每个项集与 e_i 合并形成最大频繁候选集 E2/* bottom = $1, \ldots, \lceil |e_i.\text{OFI-list}|/2 \rceil$ */

do 集合 E1 中的每个项目集 $E1_m$ 及 E2 中的每个项 $E2_n$

$(m, n = 1,2,\cdots)$

遍历 SEFI-tree,计算 $E1_m.esup$;

if $E1_m.esup >= s \cdot E1_m.CL$

if $E1_m \not\subset$ MFI 并且不是 MFI 的子集 then

MFI = MFI$\cup E1_m$;

从 E2 中剪去含有 $E1_m$ 的子集;

endif

遍历 SEFI-tree,计算 $E2_n.esup$;

if $E2_n.esup >= s \cdot E2_n.CL$

if $E2_n \not\subset$ MFI 并且不是 MFI 的子集 MFI then

MFI = MFI$\cup E2_n$;

else

从 E1 中剪去含有 $E2_n$ 的超集;

endif

until 集合 E1 与 E2 为空集;

endfor

2.4 算法分析

2.4.1 算法正确性分析

算法真实频率值与估计频率值的误差小于 $\varepsilon \cdot x.CL$

证明:设 x 在第 i 个窗口出现,$i=1,2,\cdots N$,且为不频繁项,在第 p 个窗口为频繁项,$p>i$,则 $x.tsup = x.esup + \sum_{i=1}^{p} x_i.esup \quad i=1,2,\ldots,p$

由 MMFI-DS 算法知,x 在第 i 个窗口内是非频繁项集知:$x_i.esup \leqslant \varepsilon \cdot x_i.CL$

$$\sum_{i=1}^{p} x_i.esup \leqslant \varepsilon \cdot \sum_{i=1}^{p} x_i.CL < \varepsilon \cdot x.CL$$

$$x.tsup = x.esup + \sum_{i=1}^{p} x_i.esup < x.esup + \varepsilon \cdot x.CL$$

即:$x.tsup - x.esup < \varepsilon \cdot x.CL$,证毕。

由 MMFI-DS 算法知,算法显然满足 ε-近似要求的 1、2 条,根据上述分析,MMFI-DS 算法满足 ε-近似要求。

2.4.2 算法复杂性比较

2.4.2.1 时间复杂性

设当前窗口的标识为 N,每个基本窗口的事务数为 T,每个事务的平均项目数为 I,FI-list 中频繁项目的个数为 k。

对于 DSM-MFI 算法:(1)在 SFI-forest 构造和维护算法中,算法需把 $(C_I^1 + C_I^2 + \ldots + C_I^I) \times T \times N$ 个项目插入到 SFI-forest 中,删除一个不频繁一项目就要遍历 $\sum_{i=1}^{\lfloor T/2 \rfloor} C_{\lfloor T/2 \rfloor + i}^{i}$ 个[13]结点;(2)在发现最大频繁项集的算法中,设 FI-list 中每个实体 $e_i(i=1,2,\cdots,k)$ 都有一个 e_i.OFI-list,其包含 1 项集的数目为 $|e_i.\text{OFI-list}|$,本文用 j_i 表示,算法需要进行 $\sum_{i=1}^{k}(C_{j_i}^1 + C_{j_i}^2 + \ldots + C_{j_i}^{j_i})$ 支持度计算,由此得出:

$$\text{DSM-MFI 算法的时间复杂度} = (I^2 - I)/2 \times T \times N + C1 \times \sum_{i=1}^{\lceil T/2 \rceil} C_{\lceil T/2 \rceil + i}^{i} + C2 \times \left(\sum_{i=1}^{k}(C_{j_i}^1 + C_{j_i}^2 + \ldots + C_{j_i}^{j_i})\right) \quad (1)$$

其中,$C1$ 为不频繁项目个数,$C2$ 为遍历一个项目集所需的时间。对于 MMFI-DS 算法:(1)在 SEFI-tree 构造和维护算法中,算法需把 $T \times N$ 个项目插入到 SEFI-tree 中,删除一个不频繁项目,算法只需删除此

