Efficient Mechanism to Discover Frequent Pattern over Online Data Streams

Mr. Velusamy. A[1], Ms. Shobana. G[2], Ms. Saranya. V[3]
[1]Assistant Professor, *Sri Krishna College of Engineering and Technology, Coimbatore, Tamilnadu, India.*
[2]Assistant Professor, *Sri Krishna College of Engineering and Technology, Coimbatore, Tamilnadu, India.*
[3]Assistant Professor, *Sri Krishna College of Engineering and Technology, Coimbatore, Tamilnadu, India.*

Abstract

Mining frequent items is one of the most important research topics in data mining. The function is to mine the transactional data which describes the item purchased by the customer. Traditional frequent itemset mining approaches have mainly considered the problem of mining static transaction databases. In these methods, transactions are in secondary storage so that multiple scans over the data can be performed. This paper proposes a data mining method for finding recent frequent items over an online data stream. A data stream is a continuous, huge, fast changing, rapid, infinite sequence of data elements. 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. In this method, 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. It also an effective bit- sequence representation of items is uses to reduce the time and memory needed to slide the windows.

Keywords— *Data mining; Data stream; Sliding window model; Association rule; Frequent itemset*

1. Introduction

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. A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. It is impossible to maintain all the elements of data streams [1]. 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. First, each element (e.g., transaction) in the data stream can be examined only once or twice, making traditional multiple-scan approaches infeasible. Second, the consumption of memory space should be confined in a range, despite that data elements are continuously streaming into the local site. The mining task should proceed normally and offer acceptable quality of results. Fourth, the latest analysis result of the data stream should be available as soon as possible when the user invokes a query [2] [6].

This result, one good stream mining algorithm to possess efficient performance and high throughput. Slight approximate errors occurred in the mining result is usually acceptable by the user.

2. Existing System

Traditional frequent itemset mining approaches have mainly considered the problem of mining static transaction databases. 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. The traditional 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.

3. Related Work

The frequency of a pattern is the number of transactions containing the pattern in the transaction database. The problem of frequent pattern mining is to find the complete set of patterns satisfying a minimum support in the transaction database. The downward closure property is used to prune the infrequent patterns. This property tells that if a pattern is infrequent then all of its super patterns must be infrequent. The Apriori algorithm is the initial solution of frequent pattern mining problem. But it suffers from the level-wise candidate generation-and-test problem and needs several database scans [3]. The FP-growth algorithm solved this problem by using FP-tree based solution without any candidate generation and using only two database scans[8]. Other research has been done to efficiently mine frequent patterns. However, this traditional frequent pattern mining considers equal profit/weight for all items.

3.1 Sliding Window

Mining recent frequent patterns using the sliding window technique has also been studied in the literature. Data stream for frequent patterns using a time-sensitive sliding window. The window size is defined by a fixed period of time. In this approach, the incoming stream within a window time period is divided into several batches, and frequent patterns are mined in each batch individually. Using a discounting mechanism, the method discards the old patterns .Chang and Lee proposed estWin that finds recent frequent patterns adaptively over an online transactional data stream using the sliding window model[5][7]. This algorithm requires the minimum support threshold and another parameter termed the significant support to adaptively maintain the approximate frequent patterns window after window.

3.2 Bit-Sequence Method

The authors proposed an Apriori-based algorithm, called MFI-TransSW, which finds complete set of recent frequent patterns by using bit-sequences to keep track of the occurrence of all items in the transactions of the current fixed-sized sliding window. To remove old data and to reflect the inclusion of new data it performs a bit-wise left-shift operation for all bit-sequences. This approach is based on transaction-sensitive sliding window where the bit-sequence update operation is performed at the arrival of every single transaction [6]. The MFI-TransSW applies the level-wise candidate-generation-and test methodology to find the complete set of recent frequent patterns from the current window. Therefore, it suffers from the Apriori limitation of huge candidate pattern generation, especially when mining stream data that contain large number of and/or long frequent patterns, and/or with lower support count. Furthermore, the transaction-by-transaction update mechanism may limit its performance when stream flows at high speed. Again, since the approach maintains the bit-sequence information in full for all items in the window, it fails to achieve memory efficiency when the window contains large number of transactions and distinct items, which is very common in data stream environment[6]. Even though MFITransSW discovers recent frequent patterns from a data stream, it differs significantly from the proposed technique in both mining approach and data processing strategy.

