RESEARCH ARTICLE | MAY 08 2017

A variable size sliding window based frequent itemsets mining algorithm in data stream

Haiqing Li; Lang Wang

AIP Conference Proceedings 1839, 020146 (2017)
https://doi.org/10.1063/1.4982511

A Variable Size Sliding Window Based Frequent Itemsets Mining Algorithm in Data Stream

Haiqing Li[a)], Lang Wang[b)]

[1]*College of College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China.*

[a)]Corresponding author: haili042@foxmail.com
[b)]qjzxwanglang@foxmail.com

Abstract. Due to the unpredictability and the concept drift character of the data stream, the traditional sliding window is difficult to adapt to frequent itemsets mining in data stream. A new variable sliding window based VSW-SCPS algorithm is proposed. The algorithm maintains a tree structure of SCPS-tree in memory, which is the storage structure of sliding window. When data flowing in, the SCPS-tree will adjust dynamically, the window size will be adjusted by the detection of concept drift according to the result of FP-growth mining algorithm. Experimental results show that the proposed algorithm has good time efficiency.

Keywords: data stream; concept drift; frequent itemsets; variable sliding window.

INTRODUCTION

Data stream is a set of real-time continuous data sequences. As an important step in the acquisition of association rules, frequent itemsets mining is attracting widely attention and research, and with the increasing demand of real-time analysis of large data, it is gradually become a hotspot in data mining. Unlike traditional static data sets, the streaming data are infinite, fast changing, massive and unpredictable[1]. So the frequent itemsets must be mined in a limited memory space. In addition, the most typical feature of the data stream is the concept drift, that is, the underlying concept of potential fundamental changes. For example, customer online shopping preferences will change with personal interests or other factors. This phenomenon of implicit target concept transfer, or even radical change, which occurs due to context changes in the data stream is called concept drift[2].

RELATED WORKS

At present, most of data stream frequent itemsets mining algorithm are based on the window model. Such as the lossy counting algorithm[3] using the landmark window model, the FP-stream algorithm[4] using the damped window model, the Moment algorithm[5] and the MSW algorithm[6] are based on the sliding window model. As the sliding window model is simple and easy to understand, it has been widely applied. Most of the sliding window-based algorithms are based on a fixed size window, which will delete the oldest batch of data when a new batch of data arrives, to maintain a fixed size. However, the window size determination requires a certain amount of prior knowledge, if it is set too large, it may cause too much useless patterns, which will affect the efficiency of mining; if it is set too small, it may lead to concept drift and affect mining accuracy. Therefore, for mining streaming data, fixed sliding window is difficult to adapt to the data stream implied concept drift characteristics. Some variable sliding window based algorithms are proposed, such as VSW algorithm[7] and MFW algorithm[8]. VSW algorithm proposed a variable sliding window model, where the window size measure is actively adjusted based on the detection of concept drift within the data stream. The window grows as the concept is fixed and gets smaller when

Materials Science, Energy Technology, and Power Engineering I
AIP Conf. Proc. 1839, 020146-1–020146-6; doi: 10.1063/1.4982511
Published by AIP Publishing. 978-0-7354-1509-6/$30.00

variation in concept captured. The algorithm effectively captures concept change, resizes the window size and acquires itself to new concept by scanning once the data stream. However, as it is use the Eclat algorithm[9] for mining, it is not efficient when the data volume is large.

Most of the algorithm are extended version of the well-known algorithm of FP-Growth[10] which is a tree based method without candidate generation proposed for mining in static databases. such as DSTree[11]. In this algorithm, transactions within a window are divided into a number of batches (or panes) and the information about every batch is maintained in a prefix tree. Nodes of the prefix tree show items in the transactions which are sorted in a canonical order. Transactions are inserted to and removed from the tree in a batch by batch manner. Frequent itemsets are mined using the FP-Growth method when the user requests them. Another recently proposed prefix tree based algorithm is the CPS-Tree[12] which is superior to the DSTree. This method is similar to the DSTree algorithm but dynamically reconstructs the prefix tree to maintain support descending order of nodes. Tree reconstruction is performed to reduce the amount of memory usage for the tree structure and to enhance the mining time.

