高技术通讯 2010 年 第 20 卷 第 11 期:1142 ~ 1148

doi:10.3772/j.issn.1002-0470.2010.11.008

基于滑动窗口的数据流最大频繁项集的挖掘①

毛伊敏②*** 李 宏* 杨路明* 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 鉴于频繁项集存在数据和模式冗余的问题,挖掘数据流最大频繁项集的算法引起了极大的关注,本文提出了一种挖掘数据流滑动窗口内最大频繁项集算法——MMFI-SW 算法。该算法首先使用类似 FP-tree 的数据结构记录最新到达的数据流信息,同时删除过时的数据和大量的不频繁项目,然后设计一个创新的方法有效地从数据流滑动窗口中输出最大频繁项集。理论分析与实验结果表明,MMFI-SW 算法具有较低的时间复杂度。

关键词 数据挖掘,数据流,滑动窗口,频繁项集,最大频繁项集

0 引 言

数据流已在商务管理中的性能监测、网络流量管理中的异常检测及报警、零售业中的事务处理、银行业的自动柜员机管理、网站访问记录的统计与分析以及传感器网络数据的管理等领域中得到广泛的应用。数据流的处理成为当前数据库领域的一个研究热点。其中,数据流频繁项集的挖掘是数据流挖掘中最基本的问题之一。由于最大频繁项集隐含了所有频繁项集,挖掘数据流最大频繁项集更具有挑战意义。针对传统挖掘算法不能很好地适应数据流的高速、无限、不可预测的特点,本文提出了一种在滑动窗口(sliding windows, SW)挖掘最大频繁项集(mining maximal frequent itemset, MMFI)的算法——MMFI-SW 算法,并对其进行了详细报道。

1 研究背景

目前,许多数据流频繁模式挖掘算法被陆续提出,按照数据流的处理模型进行分类,数据流频繁模式挖掘的研究分为三类:界标窗口类、衰减窗口类和滑动窗口类。

(1)在界标窗口处理模型。2005 年,Li 等提出了 DSM-MFI[1]算法,它基于 Apriori[2]算法思想,使用基于前缀树结构存储数据流信息,实现最大频繁项集的挖掘;2006 年,Yu 等提出了 false-negative 算法[3],它基于切尔诺夫约束,从高速的事务型数据流中挖掘频繁项集,使用运行错误参数剪掉非频繁项集和可靠性参数控制内存;2007 年,Mao 提出了 INSTANT[4]算法,它在内存中维护不同支持度级别的项集,并对项集定义了一些子操作(sub-opertator),当新事务到达时,通过一系列的子操作,直接把最大频繁项陈列给用户。

(2)在衰减窗口处理模型。2003 年,Giannella 等提出了一种传统的频繁模式树(FP-Tree)[5]改造的处理数据流的算法——FP-stream 算法[6],它利用不同时间粒度实现不同时间段的频繁项集的生成工作;2008 年,Ming 等人提出 MRVSDS[7]算法,存储当前窗口的频繁项集到 PFI-tree 中,当项目的衰减支持度小于最小支持度时,项目从 PFI-tree 中删除,此外设计了 DSYV 结构用来存储被处理的事务,当最近的支持度小于历史最小支持度时,重新挖掘 DSYV 中的事务来发现频繁项集。

(3)在滑动窗口处理模型。2004 年,Chi 等提出了 Moment[8]算法,它采用前缀项目树存储、维护结点信息,通过结点类型转换,对其进行不同的处理从而挖掘闭频繁项集;2006 年,Jiang 等提出了一种对 Moment 改进的 CFI-Stream 算法[9],它使用 DIUtree 维护闭项目集的信息,极大节省存储空间;2008 年,

① 国家自然科学基金(60873082)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:2009-08-10)