3.3 Association Rule

An association is a rule of the format: LHS _ RHS, where LHS and RHS stand for Left Hand Side and Right Hand Side respectively. These are two sets of items and do not share common items. A set of items is called an itemset. The goal of association rule discovery is to find associations among items from a set of transactions, each of which contains a set of items. Generally the algorithm finds a subset of association rules that satisfy certain constraints [3]. The most commonly used constraint is minimum support. The support of a rule is defined as the support of the itemset consisting of both the LHS and the RHS. The support of an itemset is the percentage of transactions in the transaction set that contains the itemset. An itemset with a support higher than a given minimum support is called frequent itemset[4]. Similarly, a rule is frequent if its support is higher than the minimum support. Minimum confidence, which is the minimum ratio of the support of the rule and the support of the LHS, is another commonly used constraint for association rules.

4. Problem Statement

Let $\psi = \{i1, i2. \ . \ . \ im\}$ be a set of items. A transaction T = (TID,x1,x2, . . . ,xn), xi ϵ ψ , for $1 \leq i \leq$ n, is a set of items, while n is called the size of the transaction, and TID is the unique identifier of the transaction. An itemset is a non-empty set of items. An itemset with size k is called a k-itemset. A transaction data stream TDS = T1, T2, ..., TN is a continuous sequence of transactions, where N is the TID of latest incoming transaction TN. A Transaction-sensitive window (TransSW) in the transaction data stream is a

window that slides forward for every transaction. The window at each slide has a fixed number, w, of transactions, and w is called the size of the window. Hence, the current transaction-sensitive window is TransSWNW+1 = [TN_w+1,TN_w+2, . . . ,TN], where N _ w + 1 is the window id of current TransSW. The support of an itemset X over TransSW, denoted as sup(X), is the number of transactions in TransSW containing X as a subset.

5. Proposed System

We propose a new method, which used 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. It also an effective bit- sequence representation of items is used to reduce the time and memory needed to slide the windows.

Each transaction in the data stream can be examined only once, making traditional multiple-scan approaches infeasible. The consumption of memory space should be confined in a range. The data characteristic of incoming stream may be unpredictable, the mining task should proceed normally and offer acceptable quality of results. The latest analysis result of the data stream should be available as soon as possible when the user invokes a query.

Transaction Data Stream	FIs in TransSW1	FIs in TransSW2
<T1, (acd) >	(a), (b), (c), (e), (ac),	(b), (c), (bc), (be), (ce),
<T2, (bce) >	(bc), (be), (ce), (bce)	(bce)
<T3, (abce) >		
<T4, (be) >		

Table 1. Data stream and the frequent itemsets over two consecutive transSWs.

5.1 Mining of Frequent Itemsets

We describe our proposed single-pass mining algorithm, called MFI-TransSW and its bit-sequence representation of items. Compared with other sliding window based mining techniques, we save memory and improve speed by dynamically maintaining all transactions in the current sliding window by using an effective bit-sequence representation of items.

5.2 Bit-Sequence Representation of an Item

In MFI-TransSW algorithm, for each item X in the current transaction-sensitive sliding window TransSW, a bitsequence with w bits, denoted as Bit(X), is constructed. If an item X is in the ith transaction of current TransSW, the ith bit of Bit(X) is set to be 1; otherwise, it is set to be 0. For example, in Table 1, the first sliding window Trans- SW1 consists of three transactions: (T1, (acd)), (T2, (bce)), and (T3, (abce)), but the TransSW2 consists of transactions: (T2, (bce)), (T3, (abce)), and (T4, (be)). Because itema appears in the 1st and 3rd transactions of TransSW1, the bit-sequence of a, Bit(a), is 101. Similarly, Bit(b) = 011, Bit(c) = 111, Bit(d) = 100, and Bit(e) = 011.

