2019 年 1 月 第 1 期
Vol. 40 No. 1 2019

滑动窗口中 FP-Tree 的频繁项集挖掘算法的研究

李芬田,王红梅,潘 超

(长春工业大学 计算机科学与工程学院,长春 130012)
E-mail: wanghm@ccut.edu.cn

摘 要: 当有大量的事务插入或者删除时,针对 pWin 算法在窗口滑动阶段反复访问前缀树进行事务的更新; DSFPM 算法中 DSFPM-Tree 中大量的父子之间存在不频繁的关系,因此建立的 DSFPM-Tree 比较高,特别是在窗口滑动的时候,需要频繁更新 DSFPM-Tree 带来很大的时间开销等缺点,提出滑动窗口中 FP-Tree 的频繁项集挖掘算法. 算法将数据流分成大小相等的模块来进行挖掘,每个模块均采用上三角矩阵存储,并且设计了一种概要结构 NCFP-Tree 来存储每个基本窗口中的临界频繁项集,窗口每次滑动一个基本窗口,利用优化的频繁项集挖掘算法,分别把各个基本窗口中的临界频繁项集挖掘出来. 用 C 实现了该算法,实验结果证明了该算法比其他两个算法的时间效率更高,查全率和查准率都优于其它两个算法,具有良好的性能.

关 键 词: 数据流; 滑动窗口; 数据挖掘; 临界频繁项集
中图分类号: TP311 **文献标识码:** A **文 章 编 号:** 1000-1220(2019)01-0045-05

Research on Frequent Itemsets Mining Algorithm of FP-tree in Sliding Window

LI Fen-tian, WANG Hong-mei, PAN Chao
(School of Computer Science & Engineering, Changchun University of Technology, Changchun 130012, China)

Abstract: When a large number of transactions are inserted or deleted, the pWin algorithm repeatedly accesses the prefix tree for transaction updating during the window sliding phase. The DSFPM algorithm has an infrequent relationship between a large number of father-son relationships in the DSFPM-Tree. Therefore, the established DSFPM-Tree is relatively high. In particular, when the window is sliding, frequent updating of the DSFPM-Tree requires a lot of time overhead, and the FP-Tree frequent itemsets mining algorithm in the sliding window is proposed. The algorithm divides the data flow into modules of equal size for mining. Each module adopts the upper triangular matrix storage, and a synoptic structure NCFP-Tree is designed to store the critical frequent itemsets in each basic window. Each time the window slides a basic window, the use of optimized frequent itemsets mining algorithm, respectively, each of the basic window in the critical frequent itemsets mining. The algorithm is implemented in C, and the experimental results show that the algorithm is more efficient than the other two algorithms in time efficiency, the recall rate and precision are better than the other two algorithms, with good performance.

Key words: data flow; sliding window; data mining; critical frequent itemsets

1 引 言

数据流[1]的应用越来越广泛,比如我们常见的网络监控、股票的交易、网络通讯的监控、传感器网络和天气或环境的检测都会生成大量的数据流. 随着数据流应用的普及,数据流环境下的数据挖掘越来越引起人们的兴趣,又因为数据流中的频繁项集又是数据流挖掘中最基本的问题之一,所以近十几年得到许多学者的研究,但是由于数据流具有连续、无限、快速、随着时间变化且不可预知的等特性,从而在数据流环境下挖掘频繁项集带来了很大的挑战. 近几年来大量的数据流频繁项集挖掘算法被学者们陆续提出[2-5]. 其中最典型的是 Han 等人提出的 FP-Growth 算法[6],Manku 等人提出的 estDec 算法[7],Leung 等人提出的 DSTree 算法[8] 和 Giannella 等人提出的 FP-stream 算法[9],FP-stream 算法将最近的频繁项集保存在倾斜时间窗口里,将旧的频繁项集以粗糙的时间粒度保存,设计了 Pattern-tree 来维护频繁模式信息,进而实现数据流上频繁项集的挖掘; Lee 等人提出 WMFP-SW 算法[10] 和刘学军等人提出的 DS-CFI 算法[11] 是在 FP-Growth 算法上的改进实现了频繁项集的挖掘,但是这两个算法在处理稠密数据集时动态更新的速度比较慢; 李国徽等人提出的 DSFPM 算法[12] 能动态的挖掘出滑动窗口内的完全频繁项集,但是 DSFPM-Tree 中大量的父子节点存在不频繁的关系时,建立的 DSFPM-Tree 比较高,并且树中分支多,最关键的是窗口在滑动时,需要频繁的更新 DSFPM-Tree,导致时间开销比较大; Deypir 等人提出的 pWin 算法[13] 采用前缀树存储频繁项集,用反向深度遍历挖掘频繁项集,但是当有大量的事务插入或删除时,频繁的访问前缀树比较浪费时间.