THE VSW-SCPS ALGORITHM

In this paper, a variable Sliding Window Based Simple Compact Pattern Stream (VSW-SCPS) is proposed, which uses a new SCPS-tree tree structure to store variable sliding window. When each pane of data arrives, the transactions incrementally inserted into the SCPS-tree, and be dynamically adjusted through the BSM (Branch Sorting Method) strategy[13]. Then, detect the concept drift according to the mining results of the current window. If the concept drift is detected, the expired data are quickly deleted by using the support count before the checkpoint and the support count after the checkpoint which are recorded in the tail node.

Definitions

Let $I=\{I_1,I_2,I_3,\ldots,I_m\}$ be a set of items. For any transaction $T_i\,(i=1,2,\ldots,n)$, which is a subset of I. Let $DS=\{T_1,T_2,T_3,...,T_n,...\}$ be a data stream, wherein $T_j\,(j=1,2,3,\ldots,n,\ldots)$ represents the jth incoming transaction, and j can be called the *tid* of transaction. Let Itemset $X\subseteq I$. The support count SC of X is defined as the number of received transactions of the stream that contain X. For a given support threshold $minSup\,(0<minSup\le 1)$, X is frequent itemset if $SC\ge minSup\times|D|$, wherein $|D|$ is the number of transactions received. Sliding window SW over data stream DS contains $|SW|$ latest transactions in the stream, where $|SW|$ is the size of the window. When the SW slide forward, it insert a pane of transactions and remove oldest pane of transactions. A pane contains a fixed given number of transactions. Let $T=\{I_1,I_2,I_3,...,I_n\}$ be a transactions in the descending order of support, then the I_n is tail item. If this ordered transaction is inserted into an SCPS-tree, then the node represented by I_n is called the tail node. Non-tail nodes are called general nodes.

Insertion of SCPS-tree

The SCPS-tree is a tree-like data structure similar to the FP-tree. Let p be a non-root node in the SCPS-tree. If p is a general node, it contains 5 fields: $p.item$ is the item name, $p.count$ records the support count of $p.item$, $p.next$ points to the next node of the same item name, $p.parent$ records the parent node, and the $p.children$ is the head pointer of children nodes. If p is a tail node, it contains 2 additional fields: $p.preCount$ records the support count before the checkpoint, $p.curCount$ records the support count after the checkpoint. The initial value of $p.preCount$ and $p.curCount$ is 0. The SCPS-tree also contains a *HeadTable* structure that records the total support count for each item in the tree and is sorted in descending order. Starting from the empty tree, each incoming pane of data can be inserted into the SCPS-tree by algorithm 1:

ALGORITHM 1 SCPS-tree insertion
INPUT: SCPS-tree *Tree*, checkpoint *CP*, a pane of transactions *Pane*
OUTPUT: SCPS-tree *Tree*
(1) Copy = *Tree.HeadTable*; // copy the head table

