EFFICIENT TECHNIQUE FOR MINING FREQUENT PATTERNS OVER DATA STREAM

Niyati M. Mevada and Ms. Jayna B. Shah
Dept. of Computer Engineering ,
Sardar Vallabhbhai patel Institute of technology, Vasad-388306
Gujarat , India

Abstract— Mining frequent items is one of the most important research topics in data mining. In existing system an effective bit-sequence based, one-pass algorithm, called MFI-Trans-SW (Mining Frequent Itemsets within a Transaction Sliding Window), to mine the set of frequent itemsets from data streams within a transaction sliding window which consists of a fixed number of transactions. MFI-TransSW algorithm consists of three phases: window initialization, window sliding and pattern generation. The existing system mines the frequent patterns for the recent data only . In proposed system, we are going to mine the frequent patterns for overall all data. Even the historical data is useful when frequent patterns are mined. As soon as the transaction arrives , each incoming transaction is scanned . If the itemset exist in the transaction the support count is incremented by 1. Otherwise the support count would remain same as it was. Frequent as well as infrequent patterns are maintained in the system. The proposed system not only attain highly accurate mining results, but also run significant faster than existing algorithms for mining frequent itemsets from data streams without using a sliding window.

Index Terms—frequent itemsets , data stream

I. INTRODUCTION

A data stream is a continuous, huge, fast changing, rapid, infinite sequence of data elements. We can say data stream is an ordered sequence of elements that arrives in timely order. It is assumed that the stream can only be scanned once and hence if an item is passed, it cannot be revisited, unless it is stored in main memory. Different from data in traditional static datasets, data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time [1]. It is often refer to as streaming data. In this , it uses multiple segments for handling different size of windows over data streams. Storing these segments in a data structure, the usage of memory can be optimized. Many applications generate large amount of data streams in real time, such as sensor data generated from sensor networks, online transaction flows in retail chains, Web record and click-streams in Web applications, call records in telecommunications, performance measurement in network monitoring and traffic management.

Data streams can be further classified into offline data streams and online data streams. Offline data streams are characterized by regular bulk arrivals [4], such as a bulk addition of new transactions as in a data warehouse system. Online data streams are characterized by real-time updated data that come one by one in time, such as an a continuously generated transaction as in a network monitoring system. Bulk data processing is not possible for one streaming data. Due to the characteristics of data streams, there are some inherent challenges for mining streaming data [8]. First, each data element of stream should be examined at most once. Second, the memory usage in the process of mining data streams should be bounded even though new data elements are continuously generated from the streams. Third, each element due to the characteristics of data streams, there are some inherent challenges for mining streaming data. Fourth, the analytical outputs of the stream should be instantly available when the user requested. Finally, the errors of outputs should be constricted as small as possible. The continuous characteristic of streaming data makes it essential to use the algorithms which require only one scan over the stream for knowledge discovery. The huge nature of stream makes it impossible to store all the data

into main memory or even in secondary storage. This motivates the design of a summary data structure with small footprints that can support both one-time and continuous queries [5]. In other words, one-pass data stream mining algorithms have to sacrifice the correctness in the analytical results by allowing some counting errors. Consequently, previous multiple-pass data mining algorithms studied for static datasets are not feasible for mining data streams.

Frequent itemset mining is a KDD technique which is the basic of many other techniques, such as association rule mining, sequence pattern mining, classification, and clustering. It is impossible to maintain all the elements of data streams [3]. This rapid generation of continuous streams of information has challenged our storage, computation and communication capabilities in computing systems. Data Stream mining refers to informational structure extraction as models and patterns from continuous data streams [5]. Data Streams have different challenges in many aspects, such as computational, storage, querying and mining. Data stream mining differs from traditional data mining since its input of mining is data streams, while the latter focuses on mining (static) databases. Compared to traditional databases, mining in data streams has more constraints and requirements. The mining task should proceed normally and offer acceptable quality of This result, one good stream mining algorithm to possess efficient performance and high throughput [7]. Slight approximate errors occurred in the mining result is usually acceptable by the user[2] [4].