本文针对以上算法的不足,提出了滑动窗口中 FP-Tree 的频繁项集挖掘算法 MFI-SW(A frequent itemset mining algorithm in the sliding window),MFI-SW 算法将数据流分块挖

收稿日期: 2018-03-06 收修改稿日期: 2018-03-26 基金项目: 吉林省科技厅发展计划项目(20160415013JH)资助; 吉林省科技厅重大科技招标专项(20160203010GX)资助. 作者简介: 李芬田,女,1990 年,硕士研究生,研究方向为数据挖掘及应用; 王红梅,女,1968 年生,博士,教授,研究方向为数据挖掘及应用; 潘 超,男,1981 年生,硕士,讲师,研究方向为数据挖掘、机器学习.

掘,采用上三角矩阵存储,在内存中用 NCFP-Tree 保存每个基本窗口中的临界频繁项集,窗口滑动阶段只需要将旧的 NCFP-Tree 删除,建立新的 NCFP-Tree 即可,然后采用优化的算法挖掘每个基本窗口中临界频繁项集,进而挖掘出完全频繁项集.

2 问题描述

定义 1 [14]. (数据流):设 $I=\{I_1,I_2,\cdots,I_n\}$ 是项的集合,I_i 表示第 i 项($1\leqslant i\leqslant n$),数据流 DS 是一组连续不断到达的数据序列$\{T_1,T_2,\cdots,T_n\}$的集合,T_j 表示第 j 个到达的事务($1\leqslant j\leqslant m$),对于任意 T_j 都有 $T_j\subseteq I$.

定义 2 [4]. (滑动窗口):将数据流中的数据分成大小相等的块,这样的每个分块是一个基本窗口,记作 w. 每块中包含的事务个数是基本窗口的大小,记作$|w|$. k 个连续的基本窗口组成一个滑动窗口 W,记为 $W=\langle w_1,w_2,\cdots,w_k\rangle$,k 值是预先固定的. 滑动窗口的大小指的是每个滑动窗口中基本窗口的个数,记为$|W|$.

支持度 [14]:$\forall X\subseteq I$ 滑动窗口中包含项集 X 的事务个数称为 X 的支持度计数,记为 S,若 $\sup(X)=S/|w|$,则 $\sup(X)$ 称为 X 在 w 中的支持度.

定义 3. (频繁项集,临界频繁项集,非频繁项集):对于用户给定的最小支持度阈值为 ζ 和误差因子为 ε,假设$|W|$($|w|$)表示滑动窗口 W(基本窗口 w)的大小. 在滑动窗口 W(基本窗口 w)中模式 A 的支持度计数用 $f_W(A)$($f_w(A)$)表示. 如果满足 $f_W(A)\geqslant\zeta|W|$($f_w(A)\geqslant\zeta|w|$),就称模式 A 是滑动窗口 W(基本窗口 w)中的频繁项集. 如果满足 $f_W(A)\geqslant(\zeta-\varepsilon)|W|$($f_w(A)\geqslant(\zeta-\varepsilon)|w|$),则称为模式 A 是滑动窗口 W(基本窗口 w)中的临界频繁项集. 如果满足 $f_W(A)<\varepsilon|W|$($f_w(A)<\varepsilon|w|$),就称模式 A 是滑动窗口 W(基本窗口 w)中的非频繁项集.