Ranganath 等人提出 Stream-Close[10] 算法,使用 LDIUtree 存储当前闭频繁项集,设计了"最大项目匹配"和"相同前缀"剪枝策略缩小搜索间,在挖掘过程中采用"向前看"方法进行闭包检测从而获得当前窗口的闭频繁项集;2009 年,Li[11] 等人提出了 MFI-TransSW 算法,每个事务的项目用二进制位序列表示,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。

在滑动窗口中,新数据不断进入,旧数据不断被淘汰,滑动窗口包含的数据集既增又减,其频繁模式的挖掘更为困难。然而文献[1,4]提出的是基于界标窗口而设计的最大频繁项集挖掘算法,文献[8-11]提出的是在滑动窗口中挖掘频繁项集和闭频繁项集的算法。根据我们目前所掌握的信息,还没有文献给出滑动窗口中挖掘最大频繁项集的算法。针对这个问题,本文提出 MMFI-SW 算法。该方法使用结构紧凑的当前频繁模式树(the current frequent pattern tree,CFP-tree)来压缩存储数据流滑动窗口内的频繁项,当新的数据流流进时,CFP-tree 可以及时增量捕获数据流中的最新信息。当过时的数据流流出时,不需遍历整个 CFP-tree,通过对指针的操作,该算法从 CFP-tree 中删除过时的项目和大量的不频繁项目,从而减少 CFP-tree 的空间复杂度与维护代价,然后采用一种创新的方法有效地在数据流环境中挖掘最大频繁项集。

2 问题定义

本节主要介绍挖掘数据流滑动窗口内最大频繁项集的相关概念。

定义 1 一个数据流是一个连续的、无穷的基本窗口序列,$DS=[w_1,w_2,\cdots,w_n]$,$w_i,\forall i=1,2,\cdots,N$,$i$ 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,$<T_1,T_2,\cdots,T_k,\cdots>$,$T_k=\{x_1,x_2,\cdots,x_m,\cdots\}$,$T_k(k=1,2,\cdots)$ 称为事务,$x_i(i=1,2,\cdots,p)$ 称为项目。

定义 2 滑动窗口 sw 是由一系列连续的基本窗口组成,记为 $[w_1,w_2,\cdots,w_N]$。每一个滑动窗口包含的基本窗口的个数就是滑动窗口的大小。

定义 3 设给定的支持度 s 和允许偏差 $\varepsilon(\varepsilon<s)$,$|N|$ 表示滑动窗口 sw 中的事务数,滑动窗口 sw 项目 x_i 的支持数记为 $f_{sw}(x_i)$,如果有 $f_{sw}(x_i)>(s-\varepsilon)|N|$,则称 x_i 为滑动窗口 sw 中的频繁项目集,简称频繁项集;如果有 $f_{sw}(x_i)>\varepsilon|N|$,则称 x_i 为滑动窗口 sw 中的潜在频繁项集;如果有 $f_{sw}(x_i)\leqslant\varepsilon|N|$,则称 x_i 为滑动窗口 sw 中的不频繁项集。

定义 4 $f'_{sw}(x_i)$ 是项目 x_i 的真实支持度,其值为项目 x_i 在事务中所包含的数目,$f_{sw}(x_i)$ 是项目 x_i 的估计支持度,其值为项目 x_i 在概要数据结构中所包含的数目,并且 $1\leqslant f_{sw}(x_i)\leqslant f'_{sw}(x_i)$。

定义 5 设项目 X、Y,若 $X\prec Y$,则项目 X 的字典序列小于项目 Y。

定义 6 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 7 若频繁项集 X 的所有超集都是非频繁项集,则称 X 为最大频繁项集;将所有最大频繁项集组成的集合称为最大频繁集(maximum frequent sets,MFS)。