II. EXISTING SYSTEM

Frequent itemset mining approaches have mainly considered the problem of mining transactional databases[9]. In these methods, transactions are stored in secondary storage so that multiple scans over the data can be performed. It accepts only one minimum support and using fixed window length. In these method old data required many times. So, it needs huge memory to stored data. The traditional data mining methodology may not be valid in a data stream. Because it uses huge memory to store data, high processing power, several iterations of the data, uses a uniform minimum support threshold[11]. Here an effective bit- sequence representation of items is used to reduce the time and memory needed to slide the windows. The three main phases of MFI-TranSW are : Window Initialization Phase, Window Sliding Phase & Frequent Itemset Generation Phase. These phases are discussed below .

2.1 Window Initialization Phase :

The window initialization phase is activated while the number of transactions generated so far in a transaction data stream is less than or equal to a user predefined sliding window size [6]. In this phase, each item of the new incoming transaction is transformed into its bit-sequence representation.

2.2 Window Sliding Phase :

The window sliding phase is activated after the current sliding window becomes full. A new incoming transaction is appended to the current sliding window, and the oldest transaction is removed from the window . For removing oldest information, a method is used in the algorithm[1]. Based on the bit-sequence representation, MFI-TransSW algorithm uses the bitwise left shift operation to remove the aged transaction from the set of items in the current sliding window .After sliding the window, an effective pruning method, called Item- Prune, is used to improve the memory usage.

2.3 Frequent Itemset Generation Phase:

The frequent itemsets generation phase is performed only when the up-to-date set of frequent itemsets is requested [10]. Finally we get frequent itemsets for each particular window.

III. PROPOSED SYSTEM

The existing system mines the frequent patterns for the recent data only . We can also mine the frequent patterns for overall all data Even the historical data is useful when frequent pattern are mined. In the proposed system , as soon as the transaction arrives , each incoming transaction is scanned . If the itemset

exist in the transaction. The support count is incremented by 1. Otherwise the support count would remain same as it was. Once it is done for 1- itemset . Similarly the process is carried out for 2-itemset as well as for 3-itemset . The subsets are formed until no pairs are generated. As we are going to increment the counter in our proposed work for each and every itemsets. Here one problem may arise, if the data would be of two or four years then how long we are going to maintain the counter. So we are going to use the concept of data warehouse where for every previous month the past frequent as well as infrequent patterns are dumped into the data warehouse and the counter will start again from one for the new month. Thus the memory usage will also reduced and the data will be preserved in the data warehouse

In our proposed system we provide two options . We can mine patterns either monthly or overall. If month is selected than frequent patterns of that month are displayed. Frequent as well as infrequent patterns are maintained in the system. As no infrequent itemsets are discarded, the accuracy is increased. Thus the proposed system maintains the set of frequent patterns for overall data also.Considered the below example for understanding the working of proposed system. Let the first two transactions in a transaction data stream be <T1 (abd)>, <T2 (acd)>

TRANSACTION	ITEMS
T1	a b d
T2	a c d

Table : 3.1

Scan first transaction as well as calculate support count of it.

ITEMSET	SUPPORT COUNT
a	1
b	1
d	1

Table : 3.2

Generating 2-itemsets.

ITEMSET	SUPPORT COUNT
ab	1
ad	1
bd	1

Table : 3.3

Generating 3-itemsets

ITEMSET	SUPPORT COUNT
Abd	1

Table : 3.4

Scanning new incoming transaction i.e T2 [a c d] and calculating support count.