不频繁项在 EIS-tree 中结点的个数，设其数目为 $C3$，显然 $C3 < \sum_{i=1}^{|T/2|} C^i_{|T/2|+i}$；(2)在发现最大频繁项集的算法中，对 FI-list 中每个实体 e_i 需要进行 $\sum_{i=1}^{k}(C^1_{j_i} + C^2_{j_i} + \cdots + C^i_{j_i})$ 支持度计算，但是通过剪枝，需 $\sum_{i=1}^{k}(F^1_{j_i} + F^2_{j_i} + \cdots + F^i_{j_i})$ 支持度计算，其中 $F^1_{j_i}, F^2_{j_i}, \cdots, F^i_{j_i}$ 分别为相应的 $C^1_{j_i}, C^2_{j_i}, \cdots, C^i_{j_i}$ 项目集剪枝后剩下的项目集，显然 $(F^1_{j_i} + F^2_{j_i} + \cdots + F^i_{j_i}) < (c^1_{j_i} + c^2_{j_i} + \cdots + c^i_{j_i})$，由此得出：

MMFI-DS 算法的时间复杂度 = $T \times N + C1 \times C3 + C2 \times \sum_{i=1}^{k}(F^1_{j_i} + F^2_{j_i} + \cdots + F^i_{j_i})$ (2)

比较(1)式与(2)式，显然(2)式 < (1)式。

2.4.2.2 空间复杂度比较

设频繁一项目的数目为 k，DSM-MFI 算法与 MMFI-DS 算法的存储结构都由 3 部分构成，相同点是都有 FI-list 与 OFI-list 两项，不同点是 DSM-MFI 算法用一系列的 SFI-tree 存储频繁项目，MMFI-DS 算法用 EIS-tree 存储频繁项目，DSM-MFI 算法需存储 2^k 个[13]频繁项目结点，MMFI-DS 算法需存储 $C \times K$（其中 C 为常数）个频繁项结点，显然，$2^k > C \times K$。

3 实验结果及分析

在文献[13]的基础上，我们用 VC++6.0 在内存 512M，CPU 为 PertiumIII-2.66GHz，操作系统为 Windows XP 的 PC 机上实现了 DSM-MFI、MMFI-DS 算法，使用了与文献[13]同样的 2 套测试数据，T10.I5.D1000K 和 T30.I20.D1000K，其中 |T| 表示数据集中事务的平均长度，|I| 表示潜在频繁项集的平均长度，|D| 表示总的事务数目。T10.I5 为稀疏数据集，T30.I20 为稠密数据集，这两个数据集有 1 000 000 个不同的事务，为了模拟数据流的持续性特点，合成数据的每个基本窗口的事务数目为 50K，1K 代表 1000 个事务，在模拟实验过程中，总共有 20 个基本窗口，ε 取值为 0.01。

实验 1（MMFI-DS 算法收缩性的研究），在实验中，我们使用了两套数据集，即 T10.I5.D1000K 和 T30.I20.D1000K，最小支持度为 0.1。图 1(a)说明，当数据从 2000K 增长到 10000K 时，运行时间增长比较平缓，图 1(b)说明，当数据不断增长时，所消耗的存储容量趋于稳定。从图 1 看出，MMFI-DS 算法具有可收缩性和可行性。

