可变滑动窗口在数据流频繁模式挖掘上的应用①

苏 勇，范玉玲

(江苏科技大学 计算机科学与工程学院，镇江 212003)

摘 要：为了更好的挖掘数据流，对传统的滑动窗口机制进行改进，提出一种大小可变的滑动窗口机制的数据流频繁集挖掘算法 DS-stream 算法。该算法能够根据数据流的数据分布变化自适应调整窗口大小，节省了没必要的空间与时间消耗。算法采用一种分区窗口机制，结合基本窗口和时间窗口，同时考虑数据流的海量特性和时变特性，利用前缀树的概要数据结构。实验结果表明, DS-stream 算法在挖掘数据流频繁集上有很好的时间与空间效率。

关键词：数据流；滑动窗口；频繁项；闭合项集；DS-stream

Data stream of Closed Pattern Mining Based on Variable Slide Window

SU Yong, FAN Yu-Ling

(Computer Science and Engineer Institute, Jiangsu University of Science and Technology, Zhenjiang 212003, China)

Abstract: To mine data stream efficiency, a new slide window, combines basic window and time window, adopts a scheme of window based on sector. It could apply to the real data streams and change the size of the windows, and save time and space that no necessary to use. And the algorithm based on frequent itemsets mining-DS-stream could self-adaptively adjust the size of time windows, considering the large number and chronotropic character of the data stream. In the algorithm, transaction list group is adopted as synopsis data structure. The experiments results in Eclipse indicate that the DS-stream algorithm is more effective in term of temporal and spatial performance.

Keywords: data stream; slide window; frequent itemsets; closed items; DS-stream

频繁模式挖掘是数据挖掘的精髓，并且在上个世纪被广泛研究。近年来，挖掘数据流上的频繁模式引起了大量研究者兴趣。GS.Manku 和 R.Motwani 提出了两个频繁模式挖掘算法 Sticky Sampling 和 Lossy Counting。算法利用数据分段的思想，在设定支持度闭值和错误因子下，给出了求解单个频繁项的有效算法[2]。继 Teng 等人利用滑动窗口模型来挖掘数据流中的最近频繁项集，Yun Chi 提出基于滑动窗口的闭合频繁模式挖掘算法 Moment，利用 FP-tree 结构和 CET 结构进行数据流的处理和更新，在 CET 结构上根据闭合频繁模式的概念对闭合频繁模式进行挖掘[6]。

数据流到达的速率并不是固定不变的，相同历史时间段内的事务的数目也不一定相同，有时甚至相差巨大。在前人的研究滑动窗口的闭合频繁模式挖掘的基础上，考虑数据流的海量特性和时变特性，提出了可调控滑动窗口的数据流频繁集挖掘算法 DS-stream，能在可变的窗口上挖掘频繁项集。

1 滑动窗口

在实际的数据流挖掘中，数据流并不是以相同速率到来的，为了提高数据流挖掘的精确性与效率性，这里设计一种可以根据数据流的速度来动态改变窗口大小的机制。将滑动窗口分为多个分区窗口[1]，而每个分区窗口由固定事务窗口 FTW 和可调时间窗口 TW 组成。滑动窗口为 $SW=\{(FTW_1,TW_1),(FTW_2,TW_2),\ldots,(FTW_i,TW_i),\ldots,(FTW_n,TW_n)\}$，$(FTW_n,TW_n)$是最近到达滑动窗口的数据。固定事务窗口的长度设为 w，数据以正常流速充满固定事务窗口即窗口长度达到 w 的时

① 收稿时间:2010-10-11;收到修改稿时间:2010-11-30

间为 τ，数据流流入滑动窗口的初始时刻为 $t_0 = t_1^1 = 0$。

$FTW_i=\{(T_1^i,t_1^i),(T_2^i,t_2^i),...,(T_j^i,t_j^i),...,(T_w^i,t_w^i)\}$

$TW_i=\{(T_{w+1}^i,t_{w+1}^i),(T_{w+2}^i,t_{w+2}^i),...,(T_{w+j}^i,t_{w+j}^i),...,(T_{w+ai}^i,t_{w+ai}^i)\}$，