性质 1 [14]. 假设 X 是非频繁项集,那么它的任何超集都是非频繁项集.

性质 2. 在 NCFP-Tree 中的父亲和孩子组成的 2-项集包含临界频繁 2-项集的全部信息.

证明: 因为在构建 NCFP-Tree 时,只有插入节点和某个祖先节点能够组成临界频繁 2-项集才可以插入,否则删除该结点.

性质 3. 项目矩阵 PM 中包含频繁 1-项集和频繁 2-项集的全部信息.

定理 1. 每个基本窗口中的非频繁项集被删除后,尽管它们在整个滑动窗口中,是频繁的,其中输出支持度的误差也不会超过 ε,是可以接受的.

证明: 设$|w|$表示一个基本窗口的宽度,$|W|$表示滑动窗口的长度,设定最小支持度阈值为 ζ 和误差因子为 ε,如果模式 A 在滑动窗口中是频繁的,则满足 $f_W(A)\geqslant\zeta|W|$,如果在基本窗口 $w_i(1\leqslant i\leqslant k)$ 中是临界频繁的,则满足 $f_{wi}(A)\geqslant(\zeta-\varepsilon)|w_i|$. 如果在基本窗口 $w_j(1\leqslant j\leqslant k)$ 中是非频繁的,则满足 $f_{wj}(A)<\varepsilon|w_j|$. 因此,$f_W(A)=\sum f_{wi}(A)+\sum f_{wj}(A)$,所以对模式 A 支持度为 $f_W(A)=\sum f_{wi}(A)=f_W(A)-\sum f_{wj}(A)\geqslant\zeta|W|-\sum\varepsilon|w|\geqslant(\zeta-\varepsilon)|W|$. 因此得证,由定理 1 知,仅仅保存每个基本窗口中的临界频繁项集,就可以满足在误差允许的范围内输出的项集是滑动窗口中所有的频繁项集.

3 MFI-SW 算法描述

3.1 MFI-SW 算法的数据结构

该算法的数据结构包含 3 种,分别是项目矩阵(PM),临界频繁模式树 NCFP-Tree 和频繁项集表(FI-i_List).

1) 项目矩阵(PM):PM 为上三角矩阵,每行或每列存储数据库中项的集合,其中项目矩阵中元素 PM(i,j)的值表示 <Ii,Ij> 出现的次数.

2) 临界频繁模式树 NCFP-Tree,由根节点(用 null 标记)、临界频繁项集构成的前缀子树和临界频繁项头表三部分组成.

除了根节点之外的每个节点是由 5 个域组成的(itemname,support,parent,child,nextnode),其中 itemname 表示项名,支持度计数用 support 表示,指向父节点的指针用 parent_link 表示,指向孩子节点的指针用 child_link 表示;指向树中具有相同项名的下一个节点的指针用 nextnode_link 表示.

临界频繁项头表 ID_List 中每个元组包含 2 个域(It_id,link),其中 It_id 表示基本窗口中临界频繁 1-项集的项名,link 表示链头指针,指向 NCFP-Tree 中有相同项名的第一个节点,需降序排列每个基本窗口中项的支持度计数.

3) FI-i_List():一种数组结构,包括 k+2 个域(Item,F1,F2,…,Fk,F),其中 Item 表示项名,Fi 表示第 i 个基本窗口挖掘的临界频繁项集的支持度计数,F 表示滑动窗口中项集的支持度计数的和. 为了提高时间效率,将挖掘出来的临界频繁项集分开存储,临界频繁 i-项集存储在 FI-i_List 中,其中($1\leqslant i\leqslant n$). 该表起始状态为空,将挖掘的基本窗口中临界频繁项集存储在对应的表中,关键的步骤是当窗口滑动时,将挖掘的新本窗口中的项集直接覆盖旧基本窗口中对应的项集即可.

