IJIRSET

ISSN(Online): 2319 - 8753
ISSN (Print) :2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

An Efficient Sequential Frequent Pattern Analysis Using DBCA

K.Sasikala [1], P.Velusamy [2]

P.G. Student, Department of Computer Science and Engineering, Sengunthar Engineering College, Tiruchengode, Tamilnadu, India[1]

Assistant Professor, Department of Computer Science and Engineering, Sengunthar Engineering College, Tiruchengode, Tamilnadu, India[2]

ABSTRACT: In this work, the practical problem of frequent-itemset discovery in data-stream environments which may suffer from data overload. The main issues include frequent-pattern mining and data-overload handling. Therefore, a mining algorithm together with Separate dedicated overload-handling mechanisms is proposed. The algorithm DBCA (Dynamic Base Combinatorial Algorithm) extracts basic information from streaming data and keeps the information in its data structure.

The DBCA algorithm extracts base information from data streams in a dynamic way. More specifically, it keeps base information on a data stream with the size concerning the average length n of transactions. It could effectively manage data overload with the overload-handling mechanisms. Our results may leads to a possible solution for sequential frequent-pattern mining in dynamic streams, the Sliding window by pruning the excess of incoming data and dealing only with the trimmed data, not by processing on the full amount of incoming data. Depending on how overloading data can be trimmed, there may be various policies on load shedding, and we have described three such policies. The proposed policies, although possess different properties, have all been verified by the experiment to be effective.

KEYWORDS: DBCA Mining, frequent itemsets, sliding window, base information.

I. INTRODUCTION

Nowadays many commercial applications have their data presented in the form of continuously transmitted stream, namely data streams. In such environments, data is generated at some end nodes or remote sites and received by a local system (to be processed and stored) with continuous transmission. It is usually desirable for decision makers to find out valuable information hidden in the stream. Data-stream mining is just a technique to continuously discover useful information or knowledge from a large amount of running data elements. Like data mining in traditional databases, the subjects of data-stream mining mainly include frequent itemsets/patterns, association rules (Agrawal and Srikant, 1994), sequential rules, classification, and clustering. Through the data-stream mining process, knowledge contained inside the stream can be discovered in a dynamic way.

Data mining from data streams has three kinds of time models (or temporal spans) (Zhu and Shasha, 2002). The first one is landmark window model, in which the range of mining covers all data elements that have ever been received. The second one is damped/fading window model, in which each data element is associated with a variable weight, and recent elements have higher weights than previous ones. The third one is sliding window model, in which a fixed-length window which moves with time is given, and the range of mining covers the recent data elements contained within the window.

Due to the fact that early received data elements may become out of date and/or insignificant, i.e.,the timeliness factor, among the three models, the sliding window model is more appropriate for many data-stream applications such as finance, sales, and marketing. A data-stream mining system may suffer the problem of data overload, just like the case of over-demand electricity to a power supply system. A data stream is usually dynamic and its running speed may change with time. When the data transmission rate of the data-stream source exceeds the data processing rate of the mining algorithm of a mining system, e.g., during a peak period, the system is overloaded with data and thus unable to handle all incoming data elements properly within a time-unit. Furthermore, an overloaded system may work abnormally or even come into a crash. Accordingly, it is essential for a data-stream mining system to adequately deal with data overload and/or spikes in volume.

In this paper, we propose a load-controllable DBCA mining algorithm for discovering frequent patterns in transactional data streams. The mining algorithm works on the basis of combinatorial approximation (Jea and Li, 2009). To address the possible case of peak dataload at times, two dedicated overload-handling mechanisms are designed for the algorithm to manage overload situations with their respective means. With the load-controllable ability, a mining system with the proposed algorithm is able to work normally during high-data-load periods. Besides, according to our experimental data, the mining results (after overload handling) still possess reasonable quality in term of accuracy.