$i\in(1,k)$，$j\in(1,w)$。其中 T_i 表示到达的事务，t_i 表示事务 T_i 到达时对应的时刻，$a_i\geq 0$，$m\geq 1$。对于第一个分区窗口，若 $t_w^1<\tau$，那么固定事务窗口和时间窗口可表示为同一形式：

$FTW=\{(T_1^1,t_1^1),(T_2^1,t_2^1),...,(T_j^1,t_j^1),...,(T_w^1,t_w^1)\}$，

$TW=\{(T_{w+1}^1,t_{w+1}^1),(T_{w+2}^1,t_{w+2}^1),...,(T_{w+j}^1,t_{w+j}^1),...,(T_{w+ai}^1,t_{w+ai}^1)\}$；

若 $t_w^1\geq\tau$，那么固定事务窗口和时间窗口可表示为同一形式：

$FTW=\{(T_1^1,t_1^1),(T_2^1,t_2^1),...,(T_j^1,t_j^1),...,(T_w^1,\tau)\}$；如果 FTW 的长度远小于 w，则该窗口内数据做忽略处理，不作挖掘。

2 DS-Stream 算法

2.1 概要数据结构

DS-Stream 算法：以滑动窗口中的分区窗口为计算单位，利用已有的频繁闭合模式算法计算每个分区窗口的临界频繁闭合项集，连同其子集动态地压缩存储在 DS-tree 中，进行增量更新。构建一个模式树对挖掘的频繁闭合项集进行子集检查。DS-tree 是基于 FP-tree[3]改进的一棵前缀压缩树，结构特点如下：

（1）它使用两层散列索引结构 item-support table 对 Ktree 上支持度满足最小支持度要求的项建立索引，索引中的项按字典排序，临界频繁闭合项集及其子集存放在对应的树的分枝中；

（2）根结点之外的每一个结点由 4 个域组成（itemname，sup，closetag，isupd），itemname 代表项目名，sup 表示项目的支持度；closetag 反映了从该结点到根结点的直接子结点的路径所构成的临界频繁项集是否是临界频繁闭合项集，isupd 表示当前分区窗口中该结点是否被更新；

（3） 需要存储临界频繁闭合项集的各个子集，树中任何结点的支持数都不小于其子结点的支持数。

2.2 构建 DS-tree 算法

输入：初始滑动窗口中的分区窗口 SWi，$1\leq i\leq k$，允许误差 ε，固定窗口大小 w，时间 τ。

输出：由初始滑动窗口构造的 DS-tree。

FUNCTION BuildDS-tree (SWi , τ ,ε)

1 生成一棵初始的 DS-tree ，树中只有一个根结点；

2 For 每一个分区窗口 SWi{

3 If (j<=w) {

4 While （ $t_j^i\leq\tau$ ）{

5 将项目存储在一个 K 叉树中，记录其支持度 sup，窗口内项目数记为 count，初始为 0，count++，j++；}

6 fi = j；}

7 If（count<0.001w）

8 {不对本窗口挖掘，计算下一个分区窗口；}

9 以 ε 为支持度，对 Ktree 调用频繁闭合模式挖掘算法生成该窗口内的临界频繁闭合项集 a；

10 将 a 中的每项插入到 DS-tree 中，closetag 标记为 1，支持度为 sup，isupd 初始为 0，isupd=1；

11 项集 a 的地址保存到散列表 item-sup table 中，并按项目名的字典顺序插入，同时保存支持度；

12 删除 Ktree；}

2.3 DS-tree 更新算法

输入：一个新的分区窗口 SW_{new}；

输出：一棵更新的 DS-tree

FUNCTION UpdateDS (SW_{new})

