An Efficient Incremental Algorithm to Mine Closed Frequent Itemsets over Data Streams

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

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

ABSTRACT

The purpose of this work is to mine closed frequent itemsets from transactional data streams using a sliding window model. An efficient algorithm IMCFI is proposed for Incremental Mining of Closed Frequent Itemsets from a transactional data stream. The proposed algorithm IMCFI uses a data structure called INdexed Tree(INT) similar to NewCET used in NewMoment[5]. INT contains an index table ItemSet Reference List (ISRList) which contains information about the locations of closed frequent itemsets stored in the summary data structure. ISRList helps in quick searching of closed frequent itemset already generated which in turn reduces the overall time required to mine new closed frequent itemsets from a sliding window. Experiments show that IMCFI is time and space efficient when compared to NewMoment.

Keywords: Data streams, Data mining, Sliding window, Closed frequent itemsets

1. INTRODUCTION

Mining frequent itemsets from a transactional data stream is an important research problem in data mining. The aim of this problem is to generate frequent itemsets from transactional data streams.

Mining frequent itemsets from a data stream is a challenging task because (1) the size of the data stream is unknown; (2) it is not possible to store the elements of the entire data stream for analysis; and (3) the results generated may contain errors [2]. There are three approaches used to mine frequent itemsets in data streams: landmark windows [6], damped windows [3], and sliding windows [4]. In the landmark window model, frequent itemsets are generated for transactions between the landmark and the latest transaction. In damped window model, the recent transactions are considered more important than the previous ones.

The 19th International Conference on Management of Data (COMAD), 19th-21st Dec, 2013 at Ahmedabad, India.

In the sliding window model mining is performed on a fixed number of recently generated transactions.

The number of frequent itemsets discovered in a sliding window of a data stream is large. The cost required to maintain a large set of frequent itemsets in a data stream is more. Especially, in the case of low threshold values the performance of the algorithm may be degraded due to a large number of frequent itemsets. This problem can be avoided by mining only the closed frequent itemsets. The purpose of this work is to mine closed frequent itemsets from transactional data streams using a sliding window model. An efficient algorithm is proposed to mine frequent closed itemsets from a transactional data stream.

The remainder of the paper is organized as follows-Section 2 is on the related work. The problem is defined in section 3. The proposed algorithm is presented in Section 4. Section 5 describes the experimental results and Section 6 concludes the paper.

2. RELATED WORK

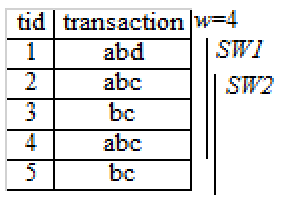
An algorithm called Moment was proposed by Chi. et al. [5] which may be considered as the first to find closed frequent itemsets in data streams. Li. Et al. [7] proposed an algorithm called NewMoment to mine the set of closed frequent itemsets from transactional data streams with a transaction-sensitive sliding window. NewMoment uses an effective bit-sequence representation of items to reduce the time and memory needed to slide the window. It uses an in-memory summary data structure called NewCET to generate a set of closed frequent itemsets.

In this paper we have proposed an algorithm in which the summary data structure used in NewMoment is modified to enhance quick search for closed frequent itemset already generated. The proposed algorithm uses a data structure similar to NewCET called INdexed Tree(INT). INT does not use a hash table. Instead, it maintains a list called ItemSet List(ISList) of frequent itemsets with their supports. For each item contained in itemsets of ISList, INT also maintains an index table called ItemSet Reference List (ISRList) which contains information about the positions of all the itemsets in ISList containing the item. This list helps in identifying the itemsets containing the item without having to scan the entire list of closed frequent itemsets. The increase in memory to store ISRList is negligible as the list of positions of itemsets, containing an item, is stored as a sequence of bits.

NewMoment updates the set of closed frequent itemsets by generating NewCET tree in all the steps. NewMoment traverses the NewCET tree in depth-first manner. The supports of new itemsets in NewCET are updated by referring to the original bit-sequences of the itemsets. When a transaction is removed from the current sliding window IMCFI updates the set of closed frequent itemsets and their supports by referring only to the ISList. Since this step does not involve access to the data in sliding window or generation and traversal of any tree like structure a considerable amount of time is saved. Since ISRList is used in searching for itemsets the overall time required is further reduced, thereby making IMCFI more time efficient as compared to NewMoment. However, INT only improves the search for closed frequent itemsets while adding a new transaction to the sliding window. Experiments have shown that the proposed algorithm IMCFI runs significantly faster than the NewMoment algorithm.