3 算法描述

本节主要介绍 CFP-tree,并详细讨论基于该数据结构挖掘数据流滑动窗口内最大频繁项目的 MMFI-SW 算法。

3.1 CFP-tree 的结构

FP-tree 具有的优点是包含挖掘频繁项集的全部信息及高度压缩存储频繁项。但是,它需要两遍扫描数据库,不适应数据流的挖掘。文献[8]提出了"结点类型转化"的思想,具有的优点是它提出数据流滑动窗口内结点类型发生变化的频度比较小,维护结点的时间开销代价小。但是,它采用前缀项目树存储结点信息,存储结点的空间代价高。本文采用文献[8]思想的优点并结合 FP-tree 的优点,开创设计 CFP-tree,挖掘滑动窗口中数据流的最大频繁项集。CFP-tree 具有如下的性质:

(1)它由一个 *IIS-tree*(improved item suffix tree)、*Item-list* 以及 *Tid-list* 组成;

(2) CFP-tree 中的项目按照字典序列排序;

(3) *IIS-tree* 除了具有 FP-tree 中 *IS-tree*[5] 每个结点所包含的 *item-name*,*count*,*node-link* 3 个域外,还有 *freq-sign*,*freq-sign* 标识该结点的类型,0 表示频繁

项,1表示潜在频繁项,2表示不频繁项;

(4) 数据流中所有的项目按照字典序列存放在 *Item-list* 表中,每个结点包含 *item-name*, *count*, *freq-sign*, *head of node-link* 4个域,其中,*head of node-link* 为指向 *IIS-tree* 中具有相同 *item-name* 的首结点的指针;

(5) *Tid-list* 的每个结点有 *tid* 和 *pointer* 两个域,*tid* 为事务的编号,*pointer* 为指向 *IIS-tree* 中每个分支的最后一个项目的指针。

本文设计的 CFP-tree 拥有如下特点:(1)高度压缩性。CFP-tree 各分支上的结点均按照字典序列方式排列,字典序列较后的项目有较多的共享前缀模式。(2)良好地动态捕捉数据流信息。当数据流到达时,每个事务的项目按照预先定义好的字典顺序排列数据项并插入到 CFP-tree 树中,不需二次扫描数据流。(3)维护时间开销小。CFP-tree 增加 *Tid-list* 表,用 *pointer* 指向 CFP-tree 中每个事务的最后一个项目,当数据流流出时,不需遍历整个 CFP-tree,能够快速移走过时的事务。此外,CFP-tree 添设 *freq-sign* 标识结点类型,方便在 CFP-tree 维护阶段和最大频繁项集输出过程中对不同模式对象的处理。

3.2 CFP-tree 的增量算法

滑动窗口的信息以基本窗口的事务数据集合为单位不断发生变化,当新的基本窗口的事务数据流入滑动窗口时,过时的基本窗口的事务数据集合从滑动窗口中删除,因此,新的事务数据流到达时,必须及时捕获最新的模式信息并把它们按照事先约定好的字典顺序储存到 CFP-tree 中,其详细过程如算法1所示。

算法1 UpdateCFP-tree

输入:*TC*:当前数据流的事务;

s:支持度;

ε:用户允许的最大误差率;

输出:产生一个 CFP-tree;

获得每个事务 *TC* 的投影 *TC'*;

for each item $x_i \in TC'$ do

if $x_i \notin$ *Item-list* then

构造一个新的实体(x_i, 1,2, *node-link*)并插入到 *Item-list* 的适当位置;

else

$x_i \cdot count = x_i \cdot count + 1$;

if x_i 的频繁类型发生变化 then

改变 *IIS-tree* 中 *item-name* $\notin x_i$ 的所有结点的频繁类型标识;

endif

endif

if *IIS-tree* 具有一个项目名为 y 且 $y.item\text{-}name \notin x_i.item\text{-}name$ then

$y.count = y.count + 1$;

else

构造一个新的实体(x_i,1,2, *node-link*)并把它插入 *IIS-tree* 中;

endif

endfor

3.3 CFP-tree 的剪枝算法

CFP-tree 捕获了当前滑动窗口中所有数据流的完整信息,其中包含了数据流中大量的不频繁项目,随着数据流不断涌进滑动窗口,不频繁项目的数量迅速膨大。另外,当滑动窗口移动时,保存在 CFP-tree 中的那些过时的事务数据集已不受到关注。大量的不频繁项目和过时的基本窗口事务数据集存储在 CFP-tree 中,极大地浪费存储空间及增大 CFP-tree 的维护代价和最大频繁项集输出代价,因此,当过时的事务数据集流出滑动窗口时,必须从 CFP-tree 中剪掉大量不频繁项目及过时窗口事务数据集,从而缩小存储空间和提高时间效率。