1 For SWnew 中的每个事务项{

2 If (j<=w) {

3 While （ $t_j^i\leq\tau$ ）{

4 将项目存储在一个 Ktree 中，记录其支持度 sup，窗口内项目数记为 count，初始为 0，count++，j++；}}

5 If（count<0.001w）

6 {不对本窗口挖掘，计算下一个分区窗口；}

7 以 ε 为支持度，对 Ktree 调用频繁闭合模式挖掘算法生成该窗口内的临界频繁闭合项集 a，以及不频繁项集 b；

8 For (a 中的每个闭合项集){

9 If（a 中的项在 item-sup table 中）

10 {更新 sup 值，isupd=1 ；}

11 Else {插入到 DS-tree 中，isupd=0；}}

12 For (a 中的每个闭合项集){

13 If（b 中的项在 item-sup table 中）

14 {更新模式树中该项的信息，isupd=1 ；}

15 Else　{插入到 DS-tree 中，isupd=0；}}

3　实验分析

3.1 实验结果与环境

实验环境为一台 Windows XP 操作系统的 Pc 机，AMD 64 位双核处理器、1G 内存，并且程序在 VC++6.0 环境中运行。实验采用 IBM 模拟数据产生器(http//www.Alma-den.ibm.com)产生的数据集。1M 代表 100 万条事务。实验中，算法所需要的存储空间为算法维护数据流上频繁模式信息所需要的存储空间，而算法的平均处理时间包括该模式树的维护时间、模式树剪枝时间以及从模式树上输出模式所需要的时间。

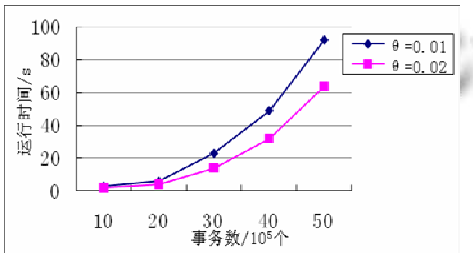

图 1　DS-stream 的时间效率

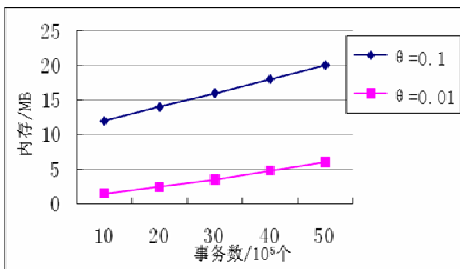

图 2　DS-stream 算法随事务数变化的空间效率

3.2 实验结果分析

实验通过分析算法在处理不同流速的数据运行所需要的最大内存空间与平均处理时间来学习算法的基本性能。由实验可知：挖掘相同的事务，最小支持度越小，运行时间越长。对于同一支持度，越多的事务数则含有越多的频繁项集，运行时间越长。这是因为最小支持度越低，挖掘相同事务数会产生越多的频繁集。实验结果表明，采用变尺度滑动窗口 DS-stream 算法在时间空间性能上有所提高。由此可以推出在挖掘大量的事务数时或者在最小支持度较小时，算法的效率有显著提高。

4　结语

目前大部分基于滑动窗口机制的数据流挖掘算法都假设滑动窗口的大小不变，然后进行增量式分析。这些算法在人工数据流上具有较好的效率，但在真实数据流上往往效率较低。同时考虑数据流的海量特性和时变特性，能在可变的窗口上挖掘频繁项集。算法采用近似挖掘技术[3]，一次扫描数据快速得到频繁项集。实验结果表明了 DS-stream 算法在性能上的效率。该算法尤其适用于流速显著变化的数据流频繁集挖掘应用领域。

参考文献

1 李建中,张冬冬.滑动窗口的动态的调整算法.软件学报,2004,15(12):1800−1814.

2 Manku GS, Motwani R. Approximate frequency counts over data streams. Proc. of the International Conference on Very Large Data Bases. Hong Kong, China, 2002:346−357.

3 Arasu A, Manku GS. Approximate counts and quantities over sliding windows. Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Paris, France: ACM Press, 2004. 286−296.

4 Chi Y, Wang HX, Philips R, Muntz R. Moment: Maintaining closed frequent itemsets over a stream sliding window. Proc. of the Fourth IEEE International Conference on Data Mining. 2004: 59−66.

5 Papadimitriou S, Sun J, Faloutsos C. Streaming pattern discovery in multiple time-series. Proc. of the 31st VLDB Conf. 2005: 697−708.