3. PROBLEM DEFINITION

Let $I=\{i_1,i_2,\ldots,i_m\}$ be a set of literals called items. A transaction $T=(tid,x_1,x_2,\ldots,x_n)$ is an (n+1)-tuple where $x_i \in I$, for $1 \leq i \leq n$, n is the size of transaction, and *tid* is the unique identifier of the transaction. A transactional data stream $D = T_1,T_2,\ldots,T_N$ is a sequence of transactions, where N is the *tid* of the latest transaction T_N (Fig.1). An itemset $X=\{x_1,x_2,\ldots,x_n\}$ is a collection of items where each item $x_i \in I$.



tid	transaction
1	abd
2	abc
3	bc
4	abc
5	bc

Item	SW1	SW2
a	1101	1010
b	1111	1111
c	0111	1111
d	1000	0000

Fig. 1. bit-sequences in Sliding Window

A sliding window *SW* of size *w* contains the latest *w* transactions of the data stream (Fig. 1.). A bit-sequence $bitseq(x) = (b_1,b_2,\ldots,b_w)$ of an item x is a sequence of bits, where w is the size of sliding window *SW*. A bit b_i is set to 1 if item x is present in the i^{th} transaction of the sliding window *SW* [7]. Otherwise b_i is set to 0. In Fig.1, $bitseq(\{a\})=(1101)$ for the sliding window SW_1. The bit-sequences of n-itemsets, where $n \geq 2$, is constructed by bitwise AND of two or more (n-1)-itemsets. In Fig. 1, $bitseq(\{ab\})=bitseq(\{a\}) \cap bitseq(\{b\})$ which is (1101). Fig. 1 shows the sliding windows SW_1 and SW_2 in the form of bit sequences of items for the example in Fig.1.

The support of an itemset X, $sup(X)$, in a sliding window *SW* is the number of transactions in *SW* having X as a subset. The support of an itemset X in a sliding window is the total count of the number of 1s in $bitseq(X)$. An itemset X is frequent if $sup(X) \geq s.w$, where s is a user specified minimum support threshold ($0 \leq s \leq 1$). An itemset X is closed if it does not have a proper superset with the same support. An itemset X is a closed frequent itemset if it is closed and frequent.

We define our problem statement as follows: Given a sliding window *SW* of size *w* and a minimum support threshold *s*, the problem is to find the set of closed frequent itemsets in *SW*, which contains the latest *w* transactions of the data stream *D*.

4. ALGORITHM IMCFI (INCREMENTAL MINING OF CLOSED FREQUENT ITEMSET)

4.1 INT- the proposed data structure

The summary data structure INT consists of (1) ISTree; (2) ISList; and (3) ISRList.

ISTree shown in Fig.2 is similar to the NewCET used in NewMoment. It consists of nodes where each node represents an itemset X and its support. Each node also contains bit-sequences of all single-itemsets for the current sliding window. ISTree maintains closed frequent itemsets only.

ISList contains the set of closed frequent itemsets. ISList is a table with three fields: (1) ItemId; (2) Itemset; and (3) Support. ItemId is the key or the identifier of the Itemset in ISList table. Support is the support of Itemset in the current sliding window.

ISRList is a table with two fields: Item, and the PositionVector. The PositionVector of an item is a bit-sequence in which the i^{th} bit is set to one if the Item belongs to the Itemset in ISList table and i is the ItemId of Itemset. ISRList speeds up the operation of searching for itemsets in ISList table.

4.2 The working of IMCFI

In this section we describe the working of IMCFI algorithm. The algorithm consists of three steps: (1) Initialize; (2) Delete; and (3) Update. The Initialize step is performed at the beginning when the first sliding window is filled with the first *w* transactions of the data stream. The Delete step is performed to update the closed frequent itemsets when a transaction leaves the sliding window, while the Update step is performed to update the closed frequent itemsets when a transaction arrives to the sliding window.

4.1 Initialize Step

In this step the first sliding window SW_1 is filled with the first *w* transactions of the data stream as shown in Fig. 1. The sliding window SW_1 contains bit sequences of all items. Each bit sequence is of size *w*, which is the size of sliding window. For an item x, the i^{th} bit in $bitseq(X)$ is set to 1 if the i^{th} transaction in *SW* contains x.