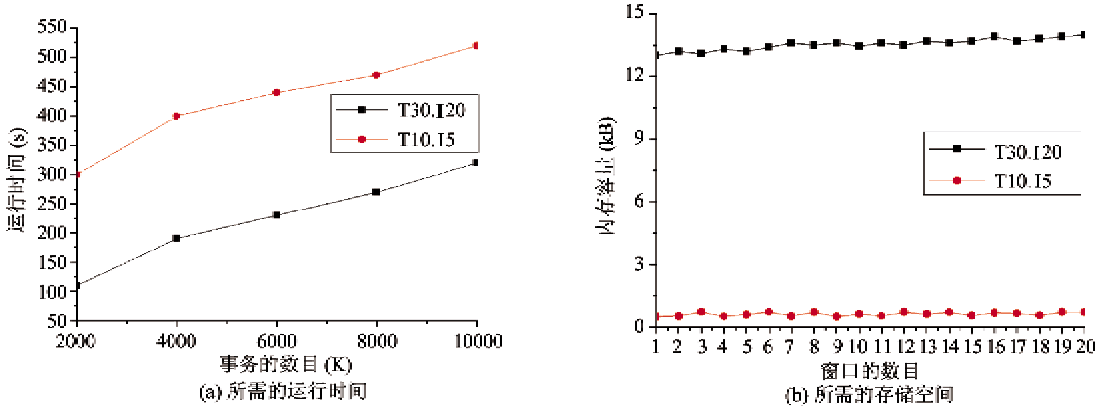

图 1 MMFI-DS 算法收缩性的研究

实验 2（DSM-MFI 与 MMFI-DS 算法在相同数据集中挖掘最大频繁集的处理时间与空间的比较），在实验中，我们使用了数据集 T30.I20.D1000K，最小支持度为 0.1。图 2(a)表明 MMFI-DS 算法的执行时间小于 DSM-MFI 算法，图 2(b)表明随着数据的增多，MMFI-DS 算法的存储容量趋于稳定，MMFI-DS 算法所需的存储容量小于 DSM-MFI 算法。

实验 3（DSM-MFI 与 MMFI-DS 算法在不同数据集中不同支持度下挖掘最大频繁集的处理时间的比较），在实验中，图 3(a)采用了 T30.I20.D1000K 数据集，图 3(b)采用了 T10.I5.D1000K 数据集，从图 3 看出，DSM-MFI 算法较适合大支持度与稀疏数据集，MMFI-DS 算法对支持度及数据集的适应性比 DSM-MFI 算法好得多。