ITEMSET	SUPPORT COUNT
A	1 + 1
B	1
D	1 + 1
C	1

Table : 3.5

Generating 2-itemsets

ITEMSET	SUPPORT COUNT
ab	1
ad	1 + 1
ac	1 + 1
bd	1
bc	1
dc	1 + 1

Table : 3.6

Generating 3-itemsets

ITEMSET	SUPPORT COUNT
abd	1
abc	1
acd	1 + 1
adb	1
bdc	1

Table : 3.7

Thus the frequent as well as infrequent itemsets are incremented and maintained in the indexing form. Thus we can also fetch the frequent patterns monthly, as all the patterns are dumped and preserved into data warehouse .

IV. EXPERIMENTAL RESULTS

We have analyzed the results of both MFI-TranSW and our proposed system. Clearly we can see here that the proposed system works better than the MFI-TranSW. All the parameters are compared below .

SR No	PARAMETERS	MFI-TranSW	PROPOSED SYSTEM
1.	Execution Time Number of transactions =100	3300 ms	456 ms
2.	Execution Time Number of transactions =800	8891 ms	4753 ms
3.	Execution Time Number of transactions =4000	12056 ms	8945 ms
4.	Execution Time Number of transactions =10000	187601 ms	115593 ms
5.	Number of frequent patterns	Approx 5 to 6 Patterns Generated per window	33 Patterns Generated

Table : 4.1

Figure 4.1 shows that the system works efficiently even on different datasets. The execution time of proposed system is

less than existing system even when different no. of transaction are considered. It proves that the proposed system takes nearly half time than MFI-TranSW.

Figure 4.2 shows the numbers of frequent patterns generated in both the system. MFI-TranSW generated frequent patterns per window whereas proposed system generates the overall frequent patterns.