3.2 NCFP-Tree 的构造方法

它由根(用"NULL"标记)、临界频繁项集构成的前缀子树和临界频繁项头表三部分组成,创建 NCFP-Tree 的根结点,对 DS 中的每个事务进行如下处理:

1) 选择 Ti 中的临界频繁项,并将它们按临界频繁项头表顺序排列,第一次扫描项集,设排序后的临界频繁项集列表为[p|P],其中 p 是第一个项目,而 P 是剩余的项目列表.

2) 调用 insert_tree([p|P],T).

注意:函数 insert_tree([p|P],T) 执行如下:

在插入节点之前,扫描项目矩阵 PM,首先判断插入结点与祖先节点组成的 2-项集是否为临界频繁 2-项集,如果有一个祖先节点 M 与该节点组合成的 2-项集满足临界频繁 2-项集,则该节点可以作为 M 的孩子节点插入,如果不满足临界频繁 2-项集,那么一层一层的向根部搜索,直到有一节点 q,它和插入节点组成的二项集的支持度满足临界频繁项集的条件. 但如果一直到根节点 NULL 都没有节点与该节点能组合为临界频繁 2-项集,那么剪掉该节点. 如果能找到该节点,插入时需要进行下列操作:判断 T 是否有子女,如果 T 有子女 N 使 N.item=p,N 的计数就加 1,否则需要创建一个新的节点 N,将 item 设置为 p,sup 设置为 1,链接到它的父节点,并通过节点链 Node_link 链接到具有相同项名的节点上.

3) 如果 P 不为空,递归调用 insert_tree(P,p) .

3.3 MFI-SW 算法描述

1. 窗口初始阶段

输入: 数据流 DS,滑动窗口 W 的大小为 m,基本窗口大小为|w|,最小误差因子 ε

输出: 项目矩阵 PM,每个基本窗口对应的 NCFP-Tree

1) 滑动窗口中含 m 个基本窗口,基本窗口大小为|w|
2) for(i = 1,i < = m,i + +) {
3) 调用函数 Creat_PM 生成项目矩阵 PM
4) 调用算法 Creat_NCFP-Tree,生成 m 个 NCFP-Tree 树}

函数 Creat_PM 如下: //项目矩阵的构造

1) PM [i,j] = 0　　//初始化矩阵 PM,n 为项的个数
2) for each Ti in DS　　//扫描事务数据库,构建 PM 矩阵
3) for each ei in Ti
4) for each ej in Ti
5) if j > = i
6) PM [i,j] + +
7) for i = 1 to n　　// n 代表基本窗口的平均长度
8) { count = 0
9) for j > = i + 1 to n
10) if PM [i,j] < ε|w|
11) delete ei
12) // ei 是不频繁项,删除支持度不符合条件的项}

2. 窗口滑动阶段

输入: 新基本窗口中的事务数据流,最小误差因子 ε,滑动窗口 W 的大小为 m

输出: 更新的项目矩阵 PM 和新基本窗口的 NCFP-tree

1) 调用函数 Creat_PM 建立新基本窗口的项目矩阵
2) 调用 Creat_NCFP-Tree 生成新基本窗口的 NCFP-Tree 树
3) 调用函数 Update_FI-i_List 更新频繁项集表

对表都进行更新,函数 Update_FI-i_List 如下:

1) int column = t%m　　//t 表示第 t 个窗口
2) for each ej in FI-i_List{
　　//ej 表示在 NCFP-Tree 树中挖掘的临界频繁项集
3) Fej = Fej + num　　//num 是临界频繁项集 ej 的支持度计数
4) else
5) creat ej
6) Fej = num}

3. 频繁项集产生阶段

输入: 滑动窗口中每个基本窗口的 NCFP-Tree,最小支持度阈值 min_sup,滑动窗口 W 的大小 m

输出: 完全频繁项集

1) 按照临界频繁项头表的顺序取项目 ei
2) for each ei in ID_List{
3) 挖掘 NCFP-Tree 中所有包含 ei 的临界频繁项集,存在对应的 FI-i_List 中}
4) for each ei in FI-i_List{
5) if F_{ei} > = min_sup
6) 输出项集 ei 和它的支持度计数并存储在 FI_List 中;
7) else
8) 输出 ø
9) 输出完全频繁项集 FI_List}

3.4 MFI-SW 算法的执行过程

以表 1 所示的事务数据流为例介绍算法的每步执行过程,设滑动窗口的大小为 2(也就是说一个滑动窗口包含 2 个基本窗口) ,最小支持 min_sup = 0.4,ε = 0.1.

