A Single-Pass Algorithm for Incremental Mining Patterns over Data Streams

Shankar B. Naik
Dept. of Computer Science
S.S.A. Govt. College
Pernem, Goa, India
xekhar@rediffmail.com

Jyoti D. Pawar
Dept. of Computer Science and Technology
Goa University
Goa, India
jyotidpawar@gmail.com

***Abstract*--Finding frequent itemsets from transactional data streams is a challenging task due to the large volumes of data in the data stream. It is not feasible to store all the elements of the data stream at once in the memory for future analysis. Approaches requiring multiple scans of the data stream elements are not suitable in a data stream environment. The number of frequent itemsets is too large. Searching for an itemset in a large set of itemsets is a time consuming process. In this paper an algorithm has been proposed to generate frequent closed itemsets from data stream. It generates frequent closed frequent itemsets without requiring multiple scans of the data stream elements in the sliding window. It maintains a set of closed itmsets in an incremental manner. The number of closed frequent itemsets is small as compared to that of all the frequent itemsets. These two properties of the algorithm makes it both memory and time efficient.**

Keywords--Data streams, Data mining, Sliding window model, Closed frequent item sets

I. INTRODUCTION

Finding frequent itemsets from data streams aims to generate frequent itemsets from a transactional data streams. A transactional data stream has itemsets as its elements[8].

Finding frequent itemsets from a data stream is difficult[2]. This is because of the large volume of data which is potentially of unbound size. This makes it impossible to store all the elements of the data stream to be stored in the computer memory at once for analysis. An element of a data stream once discarded cannot be read again.

There are three most common approaches used in processing a data stream[2][3][6]. They are the landmark window model, the damped window model, and the sliding window model. The landmark window model considers elements of the data stream from a timestamp called as a landmark till the latest element. The weighted window model considers the latest elements more significant than the earlier ones. The sliding window model considers the latest n data stream elements, where is the size of the sliding window.

The number of frequent itemsets is huge. The memory and time requirements in maintain such a huge set of frequent itemsets are large. The solution to this problem is to mine frequent itemsets that are closed. A closed itemsets has no superset with a similar support. A closed frequent itemset is a closed itemset that is frequent[7][8].

A closed item set is frequent if its support is greater than or equal to the minimum support threshold. The rationale behind generating closed frequent itemsets is that they are less in number and contain complete information all the frequent itemsets.

Generation of itemsets often require multiple database scans. Approaches requiring multiple scans of database are not suitable for data stream processing.

The algorithm SPAIM-CFI presented in this paper finds closed frequent item sets from a transactional data stream. SPAIM-CFI is a single pass algorithm. Unlike other few approaches, it doesn't require multiple scans of the data stream and sliding window. It reads a transaction of the data only twice, i.e., while it arrives and while it leaves the sliding window.

The remainder of the paper has the sections as described - Section 2 describes the work related to the study. The problem is definition is given in section 3. The algorithm SPAIM-CFI is proposed in Section 4. Experimental study is described in Section 5 and finally, Section 6 concludes the paper.

II RELATEDWORK

Chi.et.al proposed the algorithm Moment [5] to find closed frequent itemsets from a data stream. This is believed to be considered as the first approach in this area. It generates closed frequent itemsets using a tree structure which maintains all the itemsets. The number of itemsets generated and maintained by Moment is huge.

Li. Et al. [7] proposed the algorithm NewMoment to find closed frequent item sets from a transactional data stream using a transaction-sensitive sliding window. It uses a tree called a NewCET which maintains closed frequent itemsets in its nodes. NewMoment performs better than Moment as it maintains only the closed frequent itemsets unlike Moment which maintains the frequent as well as infrequent itemsets. NeMoment reads the NewCET tree and uses the information there to generate the closed frequent itemsets. Every time a new element enters the sliding window, NewMoment generates the NewCET tree to update the list of closed frequent itemsets. It generates the tree NewCET again also when a data stream element leaves the sliding window. Generation of NewCET requires NewMoment to have multiple scans of the elements in the sliding window which is time consuming.

NewMoment uses a minimum support threshold that is fixed. The minimum support value is pre-defined by the user. User cannot change the minimum support value at the time of result generation.

The algorithm SPAIM-CFI presented in this paper allows the user to provide the minimum support value online. Since SPAIM-CFI generates closed item sets, they are less in number. SPAIM-CFI reads a transaction from the sliding window only once, unlike some of the previous approaches where multiple scans are required. SPAIM-CFI does not have to access the sliding window when a transaction is added or deleted. It accesses the summary data structure only to maintain the closed item sets. This feature makes SPAIM-CFI time efficient.