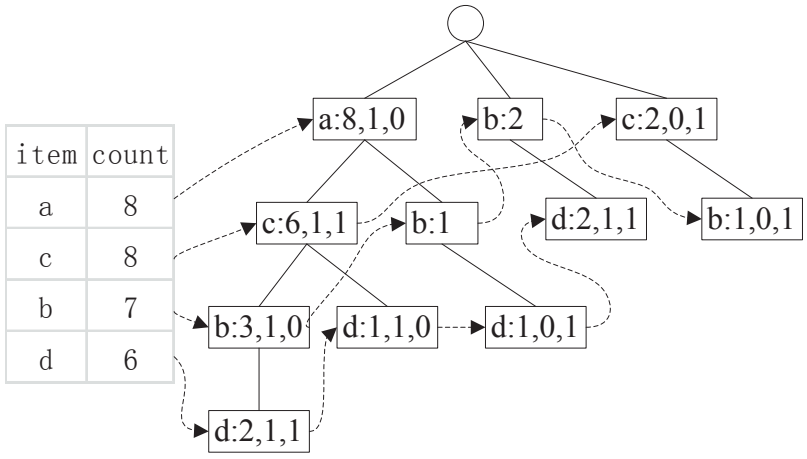
(2) For each transaction T in *Pane*
(3) T = Sort(Copy, T); // sort the transaction in frequent descending order according to the copy of the *Tree.HeadTable*
(4) For each item I in T
(5) $I.count$ ++;
(6) If I is tail item
(7) If $T.tid \le CP$ // if the current transaction is before the checkpoint
(8) $I.preCount$ ++;
(9) Else // if the current transaction is after the checkpoint
(10) $I.curCount$ ++;
(11) End If
(12) End If
(13) Update(I , *Tree.HeadTable*); // add the item I into *Tree.HeadTable*
(14) add(I); // insert the item I into the *Tree*
(15) End For
(16) End For
(17) BSM(*Tree.Root* , *Tree.HeadTable*); // use the latest HeadTable to reconstruct the SCPS-tree with the BSM strategy

Fig.1(b) shows an example of an SCPS-tree created according to the data stream shown in Fig.1(a), where the size of the sliding window is 2 panes of data, each of which contains 3 transactions. In fig.1(b), the nodes that contain 3 values are the tail nodes, which correspond to the values of *count* , *preCount* and *curCount* , respectively; other nodes are general nodes which only record value of *count* .

pane	tid	transaction
1	1	acd
	2	bd
	3	abcd
2	4	abc
	5	a
	6	ac
3	7	abd
	8	abcd
	9	ac
4	10	bd
	11	c
	12	bc

(a) Data stream instance

item	count
a	8
c	8
b	7
d	6



(b) SCPS-tree

FIGURE 1. SCPS-tree instance

Concept drift detection

In the frequent itemset mining process, the concept is defined as the set of frequent itemsets. When the data continues to flow, the corresponding concept within the sliding window will continue to change; when the window is sliding, the old concept is replaced by the new concept. The difference between the old and new concepts, that is, the number of frequent itemsets in the old and new windows, is called the rate of change. If the rate of change exceeds the user given a threshold, it means that the concept of drift has occurred. Given two time points named T and T', and $T' > T$. let F_T and $F_{T'}$ be the frequent itemsets at time T and T'. Then $F_T^+(T') = F_{T'} - F_T$ is the set of new coming frequent itemsets after T until T', and $F_T^-(T') = F_T - F_{T'}$ is the infrequent itemsets at T' which was

frequent at T . Let $FChange_T(T')$ be the change ratio between T and T' , it is satisfy with the following formula[14]:

$$FChange_T(T') = \frac{|F_T^+(T')| + |F_T^-(T')|}{|F_T| + |F_T^+(T')|} \tag{1}$$

Where $|F_T|$ is the number of itemsets in set F_T , the value of $FChange_T(T')$ is between 0 and 1.

Update of sliding window

During the mining procedure, the initial window size *initSize* is given by user, next time the window size is automatically adjusted based on the concept drift detection. The boundary point of the sliding window is determined by the checkpoint, and $CP_{init} = initSize$ .As shown in Fig.2, assume that the data is inserted from right to left into a nearest sliding window W . If the concept drift is detected, the data before checkpoint cp is removed and the new window ω is generated, and cp moves to the point when the concept drift occurs. If the concept drift is not detected, the window size will continually increase as the data arrive.

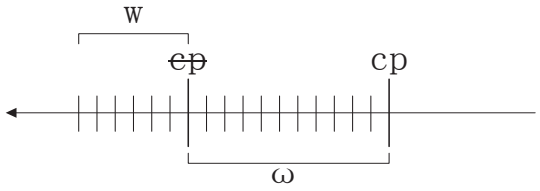


FIGURE 2. Schematic diagram of a variable sliding window model

Since the algorithm uses the SCPS-tree to store the variable sliding window, when window slide, use the Algorithm 1 to insert each pane of transactions into SCPS-tree, then use the FP-growth algorithm to mine frequent itemsets, and detect concept drift by the results. If the concept drift occurs, use the Algorithm 2 to deleted all expired information before checkpoints; otherwise, the SCPS-tree continually growth up. Concept drift may occur after one or more panes of transactions are inserted. Fig.3 is a result of shrinking according to the example of Fig.1.