The rest of this paper is organized as follows. In Section 2, related work regarding data-stream frequent-pattern mining and data-overload handling is described. Section 3 gives the symbol representation, problem definition, and goal of this research. In Section 4, a mining algorithm together with two overload-handling mechanisms is proposed and explained in detail. Section 5 presents the experimental results with analyses. In Section 6, discussions are given on the overload-handling mechanisms. Finally, Section 7 concludes this work

II. RELATED WORK

Frequent-pattern mining from data streams is initially limited to singleton items (e.g., Charikar et al., 2004; Fang et al., 1998). Lossy Counting (Manku and Motwani, 2002) is the first practical algorithm to discover frequent itemsets from transactional data streams.This algorithm works under the landmark window model. In addition to the minimum-support parameter (ms), Lossy Counting also employs an error-bound parameter, ε, to maintain those infrequent itemsets having the potential to become frequent in the near future.

The Lossy Counting processes the stream data batch by batch, and each batch includes bucket(s) of transactions. With the use of parameter ε, when an itemset is newly found, Lossy Counting knows the upper-bound of count that itemset may have before it has been monitored. As a result, the error in itemset's frequency count is limited and controllable. According to the experiments conducted by Manku and Motwani (2002), Lossy Counting can effectively find frequent itemsets over a transactional data stream. This algorithm is a representative of the ε-deficient mining methods and has many related extensions. For example, based on the estimation mechanism of Lossy Counting, a sliding window method for finding recently frequent itemsets in a data stream is proposed by Chang and Lee (2004).

A different type of method is to process on stream elements within a limited range and offer no-false mining answers. DSTree (Leung and Khan, 2006) is a representative approach of one such type, which is designed for exact (stream) mining of frequent itemsets under the sliding window model. Given a sliding window, DSTree uses its tree structure for capturing the contents of transactions in each batch of data within the current sliding window.

More specifically, DSTree enumerates all itemsets (of any length) having ever occurred in transactions within the current window and maintains them fully in its tree structure. The update of tree structure is performed on every batch, while the mining task is delayed until it is needed. According to the experimental results given by Leung and Khan (2006), mining from DSTree achieves 100% accuracy (since all itemsets having ever occurred are stored and monitored). However, because this method needs to enumerate every itemset in each of the transactions, its efficiency is badly affected by the great computation complexity. As a result, it can hardly manage with a data stream consisting of long transactions.Besides DSTree, there are other methods belonging to the type of exact stream mining. Li and Lee (2009) proposed a bit-sequence based, one-pass algorithm, called MFI-TransSW, to discover frequent itemsets from data-stream elements within a transaction-sensitive sliding window. Every item of each transaction is encoded in a bit-sequence representation for the purpose of reducing the time and memory necessary for window sliding; to slide the window efficiently, MFI-TransSW uses the left bit-shift technique for all bit-sequences. In addition, Tanbeer et al. (2009) proposed an algorithm called CPS-tree, which is closely related to DSTree, is proposed to discover the recent frequent patterns from a data stream over a sliding window. Instead of maintaining the batch information (of the sliding window) at each node of the tree structure (as DSTree does), CPS-tree maintains it only at the last node of each path to reduce the memory consumption. The authors also introduce the concept of dynamic tree restructuring to produce a compact frequency-descending tree structure during runtime.

Another type of method is to perform the mining task, i.e., discover frequent itemsets, through a support approximation process. One feasible approach is to apply the idea of Inclusion–Exclusion Principle in combinatorial mathematics (Liu, 1968), whose general form is shown in Eq. (1), to data-mining domain for support calculation, and further apply the theory of Approximate Inclusion–Exclusion (Linial and Nisan, 1990) for approximate-count computation. This mining approach is called CA (combinatorial approximation).

