Journal of Information and Operations Management
ISSN: 0976–7754 & E-ISSN: 0976–7762 , Volume 3, Issue 1, 2012, pp-153-157
Available online at http://www.bioinfo.in/contents.php?id=55

FINDING FREQUENT ITEM SETS FROM DATA STREAMS WITH SUPPORTS ESTIMATED USING TRENDS

SHANKAR B. NAIK[1] AND JYOTI D. PAWAR[2]

[1]Department of Computer Science & IT, Dhempe College, Panaji Goa, India.
[2]Department of Computer Science & Technology, Goa University, Goa, India.
*Corresponding Author: Email- xekhar@gmail.com, jdp@unigoa.ac.in

Received: December 12, 2011; Accepted: January 15, 2012

Abstract- Finding frequent itemsets in data streams is a challenging task. Traditional data mining algorithms mainly focus on finding frequent itemsets in static datasets. Those which execute on data streams give importance to recent data or assume uniform distribution of data. Also, supports of itemsets are estimated by assigning fixed values where assumptions are required. In this paper we propose an approach that estimates the support of itemsets based on the trends of the itemsets seen in the data stream.
Keywords- Data Streams, Itemsets, Minimum Support, Sliding Window, Support

Citation: Shankar B. Naik and Jyoti D. Pawar (2012) Finding Frequent Item sets from Data Streams With Supports Estimated Using Trends. Journal of Information and Operations Management ISSN: 0976–7754 & E-ISSN: 0976–7762, Volume 3, Issue 1,pp-153-157.

Introduction

Nowadays, a growing number of applications generate data streams [2] [8], such as telecom call records, web clickstreams, data collected in sensor networks, etc,. As the notion suggests, a data stream can roughly be thought of as an ordered sequence of data items, where the input arrives more or less continuously as time progresses [3][6].

Mining frequent itemsets in a data stream is a challenging task. In order to obtain frequent itemsets from a set of data, entire data set should be available at hand for processing. Such analysis can be done on static data sets very easily and accurately. It is not true in the case of a data stream. Data streams are large data sets that cannot be entirely stored in memory for processing. Also, the rate at which data is generated is very high [6]. Finding frequent itemsets involves many passes over the entire data set. This is not possible with a data stream as the entire data stream cannot be stored in memory.

In this paper we discuss how to find frequent itemsets in data streams. Data stream is divided into segments of pre-decided size, input by user. These segments are stored in memory one at a time and frequent itemsets are generated for each stored segment. The frequent itemsets of each segment are stored for future computations and the stored segment is then removed from the memory. These stored results are used in future to generate the frequent itemsets for the entire or part of data stream.

An itemset may not be frequent in a segment. In this case, it is discarded and is not stored for that segment. The same itemset may appear to be frequent in other segments. While calculating the support of the itemset for the entire data stream, it is important to know the support of the itemset in segments, in which the itemset

is not frequent. The support of the itemset in the segment, in which it is not frequent, is estimated by analyzing the trends shown by the itemset in other parts of the stream. This introduces an error in the estimated results.

The frequent itemsets generated for segments are stored in partitions. Each partition corresponds to a segment of the data stream. As the data stream increases it is difficult to store results of all the segments of the data stream. Moreover, the size of the data steam is not known a priori. Thus the total number of segments of data stream cannot be predicted in advance. If the data stream tends to be never ending then number of segments simply grow and explode. It is required to store the frequent itemsets in such a way that the amount of memory required should increase in large.

The method proposed in [7], uses a time tilted window in which results of various segments are merged together as new segments come in. It provides results in details for the latest data segments, whereas the results of older data are merged together. Frequent itemsets are generated for different levels of time granularities. It is assumed that the older data results are less significant than the recent data results. Hence as the data stream segment gets older and older its details are lost. This assumption may not be true in all the cases. In some cases data in every part of the data stream is relevant. In such cases merging should be done in a non partial manner such that every data segment is given equal importance.

When results of different segments of data stream are merged together, many estimations, incase of non-frequent itemsets, are involved. These estimations contribute to error in the end results. As the data streams evolves all the partitions get occupied at a point in time. When no more partition is found to store the frequent itemsets of the next segment, then only, two adjacent partitions could be merged together to leaving one partition free to accommodate the frequent itemsets of the new segment. At the end, even though not in detail, results will be available for all segments, of same size, of data streams. The level of details and accuracy depends on the memory available to store the partitions.