性质1 删除 CFP-tree 不频繁项目,不会影响最大频繁项集的正确输出。

证明:假设滑动窗口 *sw* 是由一系列连续的基本窗口 $sw_1, sw_2, \cdots, sw_n$ 组成,n 为滑动窗口数,$|N|$ 为滑动窗口的事务数,$|sw_n|$ 为基本窗口的事务数,ε 为用户给定的最大允许偏差。

对于滑动窗口的任意一个项目 x_i,如果它在基本窗口 sw_n 的支持数为 $f_{sw_i}(x_i)$,在 m 个基本窗口为不频繁项,在 $n-m$ 个基本窗口为潜在频繁项目或频繁项目,则项目 x_i 在滑动窗口 *sw* 真实的支持度计数为 $f'_{sw}(x_i) = f_{sw}(x_i) + \sum_{j=1}^{m} f_{sw_j}(x_i)$, $m < n$,由不频繁项定义知 $f_{sw_i}(x_i) \leqslant \varepsilon \times |SW_i| \sum_{j=1}^{m} f_{BW_j}(x) \leqslant \varepsilon \times \sum_{j=1}^{m} |BW_j| < \varepsilon \times |N|$,即 $f'_{SW}(x) - f_{SW}(x) < \varepsilon \times |N|$,证毕。

由上述分析知,项目的真实频率值与估计频率值的误差小于用户所允许的误差,剪掉 CFP-tree 不频繁项目将不会影响最大频繁项集的正确输出。

性质2 对 CFP-tree 中的任一条路径,其结点的

支持度计数按降序排序,所有叶子结点代表频繁支持度计数最少的项目。

证明:设 $node_i$ 和 $node_j$ 是 CFP-tree 中任一条路径的两个结点,$node_i$ 是 $node_j$ 的父结点。

$node_i$ 是 $node_j$ 的父结点,根据 CFP-tree 的特点和算法 1 知,$node_i < node_j$。此外,$node_i$ 和 $node_j$ 在同一条路径,往 CFP-tree 中添加 $node_j$ 时,一定要添加 $node_i$。但是,往 CFP-tree 中添加 $node_i$ 时,不一定添加 $node_j$。所以,$node_i.count > node_j.count$

由算法 1 知,所有的叶子结点都是每个分支路径最后插入的结点。由上述分析知,其父结点的频繁计数都大于叶子结点,所以叶子结点代表频繁支持度计数最少的项目。

由于滑动窗口的滑动,CFP-tree 中某些结点的模式类型会发生变化,算法通过如下两步操作,及时跟踪结点的模式信息动态变化。第一步,在 *Tid-list* 链表中存储了当前窗口所有事务数据集的事务标识 *tid*,每个事务的 *tid* 都对应一个 *pointer*,它指向 *IIS-tree* 中该事务的最后一个结点。通过操作该指针,快速地把过时事务数据集的支持度计数减 1。第二步,对项目列表的各项目的支持度计数需要重新计算并对那些模式类型发生变化的结点设置新的模式类型。