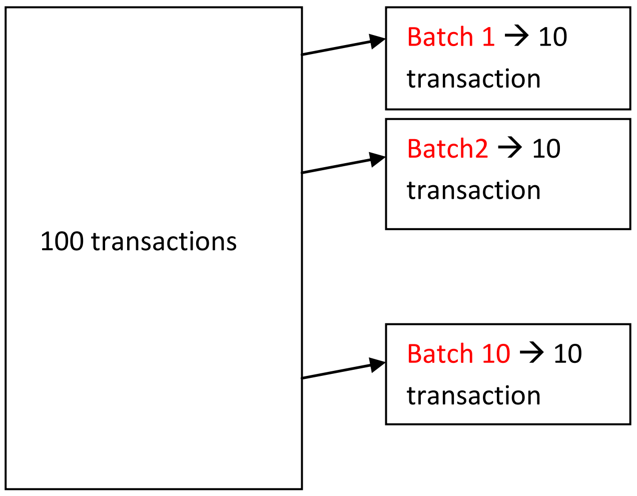
The DSCA algorithm (Jea and Li, 2009) is a typical example of such an approach. DSCA operates under the landmark window model and monitors all itemsets of lengths 1 and 2 existing in the data stream. It performs the mining task by approximating the counts of longer itemsets (based on the monitored itemsets) and returning the frequent ones as the mining outcome. According to the experimental data given by Jea and Li (2009), the performance of DSCA is quite efficient. Besides DSCA, the SWCA algorithm (Li and Jea, 2011) is also an example of CA-based approach. This method is under the sliding window model and has made an effort to improve the accuracy of support approximation.

III. PROPOSED METHODOLOGY

Our **FP-tree-based DBCA** mining adopts a pattern-fragment growth method to avoid the costly generation of a large number of candidate sets, and a partitioning-based, divide-and-conquer method is used to decompose the mining task into a set of smaller tasks for mining confined patterns in conditional databases, which dramatically reduces the search space. Our DBCA performance study shows that the FP-growth method is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than uses a technique to quickly prune candidate frequent item sets in the item set lattice. The technique gathers Pattern information for a node used to find the next node during depth-first mining in the lattice. Items are dynamically reordered based on the tail information. DBCA is about 10 times faster than UPS and others FP generation algorithms.

INPUT TRANSACTION

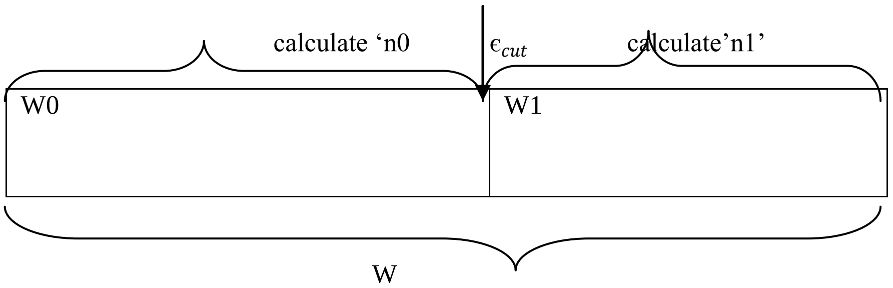
a. The input should contain large set of transaction and should be divided into batches
b. Each batch should contain some set of transaction
c. Each batch of transaction should be fed into the sliding window (w0 and w1)
d. E.g., if there are 100 transaction means, it can be divided into 10 batches, so that each batch contains 10 transaction



.

IV. DBCA BASED SLIDING WINDOW

a. The sliding window concept is used here
b. The total window size is set as ‘w’
c. The window should be divided into two sub-windows ‘w0’ and ‘w1’
d. The sub-windows should be dynamically change by using the formula



e. Here ‘n0’ denotes the maximum length of an itemset in ‘W0’ (eg., consider there is 1st batch of transaction with its corresponding itemsets in ‘W0’. Among those transaction, if the maximum length of an itemset is 4, which is {milk, bread, cheese, butter}). ‘n0’=4.
Then ‘n1’ denotes the maximum length of an itemset in ‘W1’ (eg., consider there is 2nd batch of transaction with its corresponding itemsets in ‘W1’.Among those transaction, if the maximum length of an itemset is 3, which is {milk, bread, cheese}. ‘n1’=3

‘n’= ‘n0’+ ‘n1’

Step1: calculate ‘n0’and ‘n1’ (max.. length of itemset in ‘w0’ and ‘w1’)

Step 2: calculate harmonic mean ‘m’

$$m = \frac{1}{\frac{1}{n0}+\frac{1}{n1}}$$

Step 3: calculate the approximate confidence value ‘δ′

$$\delta' = \frac{\delta}{n} \text{ (consider } \delta= 40\% \text{ or } 0.4) \quad n=n0+n1$$

Step 4: calculate ‘ϵ_{cut}’ in which it is used for partitioning the sub-windows dynamically.

$$\epsilon_{cut} = \sqrt{\left(\frac{1}{2m} \times ln\frac{4}{\delta'}\right)}$$

Let us consider the sliding window size ‘W’=20.