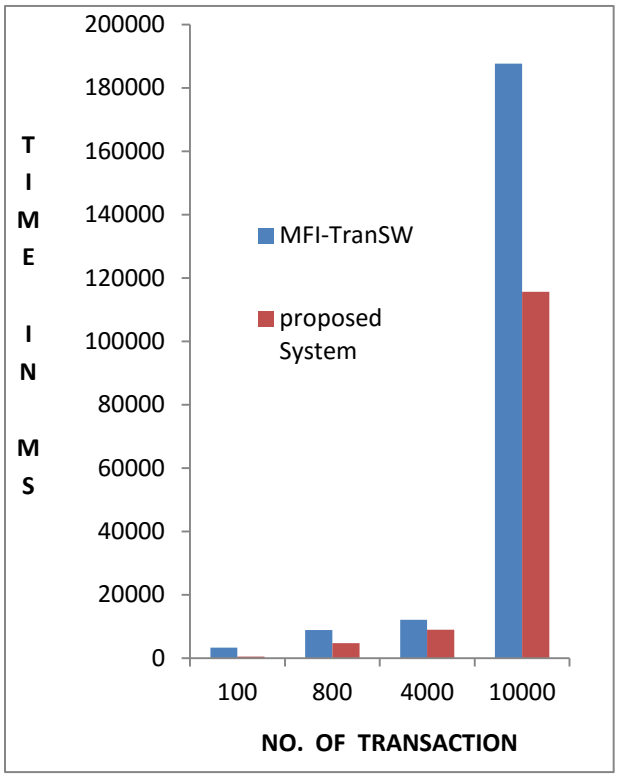

Fig : 4.1

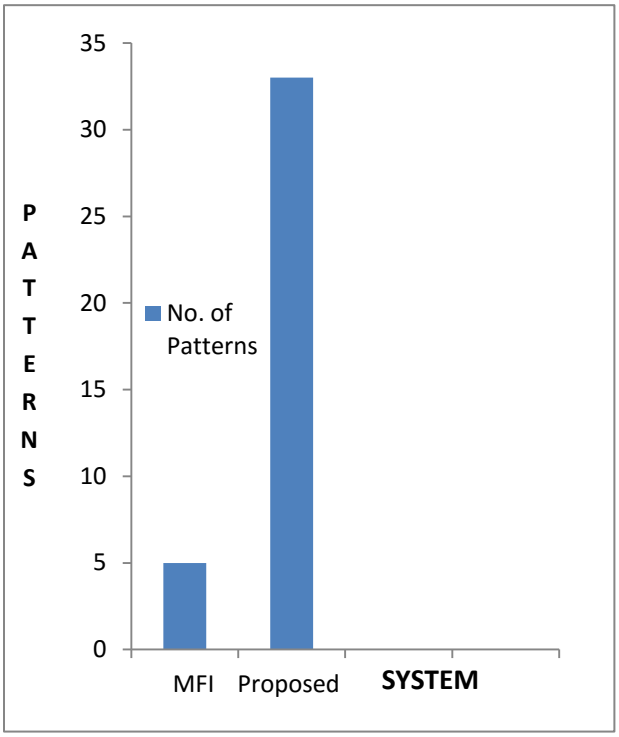

Fig : 4.2

V. Conclusion

We have isolated a number of issues in data streams, the purpose is to introduce the process of mining frequent patterns in data streams and particularly analyze the performance of the algorithm. We proposed a data mining method for finding overall frequent and infrequent items over data stream. An efficient method is used, for mining the set of frequent itemsets over data streams without a transaction sliding window. As we are going to increment the counter in our proposed work for each and every itemsets .If the data would be of two or four years then maintain the counter would be a tedious task. So we are going to use the concept of data warehouse where for every previous month the past frequent as well as infrequent patterns are dumped into the data warehouse and the counter will start from one for the new month. Thus the memory usage will also reduced and the data will be secure in the data warehouse. The proposed system provides us highly accurate mining results and the execution time is less than existing algorithms for mining frequent itemsets from data streams without using a sliding window. We have used the static data streams

in the proposed system. In future we can use online data streams for this system.

REFERENCES

1. R. AGRAWAL, T. Imielinski, and A. Swami. "Mining Association Rules between Sets of Items in Large Databases". In Proceedings of the 2008 International Conference on Management of Data, pp. 207-216, 2008

2. Mr. Velusamy., Ms. Shobana. , Ms. Saranya. "Efficient Mechanism to Discover Frequent Pattern over Online Data Streams" International Journal of Engineering Research & Technology (IJERT) Vol. 2 Issue 12, December – 2013

3. S. Muthukrishnan , "Data streams: algorithms and applications". Proceedings of the fourteenth annual ACMSIAM symposium on discrete algorithms, 2009.

4. Li, H.-F., Lee, S.-Y., Shan, M.-K. (2005a). "Online mining (recently) maximal frequent itemsets over data streams". In Prooceedings of the IEEE RIDE.

5. Chang, J., & Lee, "A sliding window method for finding recently frequent itemsets over online data streams". Journal of Information Science and Engineering, 2005.

6. Yang, C., Li, Y., Zhang, C., & Hu, Y., "A novel algorithm of mining maximal frequent pattern based on projection sum tree", Fuzzy Systems and Knowledge Discovery, vol. 1, pp.458–462, 2007

7. S. Muthukrishnan , "Data streams: algorithms and applications". Proceedings of the fourteenth annual ACMSIAM symposium on discrete algorithms, 2009.

8. Kun Li,, Yong-yan Wang, Manzoor Ellahi, Hong-an Wang "Mining Recent Frequent Itemsets in Data Streams" Fifth International Conference on Fuzzy Systems and Knowledge Discovery , IEEE-2008.

9. Rahul Anil Ghatage "Frequent Pattern Mining Over Data Stream Using Compact Sliding Window Tree & Sliding Window Model" nternational Research

10. Journal of Engineering and Technology (IRJET) Volume: 02 , July-20

11. Hua-Fu Li , Man-Kwan Shan , Suh-Yin Lee. "DSM-FI : an efficient algorithm for mining frequent itemsets in data stream" Knowl Inf Syst (2008) 17:79–97Springer