表 1　事务数据流

Table 1　Transaction data stream

Pan	TID	项集	预处理项集
W1	T1	b c d e f g	b c d f e g
	T2	b c d	b c d
	T3	a c h i	c a
	T4	b c d e f	b c d f e
	T5	b c d e	b c d e
	T6	a d f h	d a f
	T7	b c f g i	b c f g
	T8	a b c d f	b c d a f
	T9	a b e g	b a e g
	T10	a b	b a
W2	T11	a b d e f g	b f d a e g
	T12	b c d f	b f d c
	T13	a b c e f	b f a c e
	T14	b d f	b f d
	T15	b c f i	b f c
	T16	a d e g h	d a e g
	T17	b d	b d
	T18	a b e	b a e
	T19	a b c f	b f a c
	T20	b c d f g	b f d c g

1. 窗口初始阶段

数据流流入滑动窗口时,以基本窗口为单位,首先建立第一个基本窗口对应的项目矩阵 PM1,如表 2 所示,为了节省时间和空间,先把不频繁的 h 和 i 两项过滤掉. 然后根据项目矩阵 PM1 对临界频繁项头表进行降序排列,即 bcdafeg,进而将数据压缩存储在 NCFP-Tree 树中.

表 2　项目矩阵 PM1

Table 2　Project Matrix PM1

	a	b	c	d	e	f	g	h	i
a	5	3	2	2	1	2	1	2	1
b		8	6	5	4	4	3	0	1
c			7	5	3	4	2	1	2
d				6	3	4	1	1	0
e					4	2	2	0	0
f						5	2	1	1
g							3	0	1
h								2	1
i									2

扫描表 1 中预处理数据,建立对应的 NCFP-Tree1,每插入一个节点,都要扫描项目矩阵 PM1,查看该节点能否与祖先节点构成临界频繁 2-项集,如果能,则该节点插入在该祖先的孩子节点上,如果没有,继续向根部搜索,假设到达根节点 NULL 都不能找到与该节点组成临界频繁 2-项集的祖先节点,那么剪掉该结点. 如图 1 所示是第一个基本窗口对应的 NCFP-Tree1.

利用同样的方法,建立第二个基本窗口的项目矩阵 PM2

和它对应的 NCFP-Tree2.

2. 窗口滑动阶段

窗口滑动以后,采用最新的基本窗口的项目矩阵覆盖最旧的基本窗口对应的项目矩阵,释放最旧基本窗口对应的

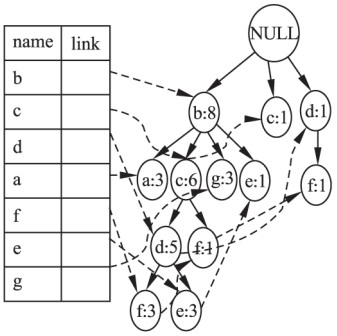

图 1 第一个基本窗口的 NCFP – Tree1

Fig. 1 First basic window NCFP – Tree1

NCFP-Tree. 同时用同样的方法建立最新的基本窗口对应的 NCFP-Tree,频繁项集表 FI-i_List 也要随之更新,这里采用新基本窗口挖掘出来的频繁项集直接覆盖最旧的基本窗口挖掘的对应项集.