Initially, we have to fix the ‘W0’ size as constant (eg., W0=10 or 11 or anything) and it should vary dynamically for remaining set of transaction. It should be user defined

First, the dataset enters the sliding window. At beginning W0=10, so that only the first batch will be stored in W0 and the next batch will be stored in ‘W1’

From ‘W0’ and ‘W1’, we have to calculate ‘n0’ and ‘n1’ and ϵ_{cut}

We have to calculate the candidate-1 itemset and candidate-2 itemset for ‘W0’ and should be stored as ‘BASE INFORMATION’ (ie., base.xls)

Once ‘W0’ finished finding its base information, then the 1st batch of transaction in ‘W0’ should be removed and stored in another database (ie., remove.xls). Then the next batch of transaction will be stored in ‘W0’

Now we have to calculate ‘ ϵ_{cut} ’ for ‘W1’ and new ‘W0’ based on ‘n1’ and new ‘n0’.Once ‘W1’ have finished finding its base information, the values should be updated in the same base information (ie., base.xls) and then the current batch of transaction in ‘W1’ should be removed and stored in remove.xls and next set of transaction should be stored in ‘W1’This process should continue until the end of the dataset.

V. DBCA MODULES

Base Information Analysis

In the base information analysis, it can mine the complete set of frequent itemsets, based on the completeness of patterns to be mined: we can distinguish the following types of frequent itemset mining, given a minimum support threshold the co-efficient, which refers to the variety of items, including first or most significant itemset .the combinatorial represents the itemset 'j' represents the length of an itemset. If the length of an itemset is 2(j=2) , it contains 1-itemset and 2-itemset (i=1,2) 'm' represents the target itemset length, m=k+1.Here 'm' denotes the itemset length that we are going to find the approximate count. (eg., if k=2, m=3) 'k' represents the base information size. In the base information, if k=2 means, it denotes that, it contains 1-itemset and 2-itemset. a_{ij} represents the ith itemset of jth itemset to use for finding approximation count.

Approximation Count Calculation

In this module is to generate the maximal frequent itemsets with minimum effort. In place of generating candidates for determining maximal frequent itemsets as done in other methods, this module adapt the concept of partitioning the data source into segments and then mining the segments for maximal frequent itemsets. Additionally, it reduces the number of scans over the transactional data source to only two. Moreover, the time spent for candidate generation is eliminated. This algorithm involves the following steps to determine the MFS from a data source:

1. Segmentation of the transactional data source.
2. Prioritization of the segments.
3. Mining of segments.

Frequent Item list Generation

This module to find frequent itemsets using candidate generation, the process of generating candidate itemsets is done using a depth-first search, and the process can be represented as a candidate itemset tree. With each step down the tree, a single item is extended onto an itemset. As the itemsets grow larger and larger, the percentage of customers who have the itemset, or the support %, will grow smaller and smaller. Eventually, this support value will go below the minimum support required for an itemset to be deemed frequent. it looking at the lexicographic tree, it is possible to draw a line that crosses all points at which an occurrence of an itemset being extended goes from frequent to infrequent. All itemsets directly above this line are termed the maximal frequent itemsets. By the DBCA principle, no itemset extensions below this line can be frequent since they all contain other itemsets within them that were found to be infrequent.

Skip and Complete Technique

This module is to stores the transactional database as a series of vertical bitmaps, where each bitmap represents an itemset in the database and a bit in each bitmap represents whether or not a given customer has the corresponding itemset. Initially, each bitmap corresponds to a 1-itemset, or a single item. The itemsets that are checked for frequency in the database become recursively longer and longer, and the vertical bitmap representation works perfectly in conjunction with this itemset extension. For example, the bitmap for the itemset (a,b) can be constructed simply by performing an AND operation on all of the bits in the bitmaps for (a) and (b). Then, to count the number of customers that have (a,b), all that needs to be done is count the number of one bits in the (a,b) bitmap equals the number of customers who have (a,b). Clearly, the bitmap structure is ideal for both candidate itemset generation and support counting.