ALGORITHM 2 SCPS-tree expired data deletion algorithm
INPUT: SCPS-tree *Tree*
OUTPUT: SCPS-tree *Tree*

(1) For each I in *Tree.HeadTable* , starting with the least supported item
(2) For each brother node N of item I
(3) If N is tail node
(4) For each node P between N and the root node
(5) $P.count = P.count - N.preCount$; // update $P.count$
(6) If $P.count == 0$ // if the current node support number is 0
(7) remove(N); // remove the node
(8) End If
(9) End For
(10) $N.preCount = N.curCount$; // set the value of $N.curCount$ to $N.preCount$
(11) $N.curCount = 0$; // reset $N.curCount$
(12) End If
(13) End For
(14) End For
(15) BSM(*Tree.Root* , *Tree.HeadTable*); // reconstructing SCPS tree with BSM strategy

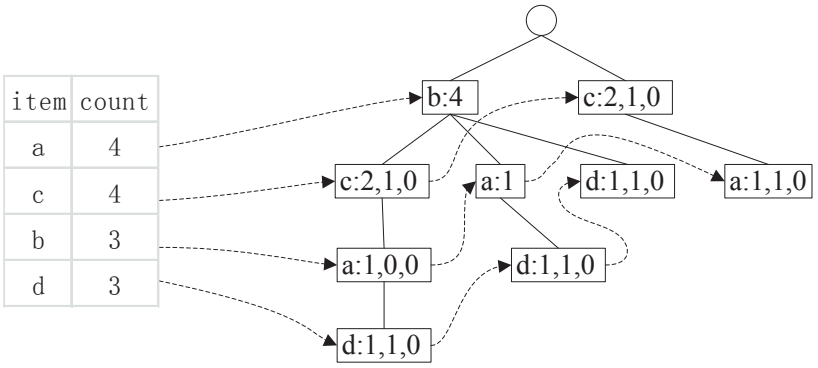
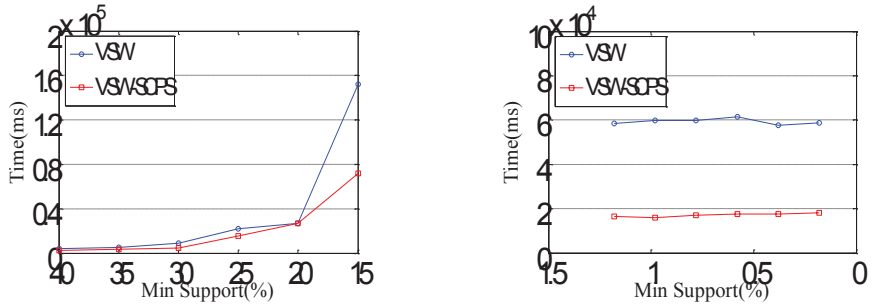


FIGURE 3. SCPS-tree after deleting expired data

EXPERIMENTAL RESULTS

We have performed a set of experiment to evaluate the proposed algorithm. All programs are written in Java SE 7.0, and run in Windows 7 operating system on a 3.20GHz CPU machine with 4GB of memory. We use the mushroom dataset and the T10I4D100K dataset, where the former is a dense dataset containing 8124 transactions, while the latter is a sparse dataset containing 100,000 transactions.