For lower values of minimum support, SPAIM-CFI outperforms other algorithm. User can specify the minimum support value online. For higher values of minimum support threshold, the efficiency of SPAIM-CFI can be improved by restricting it to generated only frequent closed item sets.

III. PROBLEM DEFINITION

A) Terminology

Let I= {i1, i2, …, im} be a set of literals representing the items.

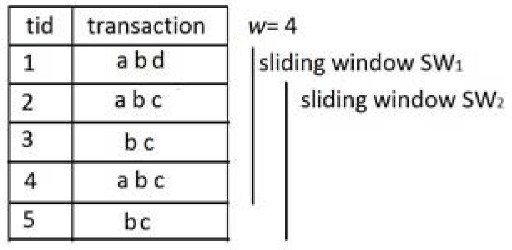
A transaction T= (tid, x1,x2,…, xn) is an (n+1)-tuple where xi∈I, for 1≤i≤n, n is the size of the transaction, and tid is the unique identifier of the transaction in the data stream.

A transactional data stream D = T1,T2,…,TN is a sequence of transactions, where N is the tid of the latest transaction TN in the data stream (Figure1).

An item set X={x1, x2,…, xn} is a collection of items where each item xi∈I.

A sliding window SW of size w contains the latest w transactions of the data stream (Fig. 1.).

The support of an item set X, denoted as sup(X), in a sliding window SW is the number of transactions in SW having X as its subset. An item set X is frequent if sup(X)≥ s.w, where w is the size of sliding window and s is a user specified minimum support threshold (0 ≤ s ≤ 1)[8][9].



tid	transaction
1	a b d
2	a b c
3	b c
4	a b c
5	bc

Figure 1. Sliding Window Example

An item set X is closed if it does not have a proper superset having same support.

An item set X becomes a closed frequent item set if it is both closed and frequent.

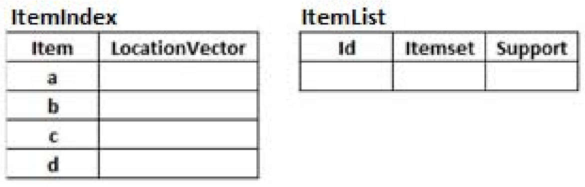
B) Problem Statement

Given a data stream D, a sliding window SW of size w and a minimum support threshold s, the problem is to incrementally mine closed frequent item sets in SW.

C) Proposed summary data structure

The summary data structure consists of (1) Item sets; and (2) ItemIndex (Figure 2).

1). ItemList(ItemSets): ItemList contains closed frequent item sets generated anytime. ItemList is a table with fields: (1) Id; (2) Item Set; and (3) Support. Id is the identifier of the Item set in ItemList table. The field Support is the support of the Item set in the current sliding window.



ItemIndex

Item	LocationVector
a	
b	
c	
d	

ItemList

Id	Itemset	Support

Figure 2 Summary data structure

2) ItemIndex (Index): ItemIndex is used to search item sets in ItemList table. ItemIndex has two fields: Item, and LVector. The LVector (LocationVector) is a bitsequence in which the ith bit is set to one if the Item belongs to the Item set in ItemList table and i is the Id of Item set in ItemList.

ItemIndex is very efficient in searching for item sets in ItemList table.

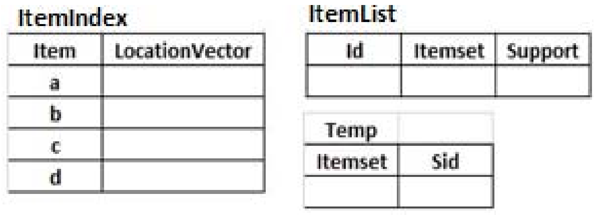
IV. ALGORITHM SPAIM-CFI (SINGLE PASS ALGORITHM FOR INCREMENTAL MINING CLOSED FREQUENT ITEM SETS)

The algorithm SPAIM-CFI consists of two steps: (1) Insertion; and (2) Deletion. The Insertion step is performed when a transaction arrives at the sliding window. The Deletion step is performed when a transaction leaves the sliding window.

A) Insertion Step

When a transaction containing the item set X arrives at the sliding window, supports of the subsets of X only change. SPAIM-CFI first searches for the subsets of X in Item sets and their supports are updated. SPAIM-CFI maintains a temporary list of item sets called Temp. Temp consists of two fields: (1) Item set; and (2) SId. SId is the Id of the item set in the ItemList which is a superset of Item set with highest support (Figure 3).