IMCFI creates 1-level nodes of ISTree. Each node contains a singleton itemset and its bit sequence. Size of each bit-sequence is w. The ISList table is empty. The ISRList table contains all the items with their corresponding

PositionVectors having all bits set to 0 as there are initially no itemsets in ISList.

The algorithm works in a depth-first manner. For each node *n*, containing a frequent itemset which is not contained in any closed frequent itemset in ISList, child nodes are generated by performing a bitwise AND of the bit-sequences of node n and the bit-sequences of its siblings which are frequent. The same procedure is followed for each child node of *n*. If there exists a node *n* such that none of its child nodes has the same support and if *n* is not contained in any itemset in ISList table with same support then *n* becomes a closed frequent itemset. The itemset represented by node *n* is inserted into the first available location ISList table. When an itemset is inserted into ISList at i^{th} record, the i^{th} bit in the PositionVectors in ISRList table of all the items belonging to the inserted itemset is set to 1.

Let the minimum support value *s* be 0.5. The ISTree after performing the Initialize stage is shown in Fig.2.

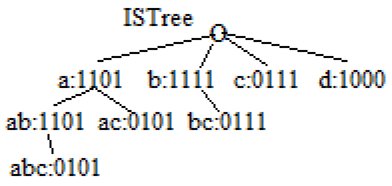


Fig. 2. ISTree for sliding window SW_1

The algorithm traverses the tree to find out the closed frequent itemsets. The closed frequent itemsets are stored in the ISList as shown in Fig.3.

ISRList

Item	PositionVector
a	1100
b	1111
c	1010
d	0000

ISList

ItemId	Itemset	Support
1	abc	2
2	ab	3
3	bc	3
4	b	4

Fig. 3. ISRList and ISList tables after Initialize step

The ISRList table contains the PositionVectors for all the itemsets. The PositionVector of item *a* contains 1 at the first and second positions. This indicates that item *a* is present in itemsets with ItemId 1 and 2 in the ISList table.

4.2 Delete Step

When a transaction is deleted from the sliding window, all the bits in the bit-sequences of items in sliding window are shifted to left by one bit. The right most bit of all bit-sequences is set to 0. When a transaction containing the itemset *X* is removed from the sliding window the supports of only the subsets of *X* should be decreased by one. Some of the subsets of *X* which are in ISList table may not remain closed frequent itemsets after the transaction is deleted. The Delete step first finds all subsets of *X*. After having found these subsets it then checks for those subsets which have become non-closed and eliminates them from the ISList. When an itemset X_r with ItemId *i* is removed from ISRList, the i^{th} bit in the PositionVector of the items belonging to X_r in the ISRList table is changed from 1 to 0.

In order to find out the itemsets which do not remain as closed frequent itemsets, the Delete step maintains a temporary list of itemsets called ISTemp. ISTemp consists of three fields: (1) Itemset; (2) SItemId; and (3) HSSItemId. SItemId of Itemset *X* is the ItemId of the itemset in the ISList which is a superset of Itemset *X* with support equal to support of Itemset *X*. HSSItemId of Itemset is the ItemId of closed superset of Itemset in the ISList with largest support and is not a proper subset of *X*, where *X* is the itemset in the transaction leaving the transaction window. IMCFI deletes the contents of ISTemp table at the end of Delete step. In-order to prevent the ISList table from simply increasing in size, IMCFI inserts a new itemset at the first available location in the ISList table. This location can be easily found by performing bitwise OR of all the PositionVectors in ISRList. The position of the first bit with value 0 represents the first empty location in ISList table.

For the example in Fig. 1 the bit-sequences of items in the sliding window after deleting the oldest transaction are, *bit-seq(a)=1010*, *bit-seq(b)=1110*, *bit-seq(c)=1110*, and *bit-seq(d)=1000*. The algorithm stores the items of the deleted transaction and left-shifts the bits of all bit-sequences by one. The right most bit of all bit-sequences is set to 0. The deleted itemset is *{abd}*.

The contents of ISList and ISRList tables are shown in Fig. 3. ISTemp table is initially empty. The algorithm first finds the bitwise OR of the PositionVectors of items *a*,*b*, and *d* to get bit-sequence 1111. This means that an intersection is required with all the four itemsets in ISList table.