完成上述操作后,CFP-tree 还记录了大量的不频繁项目和支持度计数为零的过时项目,下面分两种情况对它们进行剪枝。(1)对于 $\forall x_i \in Item-list$,如果 $f_{sw}(x_i) \leqslant \varepsilon \mid N \mid$,则 x_i 为不频繁项目。根据性质 2,x_i 的子孙也为不频繁项目。根据性质 1,*IIS-tree* 中所有与该 x_i 同名结点及子孙结点均可删除。(2)对于 $\forall x_i \in Item-list$,如果 $f_{sw}(x_i) > \varepsilon \mid N \mid$,则 x_i 为潜在频繁项目或频繁项目。根据性质 1,不能删除 *IIS-tree* 中所有与 x_i 同名的结点,但是如果 *IIS-tree* 中某个 $x_i.count = 0$,则这个 x_i 为过期项目。根据性质 2,x_i 的子孙也为过期项目。因此,*IIS-tree* 中所有与该 x_i 同名结点及子孙结点均可删除。从上述的分析知,删除过时的事务数据集及大量的不频繁项目的整个剪枝操作过程,不需要遍历 CFP-tree,从而提高了模式树剪枝操作的时间效率,其剪枝操作过程如算法 2 所示。

算法 2 PruningCFP-tree

输入:一个 CFP-tree;

s:支持度;

ε:用户允许的最大误差率;

tid_c:当前事务的编号;

输出:一个被裁剪过的 CFP-tree;

```
for each tid < tid_c, where tid ∈ Tid-list do
  for each x_i ∈ tid 对应 pointer 所指向的路径
   x_i · count = x_i · count − 1;
  endfor
endfor
for each x_i ∈ Item-list do
  统计 x_i · count;
  if x_i 的频繁类型发生变化 then
    改变 IIS-tree 中 iten-name ∉ x_i 的所有结点的频
    繁类型标识;
  endif
  if f_sw(x_i) < ε | N |
    删除 IIS-tree 中 item-name ∉ x_i 的所有结点及子
    孙结点;
  从 Item-list 中删除 x_i;
else
  删除 IIS-tree 中 item-name ∉ x_i 且 x_i · count = 0 的
  所有结点及子孙结点;
    endif
  endfor
```

3.4 最大频繁项集挖掘算法

在静态环境中,FPMAX[12]算法是一种基于 FP-tree 结构的高效挖掘最大频繁项集的算法,它在许多数据类型环境中具有较好的时间效率[12]。由于 CFP-tree 与 FP-tree 在结构上相类似,在各自处理模式信息的对象存在着差异,因此,对 FPMAX 算法进行改进以适应从 CFP-tree 中输出最大频繁项集。

性质 3 一棵被裁剪过的 CFP-tree,潜在频繁项不一定都是叶子结点。

证明:设 $\mid N \mid$ 为滑动窗口的事物数,ε 为用户允许的最大误差率,叶子结点为 $node_{\text{leaves}}$,假设所有的叶子结点为潜在频繁项,即 $f_{sw}(node_{\text{leaves}}) > \varepsilon \mid N \mid$。

在 CFP-tree 中存在这样的情况:有个项目 x_m,其在字典序列的顺序较后,即 $x_1 < x_2 \cdots < x_m$,x_m 为叶子结点,$f_{sw}(x_m) = p$ 且 $p > (s-\varepsilon) \mid N \mid$。根据算法 1 与性质 2 知,存在 p 个项目名为 x_m,$f_{sw}(x_m) = 1 < \varepsilon \mid N \mid$,即 x_m 可能是 CFP-tree 中 p 个分支的叶子结点,这与假设相矛盾,命题得证。

由性质 2 得知,FP-tree 与 CFP-tree 的每个分支上的所有结点都是按结点的频繁支持度计数降序排序,从而便利地建立每个项目的条件模式基。但是,FP-tree 存储数据库中所有的频繁项目,CFP-tree 存储了滑动窗口内的频繁项目和潜在频繁项目。因

此,与 FPMAX 算法相比,从 CFP-tree 中挖掘最大频繁项目要做如下的改进:(1)算法要忽略 *Item-list* 中的所有潜在频繁项目,只能在 *Item-list* 的所有频繁项目中建立条件模式基;(2)根据性质3,当建立某个项目的条件模式基时,该条件模式基只能包含频繁项目,不能包含潜在频繁项目;(3)设计类似 CFP-tree 结构的改进的最大频繁项集(improved maximal frequent itemsets, IMFI-tree),跟踪所有的最大频繁项目集。当用户发出发现滑动窗口最大频繁项目的请求时,挖掘最大频繁项目的过程如算法 3 所示。