3. 频繁项集产生阶段

根据性质 3 可知,项目矩阵 PM 中包含频繁 1-项集和频繁 2 项集的全部信息,因此可以通过扫描事务矩阵 PM1 得到临界频繁 1-项集 FI-1_List = { a: 5,b: 8,c: 7,d: 6,e: 4,f: 5,g: 3} 和临界频繁 2-项集 FI-2_List = { ab: 3,bc: 6,bd: 5,be: 4,bf: 4,bg: 3,cd: 5,ce: 3,cf: 4,de: 3,df: 4} ,通过利用改进的挖掘临界频繁项集技术挖掘每个基本窗口对应的 NCFP-Tree 的所有临界频繁项集,得到 FI-3_List = { bce: 3,bde: 3,cde: 3,bcf: 4,bdf: 3,cdf: 3,bcd: 5} 和临界频繁 4-项集 FI-4_List = { bcdf: 3,bcde: 3} . 用同样的方法可以挖掘第二个基本窗口,可得临界频繁 1-项集 FI-1_List = { a: 5,b: 9,c: 5,d: 6,e: 4,f: 7,g: 3} ,临界频繁 2-项集为 FI-2_List = { ab: 4,ae: 4,af: 3,bc: 5,bd: 5,be: 3,bf: 7,cf: 5,df: 4,dg: 3} ,临界频繁 3-项集 FI-3_List = { bdf: 4,abf: 3,bcf: 5} ,最终输出 FI-i_List 中的完全频繁项集的挖掘结果 FI_List = { a: 10,b: 17,c: 12,d: 12,e: 8,f: 12,bf: 11,cf: 8,df: 8,bd: 10,bc: 11,bcf: 9} .

4 MFI-SW 算法分析

实验环境是操作系统为 64 位 window7,3.20GHzCPU 和安装内存是 8.00GB,算法实现语言是 C 语言,实验数据流的数据集[15] 采用的是 IBM data generator 生成的模拟数据. 分别有稀疏集 T7I4D100K, T10I4D100K 和稠密集 T40I10D100K,其中 T 表示的是事务的平均长度,I 表示的是项集的平均最大长度,D 表示的是总的事务项目. 滑动窗口包含的基本窗口数为 3,每个基本窗口包含 10000 条事务,设支持度为 ζ,取允许误差因子 ε 固定为 0.1ζ(ζ 为最小支持度阈值) . 以下就算法的响应时间和查全率、查准率对 MFI-SW 算法和 pWin 算法、DSFPM 算法进行实验对比分析.

4.1 算法响应时间

如图 2 所示,新算法 MFI-SW 和 DSFPM 算法和 pWin 算法在三种不同的数据集上的处理时间,最小支持度阈值为 0.5%,实验结果标明 MFI-SW 算法随着平均长度的增加,时间消耗越大,在稠密数据集中的运行时间远高于稀疏集,主要是因为稠密数据集中的事务的平均长度和项集的平均最大长度比较长,在滑动窗口中,更新数据和挖掘频繁项集的过程比较消耗时间,所以该算法更合适稀疏数据集.

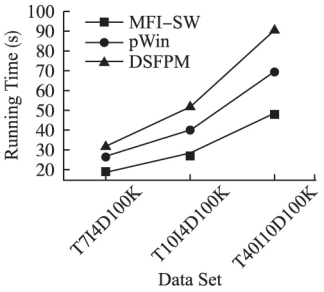

图 2 不同数据集上的运行时间比较

Fig. 2 Runtime comparison on different datasets

如图 3 所示,在稀疏集 T10I4D100K 下对 MFI-SW 算法与 DSFPM 算法、pWin 算法在不同最小支持度阈值下的处理

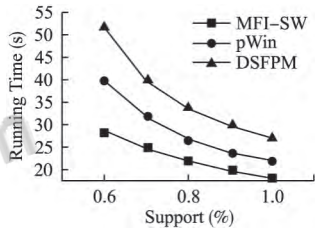

图 3 不同支持度的运行时间比较

Fig. 3 Comparison of run time with different support

时间进行了比较. 实验结果表明在最小支持度阈值从 0.5% 到 1% 变化的过程中,三个算法的时间开销的变化趋势都是随着支持度阈值的增加,每个基本窗口挖掘出来的临界频繁项集数目减少,因此时间开销也随着减小,但是 MFI-SW 算法的时间性能略高于另外两个算法.