5.3. 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 w. In this phase, each item of the new incoming transaction is transformed into its bit-sequence representation. For example, in Table 1, the first sliding window Trans- SW1 contains three transactions: T1, T2, and T3. The bit sequences of items of TransSW1 in the window initialization phase are shown in Table 2.

5.4. Window Sliding Phase

The window sliding phase is activated after the current sliding window TransSW 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, an efficient method is used in the proposed algorithm. 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. For example, in Table 1, before the fourth transaction T4 (be) is processed, the first transaction T1 must be removed from the current window using bitwise left shift on the set of items. Hence, Bit(a) is modified from 101 to 010. Similarly, Bit(c) = 110, Bit(d) = 000, Bit(b) = 110, and Bit(e) = 110.Then, the new transaction T4, (be) is processed by bit-sequence transform. The result is

shown in Table 3. Note that item d is dropped since Bit(d) = 000, i.e., sup(d)TransSW = 0.

Tid	Items	Bit Sequences in current TransSW1
T1	(acd)	Bit(a)=100, Bit(c)=100, Bit(d)=100
T2	(bce)	Bit(a)=100,Bit(c)=110,Bit(d)=100 , Bit(b)=010, Bit(e)=010
T3	(abce)	Bit(a)=101,Bit(c)=111,Bit(d)=100 , Bit(b)=011, Bit(e)=011

Table 2. Sliding transaction window1 to transaction window2

5.5 The Frequent Itemsets Generation Phase

The frequent itemsets generation phase is performed only when the up-to-date set of frequent itemsets is requested.. The MFITransSW algorithm is shown in Table 3.

First, MFI-TransSW algorithm generates three candidate 2-itemsets, (bc), (be) and (ce), by combining frequent 1-itemsets: (b), (c) and (e), where Bit(b) = 111, i.e.,sup(b) = 3, Bit(c) = 110, i.e., sup(c) = 2, and Bit(e) = 110, i.e., sup(e) = 2. 1-itemset (a) is an infrequent itemset, since its Bit(a) = 010, i.e., sup(a) = 1. All these candidates are frequent itemsets after using bitwise AND operations to count the supports of these candidates. Because the Bit(bc) is 110, the support of candidate 2-itemset bc are 2, i.e., sup(bc) = 2. Similarity, sup(be) = 3, and sup(ce) = 2. Second, MFI-TransSW generates one candidate 3-itemset (bce) according to Apriori property and uses bitwise AND operation to count the sup(bce) = 2, i.e., Bit(bc) AND Bit(be) AND Bit(ce) = 110. Because no new candidates are generated, the generation-then-test process is stopped. Hence, there are six frequent itemsets, (b), (c), (bc), (be),

CI2 in SW2
{(bc) | Bit(b) = 111 AND Bit(c) = 110}
{(be) | Bit(b) = 111 AND Bit(e) = 111}
{(ce) | Bit(c) = 110 AND Bit(e) = 111}

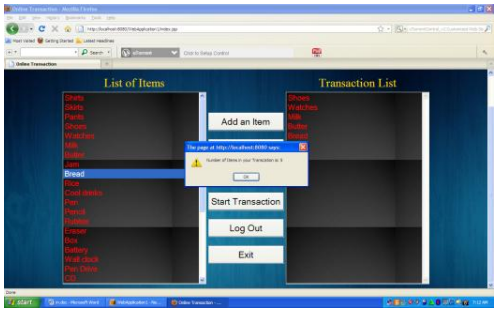
FI1 in TransSW2 (s = 0.6)	Support count
{(bce) | Bit(bce) = 110}	2

FI1 in TransSW2 (s = 0.6)	Support count
{(b) | Bit(b) = 111}	3
{(c) | Bit(c) = 110}	2
{(e) | Bit(e) = 111}	3