算法 3 Mining CFP-tree

输入:*T*:一颗 CFP-tree;

IMFIT:一颗 IMFI-tree;

Head:一个项目链表;

输出:一棵包含所有最大频繁项目的 *IMFIT*;

if *T* 包含单个路径 *P* then

插入 *Head* ∪ *P* 到 *IMFIT*;

else for each *Item-list* 中的 $x_1 > (s - \varepsilon) \mid N \mid$

添加 x_1 到 *Head*;

产生 *Head* 的条件模式基;

Tail = {模式基里的频繁项目};

subset checking(*Head* ∪ *Tail*);

if *Head* ∪ *Tail* ∉ *MFIT*

构造 CFP-tree *THead*;

call Mining CFP-tree(*THead*);

endif

从 *Head* 链表中移走 x_1;

endif

IMFI-tree 类似 CFP-tree,它满足以下三个条件的树型结构:(1)它由一个标为 *root* 的根结点、作为根结点的孩子的项目前缀子树集合以及项目列表组成;(2)子树中的每个结点由 *item-name* 和 *node-link* 2个域组成, *item-name* 记录项目名, *node-link* 指向 CFP-tree 中具有相同 *item-name* 值的下一个结点;(3)项目列表按照项目的字典序列排序,每一个表项包含 *item-name* 和 *head of node-link* 2个域, *head of node-link* 指向 CFP-tree 中具有相同的 *item-name* 值的首结点的指针。

3.5 理论分析

(1) MMFI-SW 算法满足 ε 近似要求。

证明:设任一项目 x_i, 滑动窗口的窗口数为 n, 事务总数为 $\mid N \mid$, 由 MMFI-SW 算法知,当用户发出查询请求时,分两种情况进行讨论。

①频繁项被输出:设任一频繁项目 x_i, 它在 m 个基本窗口为不频繁项目,即

$$f'_{sw}(x_i) = f_{sw}(x_i) + \sum_{j=1}^{m} f_{sw_j}(x_i), m < n$$

由频繁项目的定义知:

$$f'_{sw}(x_i) = f_{sw}(x_i) + \sum_{j=1}^{m} f_{sw_j}(x_i) > (s - \varepsilon) \mid N \mid \tag{1}$$

由不频繁定义知:

$$\sum_{j=1}^{m} f_{sw_j}(x_i) \leqslant \varepsilon \mid N \mid \tag{2}$$

由定义 4 知:

$$f_{sw}(x_i) \leqslant f'_{sw}(x_i) \tag{3}$$

由式(1)、(2)、(3)得 $f'_{sw}(x_i) > s \mid N \mid$。

②潜在的和不频繁项目不被输出:设任一项目 x_i 为潜在的或不频繁项目,由频繁项目的定义显然知:

$$f'_{sw}(x_i) < (s - \varepsilon) \mid N \mid$$

从上述的分析和性质 1 可推出:MMFI-SW 算法满足 ε 近似要求。

(2) CFP-tree 的空间复杂度 $O(\mid f_1 \mid + \mid f_2 \mid)$, $\mid f_1 \mid$, $\mid f_2 \mid$ 分别为当前窗口频繁和潜在频繁项目的数目。

证明:由 MMFI-SW 算法知,当前滑动窗口中储存的是频繁项目和潜在的频繁项目,不频繁项目与过时的频繁项目全都从 CFP-tree 中删除,由文献[5]得知, FP-tree 的空间复杂度为 $O(\mid Trans \mid)$, $\mid Trans \mid$ 为频繁项目的个数,因此,问题得证。