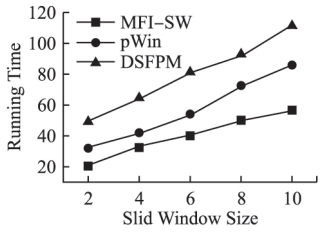

图 4 不同的窗口大小、算法运行时间的比较

Fig. 4 Different window sizes、comparison of algorithm run time

如图 4 所示,为了显示窗口大小对所有算法的时间使用的影响,在稀疏集 T10I4D1000K 中测试不同窗口大小的情况下,MFI-SW 算法与 DSFPM 算法、pWin 算法的时间开销,取最小支持度阈值 ζ = 0.5%. 实验结果表明滑动窗口越大,算法的时间开销越大,MFI-SW 算法随着滑动窗口的增加,时间开销变化较稳定,其他两个算法 pWin 和 DSFPM 在滑动窗口增加的情况下,因为它们都需要频繁的更新树结构,所以时间

开销变化较大.

4.2　算法的查准率和查全率

在算法中最大误差是一个非常重要的剪枝参数,查准率指的是算法挖掘出来的实际频繁项集的数量与算法挖掘出来的输出数量的百分比,而查全率则指的是算法挖掘出来的实际频繁项集的数量值与真实数量的百分比. 一般情况下,在数据集中挖掘频繁项集以最大的误差值作为剪枝参数,在挖掘过程中更新树结构时对算法的查全率和查准率有很大的影响,图 5 和图 6 测试了 IBM 合成数据流[16]中不同误差值对算法的查准率和查全率的影响. 假定误差值分别为 0.0002,0.0008,0.0014 和 0.002 时,观察测试结果.

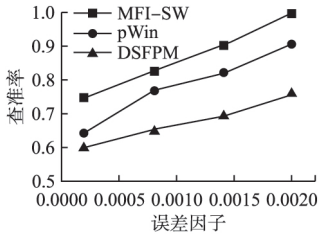

图 5　不同的误差因子、算法查准率的比较

Fig. 5　Different error factors、comparison of algorithm precision

从图 5 和图 6 中不难看出,在误差值逐渐变大的过程中,MFI-SW 算法的查准率和查全率高于 pWin 和 DSFPM 两个算

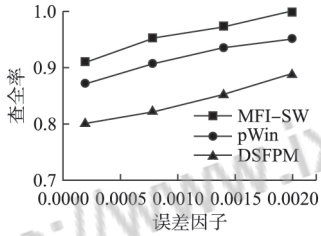

图 6　不同的误差因子、算法查全率的比较

Fig. 6　Different error factors、comparison of algorithm recall rate

法. 因为当误差值逐渐增大的过程中,挖掘出来的频繁项集的数量变少,于是用户指定的误差值越大,挖掘出来的频繁项集数量就越少,挖掘多条数据流中的频繁项集的准确率和查全率就会越高.

5　结束语

本文提出了滑动窗口中 FP-Tree 的频繁项集挖掘算法 MFI-SW,该算法采用上三角矩阵的存储方式,每个基本窗口建立一个 NCFP-Tree 来压缩存储数据信息,在相同节点个数的情况下,NCFP-Tree 较 FP-Tree 的深度低,分支少,因为 NCFP-Tree 中的父子节点保留临界频繁 2-项集的全部信息,窗口滑动时,将旧基本窗口的 NCFP-Tree 释放,建立新基本窗口的 NCFP-Tree,整个过程中保证每棵树都按基本窗口中的 1-项集的支持度计数的降序排列,有效的提高了 MFI-SW 算法的压缩效率,并为频繁项集的输出带来方便,通过与经典的算法比较证明,MFI-SW 算法具有良好的时间性能. 在此基础上还需研究以二项集为根,建立树结构,这样使树结构更矮,挖掘效率更高.