The intersection of *{abd}* with the first itemset *{abc}* is *{ab}*. Since ISTemp does not contain *{ab}*, it is entered into ISTemp table with SItemId and HSSItemID values set to 1 and 0, respectively. The intersection of *{abd}* with the second itemset in ISList Table *{ab}* is *{ab}*. Itemset *{ab}* already exists in ISTemp table with SItemId and HSSItemId values 1 and 0, respectively. Since *sup(2)>sup(1)* the SitemId of *{ab}* in ISTemp is set to 2. The algorithm sets the HSSItemId of *{ab}* to 1 since *{abc}* contains *{ab}* and the support of *{abc}* is the highest among the Itemsets containing *{ab}* in ISList so far. The algorithm repeats the same process with the other itemsets in ISList table. The contents of ISList, ISRList and ISTemp after all intersections are shown in Fig. 4a.

The supports of *{ab}* and *{b}* are decreased by 1. Itemset *{ab}* in ISTemp table has SItemId and HSSItemId values as 2 and 1 respectively. From the ISList, support at ItemId 2 is same to support at ItemId 1. Hence, itemset *{ab}* is not closed itemset. Itemset *{ab}* is removed from ISList. The first bits in PositionVectors of *a* and *b* are set to 0.

For the second itemset *{b}* in ISTemp, the support values for ItemId 4 and 2 are different. Therefore, itemset *{b}* is closed frequent, so it is retained in ISList as shown in Fig. 4b.

Fig. 4a

ISRList

Item	PositionVector
a	1100
b	1111
c	1010
d	0000

ISList

ItemId	Itemset	Support
1	abc	2
2	ab	3
3	bc	3
4	b	4

ISTemp

Itemset	SItemId	HSSItemId
ab	2	1
b	4	2

Fig. 4b

ISRList

Item	PositionVector
a	1000
b	1011
c	1010
d	0000

ISList

ItemId	Itemset	Support
1	abc	2
3	bc	3
4	b	3

ISTemp

Itemset	SItemId	HSSItemId
ab	2	1
b	4	2

Fig. 4. ISList, ISRList and ISTemp table

4.3 Update Step

When a new transaction enters the sliding window, the rightmost bit of *bitseq(x)* is set to 1, if item *x* is present in the newly added transaction, else it is set to 0. Update step maintains a list of the items present in the new transaction. It generates child nodes of all level 1 nodes containing the items in the newly added transaction. The Update step follows the same depth-first procedure as in the Initialize step. The only difference is that, if an itemset is already present in the ISList table then its support is simply updated.

5. EXPERIMENTAL RESULTS

Experiments were performed to compare the performance of IMCFI with NewMoment algorithm. All experiments are carried out on 2.26 GHz Intel® Core™ i3 PC with 3 GB memory and running on Windows 7 system. The proposed algorithm is implemented in C++ and compiled using GNU GCC compiler. The synthetic dataset is generated using IBM Synthetic Data Generator [1]. The number of transactions is 200K, average items per transaction are 10, and the number of items is 200.

5.1 Mining by varying the size of sliding window

This experiment is performed by changing the sliding window size *w* from 10K to 100K. The value of minimum support threshold is set to 0.2. Table 1 below shows that IMCFI requires less time and memory for a transition of a sliding window. These observations are done by taking average of 50 transitions.

Sliding window size(K)	Average window sliding time(seconds)		Memory in K	
	IMCFI	NewMoment	IMCFI	NewMoment
10	0.09	1.1	1.5	31.5
20	0.2	2.9	5	53
30	0.5	4.3	8	90
40	0.6	5	9	110
50	0.7	6.45	11	112.5
60	1	6.5	11.5	126
70	0.9	7	15	129
80	1.05	7.2	19	130
90	1.1	7.5	19.5	131

Table1. Time, memory required in sliding a window

As the size of sliding window increases IMCFI requires less time and memory than NewMoment. This happenes as the size of hash table in NewMoment increases with increase in sliding window size, which is not the case with IMCFI.

6. CONCLUSION

We proposed an efficient incremental algorithm to mine closed frequent itemsets over transactional data streams. A new summary data structure is developed to maintain the closed frequent itemsets in a time efficient manner. The proposed algorithm IMCFI outperforms the NewMoment algorithm in mining closed frequent itemsets over transactional data streams.

REFERENCES

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

[2] B. Babock, S. Babu, R. Motwani, and J. Widom. 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] J. Chang and W. Lee, Decaying obsolete information in finding recent frequent itemsets over data stream. *IEICE Transaction on Informations and Systems,* 6, 2004.

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

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

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

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