When a new transaction containing item set X arrives at the sliding window, SPAIM-CFI inserts X into Temp with Id 0(Figure 5, line 2). It then finds all the subsets of X which are closed for which SPAIM-CFI performs intersection of X with every item set in ItemList having at least one item common to X. SPAIM-CFI uses ItemIndex table to locate only those item sets in ItemList whose intersection with X will be a non-empty set. ItemIndex contains a vector of bits for each item such that a value 1 at ith bit in the vector means that the item is present in the Item set at location (Id) i in ItemList. To locate all such items containing at least one item of X, a bitwise OR of Vectors of items in X is performed. The bits having value 1 in the resultant vector indicate the Ids of the required Item sets in ItemList (Figure 5, sections 3 and 4)



ItemIndex

Item	LocationVector
a	
b	
c	
d	

ItemList

Id	Itemset	Support

Temp

Itemset	Sid

Figure 3: Summary data structure with Temp

Let Xi be the item set obtained by intersecting X with item set in ItemList having Id i. If Xi is not present in Temp then Xi is inserted into Temp with SId i. If Xi is already present in Temp with SId k, then the SId of Xi in Temp is replaced by i only if Support of Item set with Id i in ItemList is greater than the Support of Item set with Id k in ItemList.

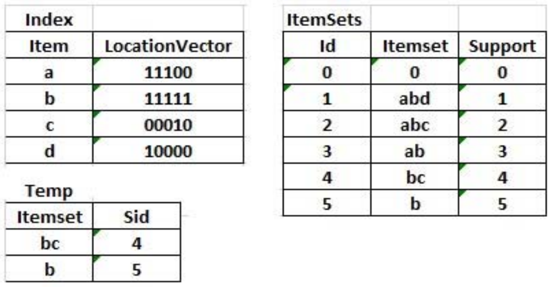
SPAIM-CFI then updates ItemList using Temp. For an Item set Xi in Temp with SId i, if the Item set with Id i in ItemList equals Xi, then the Support of that item set in ItemList is increased by 1. Otherwise, Xi becomes an new closed item sets and is inserted into ItemList at the first available location k and assigned the and assigned the Id k (Figure 5, section 5) .When an item set is inserted into ItemList at ith record, the ith bit in the LVectors in ItemIndex table of all the items belonging to the inserted item set is set to 1.

SPAIM-CFI deletes the contents of Temp table at the end of Add step. In-order to prevent the ItemList table from simply increasing in size, SPAIM-CFI inserts a new item set at the first available location in the ItemList table. This location can be easily found by performing bitwise OR of all the Vectors in ItemIndex. The position of the first bit with value 0 represents the first empty location in ItemList table.

B) Deletion Step

Let X be the itemset which is contained in the transaction leaving the sliding window. In this case only the subsets of X in ItemSets are affected by the removal of itemset X. SPAIM-CFI searches for these itemsets and reduces their support value by one. While this is done some itemsets in the ItemSets table my cease to be closed. Such itemsets are eliminated from the ItemSets table. When an itemset is removed from that ItemSets table the corresponding position bits of all items of the itemset in the Vector are set to 0.

To generate the subsets of X the Deletion step performs intersection of X with the Item sets in ItemList table containing at least one item in X. The Deletion step uses Vectors in ItemIndex table locate these Item sets in ItemList table. In order to locate their Ids, a bitwise OR is performed between the Vectors of the items belonging to X. The positions of bits with value 1 represent the Ids of the subsets of X present in ItemList table. This reduces the search time considerably.



Index

Item	LocationVector
a	11100
b	11111
c	00010
d	10000

ItemSets

Id	Itemset	Support
0	0	0
1	abd	1
2	abc	2
3	ab	3
4	bc	4
5	b	5

Temp

Itemset	Sid
bc	4
b	5

Figure 4. ItemIndex, ItemList, and ITemp

```
Input:  Datastream D
Output: ItemList of closed item sets for SW1
Procedure Insertion(Transaction T)
1.      Let X be the item set in T
2.      Insert X  into ITemp with SId 0
3.      Generate subsets of X from relevant item sets in ISsets
4.      For each subset Xi  generated item set with Id I in
ItemList
4.1     if Xi not present in ITemp
4.1.1   Add Xi to ITemp with SId i
4.2     else
4.2.1   replace SId of Xi in ITemp with I if Support
at Id=I >Support at Id=SId of Xi
5.      For each item set Xi in ITemp with SId i
5.1     if Xi=Item set Xk in ItemList with Id i
            5.1.2 increase Support of Xk in ItemList by 1
5.2     else
5.2.1   insert Xi into ItemList
5.2.2   for each item n belonging to Xi 5.2.3 set the i^th bit of
Vector of a in  ItemIndex to 1.
```

Figure 5 Insertion Step

The Deletion uses the ITemp table to keep a track of itemsets in ItemSets table that no longer remain as closed itemsets. ITemp is a table which has three fields: Itemset which represents an itemset; SId which represents the Id of the itemset superset of the same itemset and has its support equal to the support of the itemset in the ItemSets table; and HSSId which denotes Id of Itemset's supersets which are closed from the ItemList table.