Table 3. Frequent itemset generation

6. Experimental Result

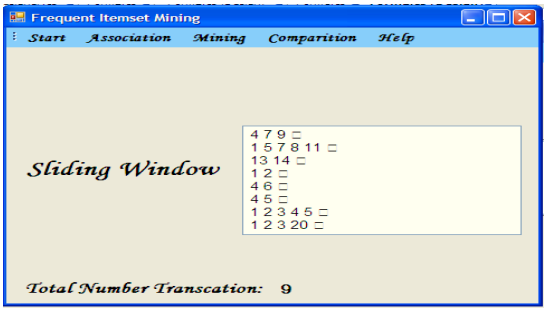
6.1 Window Initialization



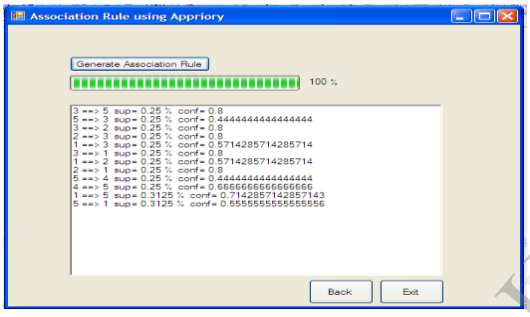
The window initialization phase is activated while the number of transactions generated so far in a transaction data stream is less than or equal to a user-predefined sliding window size w.

6.2 Bit- Sequence Method

In this phase, each item of the new incoming transaction is transformed into its bit-sequence representation.

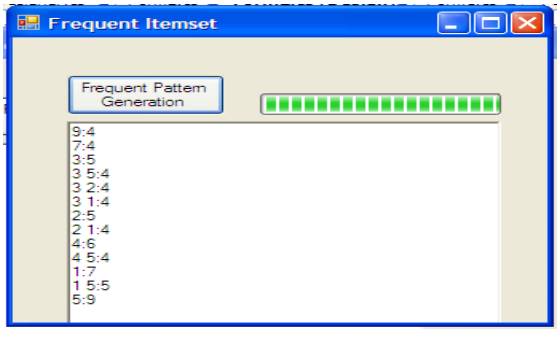


6.3 Window Sliding



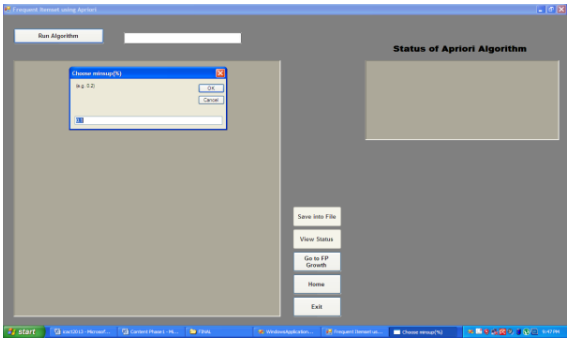
The window sliding phase is activated after the current sliding window TransSW becomes full. A new incoming transaction is appended to the current sliding window, and the oldest transaction is removed from the window.

6.4 Frequent Pattern Generation



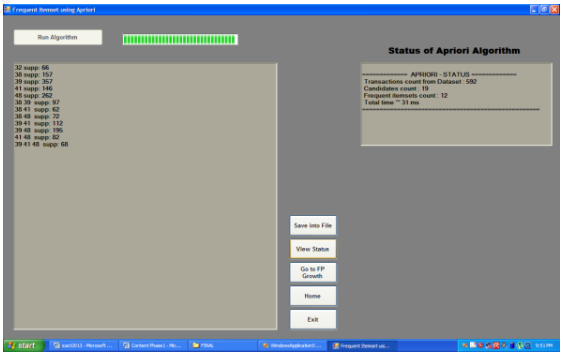
To find frequent itemsets on a data stream, we maintain a data structure that models the current frequent itemsets. We update the data structure incrementally.

6.5 Apriori Implementation



In this phase user defined the support count and window size. (e.g window size is 10, support count is 0.1).