Fig.4 shows the mining time for VSW algorithm and VSW-SCPS algorithm on mushroom and T10I4D100K datasets. Results indicate that the smaller the minimum support, the more frequent itemsets are generated, and the more mining time. It can be seen from Fig.4(a) that under the mushroom data set, the initial size of the sliding window is 2 panes, the pane size is 1000, the min support are (40%, 35%, 30%, 25%, 20%, 15%) and the change rate threshold is 0.2, the time cost of VSW-SCPS algorithm is smaller than that of VSW algorithm, and the gap becomes larger as the min support decreases. In Fig.4(b), the initial size of the sliding window is 3 panes, the size of the pane is 5000, the support is (1.2%, 1.0%, 0.8%, 0.6%, 0.4%, 0.2%), and the change rate threshold is 0.015 and the dataset is the T10I4D100K, The mining time of VSW-SCPS algorithm is obviously less than VSW algorithm. Experimental results show that the mining efficiency of VSW-SCPS algorithm is better than VSW algorithm.



(a) mushroom (InitSize=2,pane=1000,th=0.2)

(b) T10I4D100K (InitSize=3,pane=5000,th=0.015)

FIGURE 4. Algorithm running time comparison

CONCLUSION

In this paper, a variable sliding window based data stream frequent itemsets mining algorithm of VSW-SCPS is proposed. In this algorithm, we have proposed a new tree structure of SCPS-tree. This tree captures the data of a sliding window, and adjust its structure dynamically by the BSM strategy in order to achieve improve mining speed when the data flowing in. The window size is automatically adjusted based on the concept drift detection. The window expand when concept drift isn't detected, otherwise it reduce and remove old information. By recording the

support count before and after the check point on the tail node, the SCPS-tree can be easily remove all the expired data when window sliding. Moreover, it can improve window update efficiency and reduce memory cost.

ACKNOWLEDGMENTS

This work was financially supported by Chongqing University of Posts and Telecommunications Doctoral Foundation (NO. A2015-18).

REFERENCES

1. Kou Xiangxia, Ren Yonggong, Song Kuiyong, An algorithm for mining frequent itemsets in data streams over sliding window[J]. Computer Applications and Software, 2013, 30(1):143-146.
2. Widmer G, Kubat M. Learning in the presence of concept drift and hidden contexts[J]. Machine Learning, 1996, 23(1):69-101.
3. Manku G S, Motwani R. Approximate frequency counts over data streams[J]. Proceedings of the Vldb Endowment, 2012, 5(12):1699-1699.
4. Giannella C, Han J, Pei J, et al. Mining Frequent Patterns in Data Streams at Multiple Time Granularities[J]. Data Mining Next Generation Challenges & Future Directions, 2003.
5. Chi Y, Wang H, Yu P S, et al. Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window[C]// IEEE International Conference on Data Mining. IEEE, 2004:59-66.
6. Li Guohui, Chen hui, Mining the Frequent Patterns in an Arbitrary Sliding Window over Data Streams[J]. Journal of Software, 2008, 19(10):2585-2596.
7. Deypir M, Sadreddini M H, Hashemi S. Towards a variable size sliding window model for frequent itemset mining over data streams[J]. Computers & Industrial Engineering, 2012, 63(1):161-172.
8. Shin S J, Lee D S, Lee W S. CP-tree: An adaptive synopsis structure for compressing frequent itemsets over online data streams[J]. Information Sciences, 2014, 278:559-576.
9. Zaki M J. Scalable Algorithms for Association Mining[J]. IEEE Transactions on Knowledge & Data Engineering, 2000, 12(3):372-390.
10. Han J, Pei J, Yin Y, et al. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach[J]. Data Mining and Knowledge Discovery, 2004, 8(1):53-87.
11. Leung K S, Khan Q I. DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams[C]// International Conference on Data Mining. IEEE, 2006:928-932.
12. 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.
13. Tanbeer S K, Ahmed C F, Jeong B S, et al. CP-Tree: A Tree Structure for Single-Pass Frequent Pattern Mining[C]// Advances in Knowledge Discovery and Data Mining, Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings. 2008:1022-1027.
14. Koh J L, Lin C Y. Concept Shift Detection for Frequent Itemsets from Sliding Windows over Data Streams[C]// Database Systems for Advanced Applications, DASFAA 2009 International Workshops: Benchmarx, Mcis, Wdpp, Ppda, Mbc, Phd, Brisbane, Australia, April. 2009:334-348.