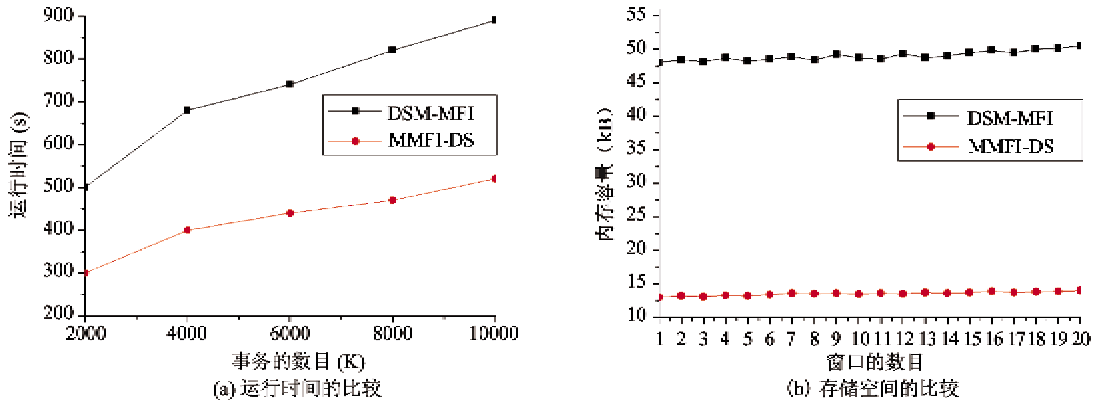

图 2　挖掘频繁集的处理时间与空间的比较

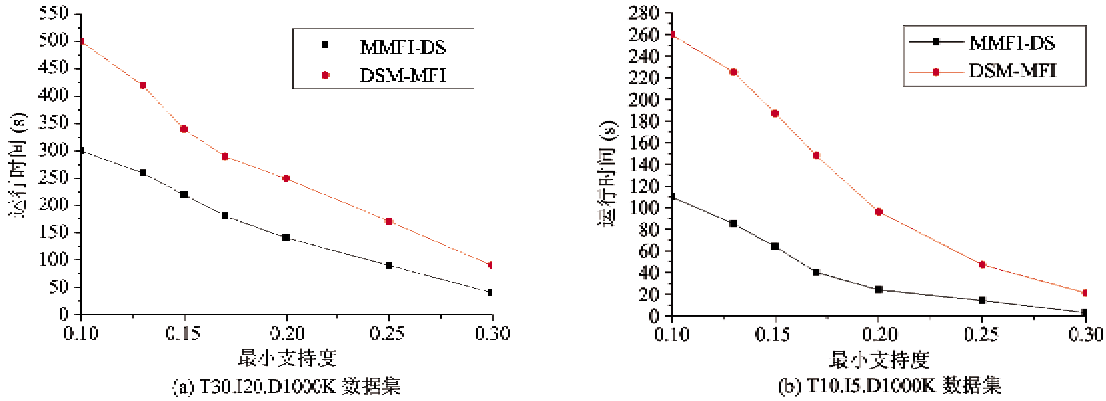

图 3　挖掘最大频繁集的处理时间的比较

4　结　论

数据流中挖掘频繁项集是当前数据挖掘领域的一个研究热点,而传统的精确挖掘频繁项集的算法不能很好地适应数据流的高速、无限、不可预测的特点。本文提出了一种数据流中最大频繁项集算法 MMFI-DS,它采用 SEFI-tree 数据结构存储有关最大频繁项集的信息,采用自顶向下和自底向上双向搜索策略,尽早剪掉较短非频繁项集的超集及较长最大频繁项集的子集,减少项目集支持度计算所耗费的时间,弥补 DSM-MFI 算法的不足,理论分析与实验表明该算法与 DSM-MFI 算法相比,能提高挖掘最大频繁项集的效率和节省存储空间。

参考文献

[1] Burdick D, Calimlim M, Gehrke J. MAFIA: A maximal frequent itemset algorithm for transactional databases. In: Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2001.443-452

[2] Pwang L, David W, Ps M. Maintenance of maximal frequent itemsets in large databases. In: Proceedings of the 2007 ACM Symposium on Applied Computing, Seoul, Korea, 2007.388-392

[3] Yang G Z. Computational aspects of mining maximal frequent patterns. *Theoretical Computer Science*, 2006, 362(1):63-85

[4] 宋余庆,朱玉全. 基于 FP-tree 的最大频繁项目集挖掘及更新算法. 软件学报, 2003, 14(9): 1586-1592

[5] 吉根林,杨明. 最大频繁项目集的快速更新.计算机学报,2005,28(1):128-135

[6] Manku G, Motwani R. Approximate frequency counts over data streams. In: Proceedings of the 28th international conference on Very Large Data, Hong Kong, China, 2002.346-357

[7] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, 1994.487-499

[8] Yu J, Chong Z, Zhang H. A false negative approach to mining frequent itemsets from high speed transactional data streams. *Information Sciences*, 2006, 176(14): 1986-2015

[9] Chi Y, Wang H, Yu P. MOMENT: maintaining closed frequent itemsets over a data stream sliding window. In: Proceedings of the 2004 IEEE International Conference on Data

Mining, Brighton, UK, 2004.59-66

[10] Li H F, Lee S Y. Mining frequent itemsets over data streams using efficient window sliding techniques. In: Expert Systems with Applications. Taiwan: Elsevier Ltd Publisher, 2009. 1466-1477

[11] Chang J, Lee W. Finding recent frequent itemsets adaptively over data stream. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Wasington D.C., USA, 2003. 487-492

[12] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD 2000), Dallas, USA, 2000. 1-12

[13] Giannella G, Han J, Yu P. Mining frequent patterns in data streams at multiple time granularities. In: Data Mining: Next Generation Challenges and Future Directions. India: PHI Publisher, 2004.191-212

[14] Li H F, Lee S Y. Online mining(recently) maximal frequent itemsets over data streams. In: Proceedings of the 15th RIDE-SDMA Conference, Tokyo, Japan, 2005.11-18

An efficient algorithm for mining maximal frequent itemsets over data streams

Mao Yinmin[*,**], Yang Luming[*], Li Hong[*], Chen Zhigang[*], Liu Lixin[*]

([*] School of Information Science and Engineering, Central South University, Changsha 410083)

([**] Applied Science Institue of Jiangxi University of Science and Technology, Ganzhou 341000)

Abstract

The paper focuses attention on the study of mining of maximal frequent itemsets from data streams to solve the problem of data and pattern redundance in frequent itemset mining, and in consideration of the problem of bad performance in operating time and memory space of the DSM-MFI, a typical algorithm for mining maximal frequent itemsets over data streams, presents an algorithm, called MMFI-DS. Firstly, the algorithm uses a new compressed tree, called the summary extended frequent item tree (SEFI-tree), to maintain the essential information about maximal frequent itemsets embedded in the stream so far, at the same time, a lot of infrequent items are deleted by pruning the tree. Then, it employs a top-bottom and bottom-top method to mine the set of all maximal frequent itemsets in landmark windows over the data stream. The theoretical analysis and experimental results show that the algorithm performs much better than the previous approaches.

Key words: data mining, data stream, landmark window, frequent itemsets, maximal frequent itemsets