6.5 Status of an Apriori

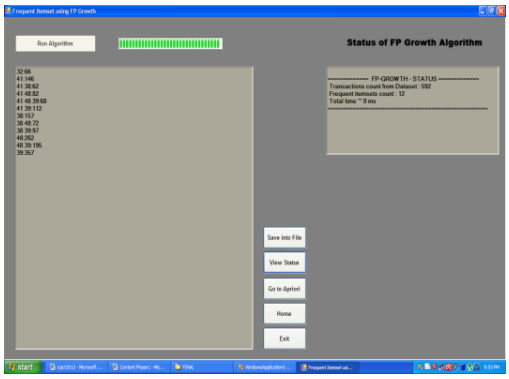


The Apriori Algorithms an influential algorithm for mining frequent item sets for boolean association rules. In computer science and data mining, Apriori is a classic algorithm for learning association rules.

Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). The algorithm attempts to find subsets which are common to at least a minimum number C (the cutoff, or confidence threshold) of the item sets.

6.6 FP Implementation

This phase generates the frequent patterns based on FP growth Algorithm. And also it generates the FP Tree to find out the frequent patterns.



The FP-growth algorithm is currently one of the fastest approaches to frequent item set mining. It is based on a prefix tree representation of the given database of transactions (called an FP-tree), which can save considerable amounts of memory for storing the transactions.FP-Growth is an algorithm for generating frequent item sets for association rules. This algorithm compresses a large database into a compact, frequent pattern– tree (FP tree) structure. FP – tree structure stores all necessary information about frequent itemsets in a database.

7. Conclusion

We proposed a data mining method for finding recent frequent items over an online data stream. An efficient single-pass method, called frequent itemset transaction sliding window, for mining the set of frequent itemsets over data streams with a transaction sensitive sliding window. Frequent itemset transaction sliding window not only attain highly accurate mining results, but also run significant faster and consume less memory than do existing algorithms for mining frequent itemsets from data streams within a sliding window, and also by comparing two algorithms, Apriori and FP growth for discovering all significant association rules between items in a large database of online transaction. Apriori based algorithm performs well for dense datasets and FP- Tree based algorithms performs well for sparse datasets. The Apriori algorithm scans the dataset repeatedly whereas the FP growth avoids the costly candidate set procedure and generates the highly condensed database called as FP tree.

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] S. Muthukrishnan , "Data streams: algorithms and applications". *Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms*, 2009.

[3]. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A.I. Fast discovery of association rules. In U.Fayyad et al. (eds), "Advances in Knowledge Discovery and Data Mining", *Menlo Park, CA: AAAI Press*, 307-328.

[4]. Webb, G.I. "Efficient search for association rules". *In Proceedings of the Sixth ACM-SIGKDD InternationalConference on Knowledge Discovery and Data Mining, New York*, NY: ACM, 99-107.

[5] 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.*

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

[7] Lucchese.C, S. Orlando, and R. Perego "Fast and memory efficient mining of frequent closed itemsets" *Knowledge and Data Engineering, IEEE Transactions*; January 2006.

[8] Han, J., Pei, J., & Yin, Y. "Mining frequent patterns without candidate generation". *In Proceedings of the 2007 international conference on management of data*, (pp. 1–12).

[9] Zhao.H, A. Lall, M. Ogihara, O. Spatscheck, J. Wang, J. Xu, "A data streaming algorithm for estimating entropies of od flows", *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007.

[10] Zhang.L, Z. Li, M. Yu, Y. Wang, Y. Jiang, "Random sampling algorithms for sliding windows over data streams", *Proc. of the 11th Joint International Computer Conference*, pp. 572–575, 2008.

[11] Pei.J, J. Han, and R. Mao "Closet: An efficient algorithm for mining frequent closed itemsets" *ACM SIGMOD International Workshop on Data Mining and Knowledge Discovery*, May 2007.

[12] Tatbul.N, S. Zdonik, "Window-aware load shedding for aggregation queries over data streams", *Proceedings of the 32nd international conference on Very large data bases*, 2006.

[13] Vitter.J.S, "Random sampling with a reservoir", *ACM Transactions on Mathematical Software (TOMS)*, v.11 n.1, pp.37–57, 1985.

[14] Indyk.P, D. Woodruff, "Optimal approximations of the frequency moments of data streams", *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pp.202–208, 2005.