Group Count Technique

This module evaluates the performance of DBCA with the load shedding mechanism during data overloading situations. The three proposed policies for the mechanism are separately tested and then compared together. Both the normal DBCA (that is, without load-shedding operation) and Lossy Counting (which has no load-shedding function) are also tested as the control group for confirming the effects of load shedding.

VI. CONCLUSION

In real-life data streams, the data transmission rate usually varies with time; at times it may beyond the data processing capability of a mining algorithm/system, which leads to the serious situation of data overload. A key contribution of this paper is to provide users with a feasible solution to frequent-pattern discovery in dynamic data streams which are prone to data overload. More specifically, a count approximation-based mining algorithm, DBCA, together with two dedicated mechanisms for overload management, is described and proposed. The DBCA algorithm extracts synopsis from received stream elements as base information, and carries out the mining task when requested through an approximating process. One remarkable feature of DBCA is that it can maintain dynamical-sized base information according to different(batches of) received data. The two mechanisms, namely processing acceleration and data-load shedding, are prepared for possible data overload situations. By running one of the mechanisms when the buffer is overloaded with data, DBCA could cope with the overload by means of either increasing its throughput or decrease the data volume. Nevertheless, so long as data overload is not the common case, that is, it may take place sometimes but not all the time, a mining system with our load-controllable method could operates normally under different data-load degrees. In other words, the system will possess the nice feature of durability.

REFERENCES

1. M. Muzammal and R. Raman, "Mining sequential patterns from probabilistic databases," in *Proc. 15th PAKDD*, Shenzhen, China, 2011.
2. D. Tanasa, J. A. López, and B. Trousse, "Extracting sequential patterns for gene regulatory expressions profiles," in *Proc. KELSI*, Milan, Italy, 2004.
3. J. Pei *et al*., "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. 17th ICDE*, Berlin, Germany, 2001.
4. R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. 11th ICDE*, Taipei, Taiwan, 1995.
5. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. C. Hsu, "FreeSpan: Frequent pattern-projected sequential pattern mining," in *Proc. 6th SIGKDD*, New York, NY, USA, 2000.
6. R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. 5th Int. Conf. EDBT*, Avignon, France, 1996.
7. Z. Zhao, D. Yan, and W. Ng, "Mining probabilistically frequent sequential patterns in uncertain databases," in *Proc. 15th Int. Conf. EDBT*, New York, NY, USA, 2012.
8. N. Pelekis, I. Kopanakis, E. E. Kotsifakos, E. Frentzos, and Y. Theodoridis, "Clustering uncertain trajectories," *Knowl. Inform. Syst.*, vol. 28, no. 1, pp. 117–147, 2010.
9. L. Sun, R. Cheng, D. W. Cheung, and J. Cheng, "Mining uncertain data with probabilistic guarantees," in *Proc. 16th ACM SIGKDD*, Washington, DC, USA, 2010.
10. C. C. Aggarwal, Y. Li, J. Wang, and J. Wang, "Frequent pattern mining with uncertain data," in *Proc. 15th ACM SIGKDD*, Paris, France, 2009.
11. Q. Zhang, F. Li, and K. Yi, "Finding frequent items in probabilistic data," in *Proc. ACM SIGMOD*, Vancouver, BC, Canada, 2008.
12. T. Bernecker, H. P. Kriegel, M. Renz, F. Verhein, and A. Zuefle, "Probabilistic frequent itemset mining in uncertain databases," in *Proc. 15th ACM SIGKDD*, Paris, France, 2009.
13. C. K. Chui, B. Kao, and E. Hung "Mining frequent itemsets from uncertain data," in *Proc. 11th PAKDD*, Yichang, China, 2007.
14. J. Yang, W. Wang, P. S. Yu, and J. Han, "Mining long sequential patterns in a noisy environment," in *Proc. ACM SIGMOD*, Madison, WI, USA, 2002.