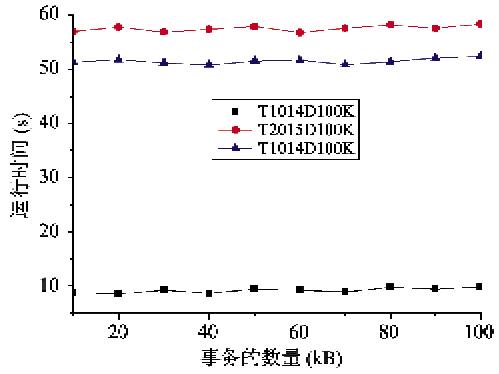
4 实验结果及分析

为了评估算法的性能,所有的实验在 CPU 为 2.0GHz 的 PentiumIV,内存为 1GB,操作系统为 Windows XP 的 PC 机上进行,所有的实验程序均采用 C 语言编写,在 VC++6.0 环境中运行,实验中的模拟数据由 IBM 模拟数据产生器(http://www.almaden.ibm.com)产生,合成数据的主要参数为:T 表示事务的平均长度,I 表示频繁项集的平均长度,D 表示事务的数目,1K 代表 1000 条事务的记录,实验中的所有的项目数目固定在 1000,用户允许最大误差为 $s/100$。

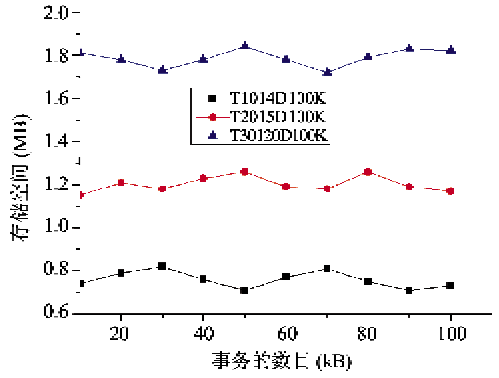
4.1 算法的性能

实验分别使用 T10I4D100K、T20I5D100K 和 T30I20D100K 三套数据集来评估算法的性能,这三套数据集的最小支持度分别为 0.3%、6% 和 15%。

设滑动窗口的大小为 10k,所有的实验在 10 个连续滑动窗口中进行。图 1(a)说明,随着事务数据流的增多,在多种数据类型环境中算法所消耗的时间变化不大。图 1(b)说明,当滑动窗口不断移动时,在三套数据集下算法所需的存储容量变化幅度不大。从图 1 看出,MMFI-SW 算法具有较好的性能。