Let the itemset contained in the data stream element leaving the sliding window be X. The Deletion step find the itemsets in ItemList table having at least one element common to X. It then performs intersection of these itemsets with X. The resultant itemsets are inserted into ITemp table. It then updates their SIds and HSSIds in the Itemp table. It then decreases the supports of these itemsets in the ItemList table by one as described in the algorithm.

The algorithm deletes the itemset X from ItemList table, if the support of the itemsets with Id SId of X in ITemp table equals the support of the itemset with id HSSId of X in ITemp table. The Deletion step then delete the itemsets in the ITemp table.

```
Procedure Deletion( Transaction T)
1.  Let X be the item set represented by T
2.  Perform bitwise OR of Vectors of items belonging to X to
    generate PoritionVector(X)
3.  For i such that the ith bit in PositionVector(X) is 1
    3.1.  Generate TempX=X∩Item set(i)
    3.2.  If TempX is not present in ITemp
        3.2.1.  Insert TempX in ITemp
        3.2.2.  Set SId to i
        3.2.3.  Set HSSId to j, where j is the Id of the proper
                superset of TempX which has the highest support
    3.3.  Else if TempX is present in ITemp with SId s and HSSId
          h
        3.3.1.  If sup(Id=s)<sup(Id=i)
            3.3.1.1.  Set SId of TempX to i
        3.3.2.  If sup(Id=h)<sup(Id=i) and TempX⊂Item set(i)
                then
            3.3.2.1.  Set HSSId of TempX as i
4.  Decrease the supports of item sets in ItemList present in
    ITemp by one
5.  For each item set I in ITemp
    5.1.  If I is not frequent
        5.1.1.  Remove I from ItemList
        5.1.2.  Update ItemIndex accordingly
        5.1.3.  Is sup(HSSId)=sup(SId) then
            5.1.3.1.  Remove I from ItemList
            5.1.3.2.  Update ItemIndex accordingly
        5.1.4.  Insert Id of I into ISVList
```

Figure 6. Deletion step

V. Experimental Results

The experiments were performed on 2.26GHz Intel® Core™ i3 PC with 3 GB memory and with Windows 7 operating system. SPAIM-CFI is implemented in C++ language. The programs were compiled using GNU GCC compiler. Two data sets, synthetic and a real data set, were used in the experiments.

The synthetic dataset was generated using IBM Synthetic Data Generator software [1]. The parameters of the synthetic dataset generated are given in Table I.

TABLE I

Parameter	Value
Number of transactions	270K
Average items per transaction	10
Number of items	210

A) Execution time versusm sliding window size

The value of minimum support threshold in this experiment was set to 0.2.

Figure 7 shows that SPAIM-CFI requires less time for a transition of a sliding window as compared to NewMoment. These observations are done by taking average of 50 transitions of a sliding window.

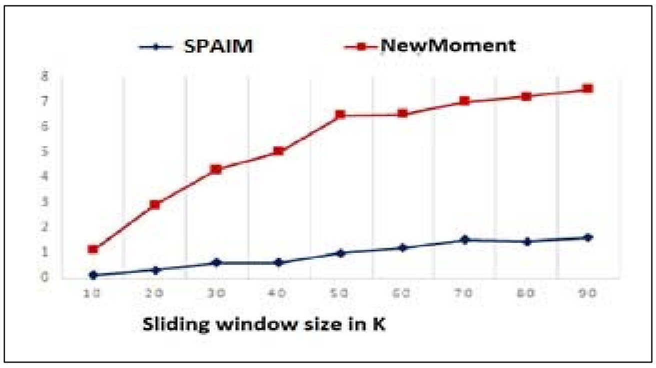


Figure 7. Average time for a transition of sliding window in seconds

B) Executin time versus minimum support threshold

This experiment was performed by varying the value of minimum support threshold. The size of sliding window is 50K. Experiments show that the SPAIM-CFI requires less time than NewMoment for lower values of minimum support threshold (Fig. 16).