References:

[1] Ao Fu-jiang, Yan Yue-jin, Huang Jian, et al. Design of data stream frequent pattern mining algorithm [J]. Computer Science, 2008, 35(3): 1-5.

[2] Li Guo-hui, Chen Hui. Frequent pattern mining in arbitrary sliding time window of data flow [J]. Journal of Software, 2008, 19(10): 2585-2596.

[3] Tu Li, Chen Ling, Bao Fang. Mining frequent items in a stream based on sliding window [J]. Journal of Chinese Computer Systems, 2012, 33(5): 940-949.

[4] Tanbeer S K, Ahmed C F, Jeong B S, et al. Sliding window-based frequent pattern mining over data streams [J]. Information Sciences, 2009, 179(22): 3843-3865.

[5] Lee G, Yun U, Ryang H, et al. Approximate maximal frequent pattern mining with weight conditions and error tolerance [J]. International Journal of Pattern Recognition and Artificial Intelligence, 2016, 30(6): 1650012-1650054.

[6] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation [C]. ACM Sigmod Record, ACM, 2000, 29(2): 1-12.

[7] Manku G S, Motwani R. Approximate frequency counts over data streams [C]. Proceedings of the 28th International Conference on Very Large Data Bases, VLDB Endowment, 2002: 346-357.

[8] Leung C K S, Khan Q I. DSTree: a tree structure for the mining of frequent sets from data streams [C]. Data Mining, ICDM'06, Sixth International Conference on. IEEE, 2006: 928-932.

[9] Giannella C, Han J, Pei J, et al. Mining frequent patterns in data streams at multiple time granularities [C]. Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT, 2003: 191-212.

[10] Lee G, Yun U, Ryu K H. Sliding window based weighted maximal frequent pattern mining over data streams [J]. Expert Systems with Applications, 2014, 41(2): 694-708.

[11] Liu Xue-jun, Xu Hong-bing, Dong Yi-sheng, et al. Mining data flow closed frequent patterns based on sliding window [J]. Journal of Computer Research and Development, 2006, 43(10): 1738-1743.

[12] Li Guo-hui, Yang Bing, Hu Dun, et al. Frequent patterns of data flow in mining sliding windows [J]. Journal of Chinese Computer System, 2008, 29(8): 1491-1497.

[13] Deypir M, Sadreddini M H, Tarahomi M. An efficient sliding window based algorithm for adaptive frequent itemset mining over data streams [J]. J. Inf. Sci. Eng., 2013, 29(5): 1001-1020.

[14] Aggarwal, Charu C, Jiawei Han, et al. Frequent pattern mining [M]. Springer, 2014.

[15] Lee G, Yun U, Ryu K H. Sliding window based weighted maximal frequent pattern mining over data streams [J]. Expert Systems with Applications, 2014, 41(2): 694-708.

[16] Wang Xin, Liu Fang-ai. An improved multi-stream collaborative frequent itemsets mining algorithm [J]. Journal of Computer Applications, 2016, 36(7): 1988-1992.

附中文参考文献:

[1] 敖富江,颜跃进,黄　健,等. 数据流频繁模式挖掘算法设计[J]. 计算机科学,2008,35(3):1-5.

[2] 李国徽,陈　辉. 挖掘数据流任意滑动时间窗口内频繁模式[J]. 软件学报,2008,19(10):2585-2596.

[3] 屠　莉,陈　峻,包　芳. 挖掘滑动窗口中的数据流频繁项算法[J]. 小型微型计算机系统,2012,33(5):940-949.

[11] 刘学军,徐宏炳,董逸生,等. 基于滑动窗口的数据流闭合频繁模式的挖掘[J]. 计算机研究与发展,2006,43(10):1738-1743.

[12] 李国徽,杨　兵,胡　惇,等. 挖掘滑动窗口中的数据流频繁模式[J]. 小型微型计算机系统,2008,29(8):1491-1497.

[16] 王　鑫,刘方爱. 改进的多数据流协同频繁项集挖掘算法[J]. 计算机应用,2016,36(7):1988-1992.