(a) 算法的运行时间

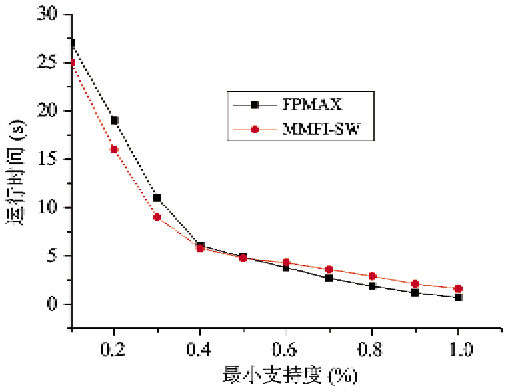


(b) 算法的存储空间

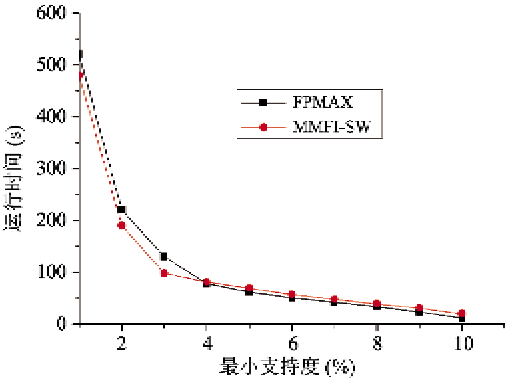
图 1　MMFI-SW 算法的性能

4.2　算法的时间效率

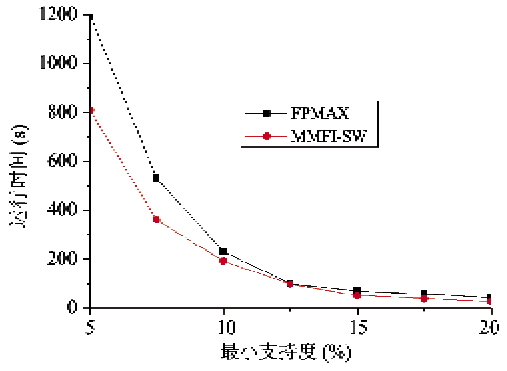
考察算法时间效率的实验是在一个滑动窗口下运行 MMFI-SW 算法的增量更新、剪枝和挖掘最大频繁项集三部分,设滑动窗口的数目为 10k,FPMAX 算法也在事务数目为 10k 的数据集下运行,实验分别采用 T10I4D10K、T20I5D10K 和 T30I20D10 数据集来评估算法处理短事务短模式、长事务短模式和长事务长模式的能力。图 2 说明,MMFI-SW 算法的运行时间与 FPMAX 算法的运行时间相差不大,FPMAX 算法在多种数据集下多种情况下具有较好的时间效率[12],因此,MMFI-SW 算法在多种数据类型环境中也具有较好的时间效率。



(a) T10I4D100K



(b) T20I5D100K



(c) T30I20D100K

图 2　MMFI-SW 算法的时间效率

5　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。针对这个问题,本文提出了一种挖掘数据流中最大频繁项集的 MMFI-SW 算法,当数据流流进时,它使用 CFP-tree 数据结构及时捕获最新数据流的信息,当数据流流出时,删除过时的数据流信息及不频

繁项目,设计一个开创性的方法,从 CFP-reee 数据结构中输出一系列数据流滑动窗口内的最大频繁项集。理论分析与实验表明该算法具有较好的时空效率。

参考文献

[1] Li H F, Lee S Y. Online mining (recently) maximal frequent itemsets over data streams. In:Proceedings of the 15th International Workshop on Research Issues in Data Engineering : Stream Data Mining and Application, Tokyo, Japan, 2005. 11-18

[2] 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

[3] 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

[4] Mao G J, Wu X D, Liu C N. Onling mining of maximal frequent itemsequences from data streams. *Journal of Information Science*, 2007,33(3):251-262

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

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

[7] Ming Y L, Chen H S. Interactive mining of frequent patterns in a data stream of time-fading models. In:Proceedings of the 8th International Conference on Intelligent Systems Design and Applications,Kaohsiung, Taiwan,China, 2008.513-518

[8] 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

[9] Jiang N, Gruenwald. CFI-stream: mining closed frequent itemsets in data streams. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,Philadelphia,USA,2006.592-597

[10] Ranganath B N, Murty M N. Stream-close: fast mining of closed frequent itemsets in high speed data streams. In:Proceeding of 2008 IEEE International Conference on Data Mining Workshops,Pisa,Italy,2008.516-525

[11] Li H F, Lee S Y. Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, 2009,36(2):1466-1477

[12] Grahne G, Zhu J F. High performance mining of maximal frequent itemsets. In:Proceedings of the 6th SIAM International Workshop on High Performance Data Mining,San Francisco, USA,2003. 135-143

Mining maximal frequent itemsets in a sliding window over data streams

Mao Yimin[*,**], Li Hong[*], Yang Luming[*], 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

In consideration of the problem of data and pattern redundancy in frequent itemsets mining and the close attention to the study of mining maximal frequent itemsets from data streams, this paper presents the MMFI-SW algorithm for mining maximal frequent itemsets in a sliding window over data streams. Firstly, it uses a data structure based on FP-tree to record the current information in streams, at the same time, the obsolete items and a lot of infrequent items are deleted by pruning the tree. Then, it designs a novel method to mine the set of all maximal frequent itemsets in a sliding window over data streams. The theoretical analysis and the experimental results show that the proposed method is efficient.

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