For higher values of minimum support NewMoment outperforms SPAIM-CFI in term of time requirements (Figure 9). This is because SPAIM-CFI is minimum support independent and generates all closed item sets, whereas, NewMoment generates closed only the item sets that are frequent. Since the number of frequent closed item sets generated is less, NewMoment is more time efficient than SPAIM-CFI for higher values of minimum support.

The efficiency of SPAIM-CFI can be improved by restricting it to generate only closed item sets which are frequent.

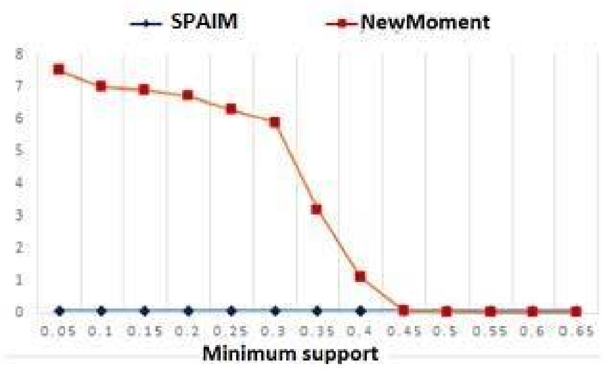


Figure 8. Average time of transition of sliding window in seconds

For higher values of minimum support, SPAIM-CFI requires more memory that NewMoment as SPAIM-CFI maintains all closed frequent item sets in its summary data structure as compared to the frequent closed item sets only in case of NewMoment.

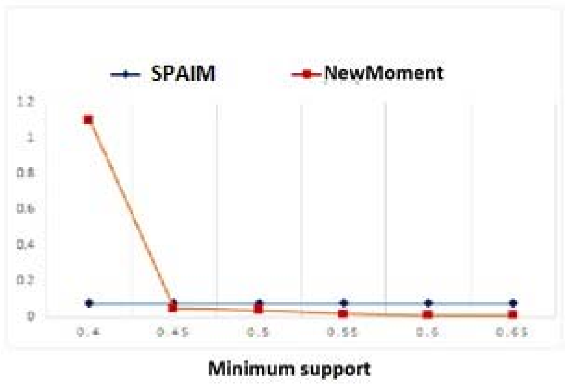


Figure 9. Average time required for transition of sliding window in seconds

CONCLUSION

The algorithm SPAIM-CFI is proposed to mine closed frequent item sets over transactional data streams. It uses a summary data structure to store the results generated, which is efficient in searching item sets stored in it. The proposed algorithm SPAIM-CFI generates all close item sets and allows the user to specify the value of minimum support threshold online. The experiments show that SPAIM-CFI outperforms NewMoment algorithm in terms of time requirements for lower values of minimum support. For higher values of minimum support, the time efficiency of SPAIM-CFI can be improved by having it to generate only the frequent item sets, which motivates us for our future work.

REFERENCES

[1] Agrawal, R. and Shrikant, R.1994. Fast algorithms for mining association rules.Proceedings of the 20th international conference on very large databases, 487-499, (1994).

[2] Babock, B., Babu, S., Motwani, R. and Widom. J. 2002 Models and issues in data stream systems. Proceedings of the 21st ACM SIGMOD-SIGACT-AIGART symposium on principles of database systems, 1-16, (2002).

[3] Chang, J., and Lee, W.(2004) Decaying obsolete information in finding recent frequent item sets over data stream. IEICE Transaction on Informations and Systems, 6, (2004).

[4] Chang, J., and Lee, W. 2004. A sliding window method for finding recently frequent item sets over online data streams. Journal of information science and engineering, 4, (2004).

[5] Chi Y., Wang, H., Yu, P., and Muntz. R..2004. MOMENT: Maintaining closed frequent item sets over a stream sliding window, Proceedings of the 4th IEEE international conference on data mining, 59-66, (2004).

[6] Jin, R. and Agrawal, G.2005. An algorithm for in-core frequent item set mining on streaming data. Proceedings of the 5th IEEE international conference on data mining, (2005).

[7] Lee, H.F. and Ho, C.C,.and.Lee, S.Y. 2009 Incremental updates of closed frequent item sets over continuous data streams, Expert system with application, 2-36, (2009).

[8] Naik, S.B., and Pawar, J.D.2013 An Efficient Incremental Algorithm to mine frequent item sets in data streams. Proceeding of the 19th International Conference on Management of Data (COMAD), 117-120, (2013).

[9] Naik, S.B. and Pawar, J.D. 2015 A quick algorithm for incremental mining closed frequent item sets over data streams. Proceeding of the 2nd IKDD CODS, (2015)