The above stated requirements and constraints have formed the motivating factors for our work.

Background

Data Streams

A data stream system may constantly produce huge amounts of data [3]. Regarding, aspects of data storage, management and processing, the continuous arrival of data items in multiple, rapid, time-variant, and potentially unbounded streams raises new challenges and research problems. Indeed, it is usually not feasible to simply store the arriving data in a traditional database management system in order to perform operations on that data later on. Rather, stream data must generally be processed in an online manner in order to guarantee that results are up-to-date and that queries can be answered with small time delay. The development of corresponding stream processing system is a topic of active research [3] [1].

The data stream model assumes that input data are not available for random access from disk or memory, such as relations in standard relational databases, but rather differs from the standard relation model in the following ways [3][1] –

1)The elements of a stream arrive incrementally in an "online" manner. That is, the stream is "active" in the sense that the incoming items trigger operations on the data rather than being sent on request

2)The order in which elements of a stream arrive are not under the control of the system

3) Data streams are of potentially unbounded size

4) Data stream elements that have been processed are either discarded or archived. They cannot be retrieved easily unless being stored in memory, which is typically small relative to the size of the stream. Stored information about past data is often referred to as synopsis

5) Due to limited resources (memory) and strict time constraints, the computation of exact results will usually not be possible. Therefore, the processing of stream data does commonly produce approximate results [4].

Frequent Itemset Mining

Data mining, also known as knowledge discovery in databases, aims at the discovery of useful information from large collections of data [1] [5] [9]. The discovered knowledge can be rules describing the properties of the data, frequently occurring patterns, clustering of objects in the database, and so on, which can be used to support various intelligent activities, such as decision making, planning, and problem solving.

The data mining model adopted in this paper for association rules is based on the minimum support confidence framework established by Agrawal et al [1]. The terminology used in the paper is described in the following paragraph.

Let $I=\{i_1,i_2,\ldots,i_N\}$ be a set of N distinct literals called items, and D a set of transactions over I. Each transaction contains a set of items $i_1,i_2,\ldots,i_k \in I$. Each transaction has an associated unique identifier called Transaction IDentifier(*TID)*.

A set of items is called an itemset. Each itemset X has an associated statistical measure called support. An itemset is a set consisting of atleast one item.

Support of an itemset X, denoted as *Supp(X),* is the fraction of transactions in D, that contain all the items in X, calculated as *number(X)/ size(D),* where *number(X)* is the number of transactions that support the itemset and *size(D)* is the total number of transactions in D.

Frequent itemset is any itemset whose support in D is no less than user input minimum support threshold denoted by minsup.

Our Proposed Approach

Let D be a data stream of size n, having itemsets as its elements. T_i is the i^{th} transaction consisting of itemsets in D. The interval *[i,j]* gives the range of data elements of D, for $j>=i$. *S[i.j]* is a segment of D having elements belonging to *[i,j].* Let SW be the sliding window, of size w over D, which means, at a time w elements of D can be stored in SW. Size of SW is never less than size of any segment of the data stream. PW is the set of partitions $P_1,P_2,\ldots P_p$, where p is the size of PW. Every P_i is a collection of frequent itemsets for some segment of D as shown in Fig.1. $P_i[a,b]$ is the i^{th} partition in PW storing frequent itemsets of segment *S[a,b].* $|P_i|$ denotes the size of segment associated with P_i, $|P_i|=b-a+1$.

Initially, the first w elements of D are stored in SW as they are generated. At this point in time, SW has the first w elements of D, i.e.

elements of segment *S[1,w].* A simple apriori algorithm is executed on *SW.* The frequent itemsets generated are stored in P_1 of *PW.* In the mean time *SW* is loaded with the next *w* elements of the data stream. Frequent itemsets are generated for the second segment, *S[w+1,2w]*, elements present in *SW* and are stored in partition P_2 of *PW*. This is repeated until *D* is over, or all *w* partitions of *PW* are filled in, whichever is earlier.

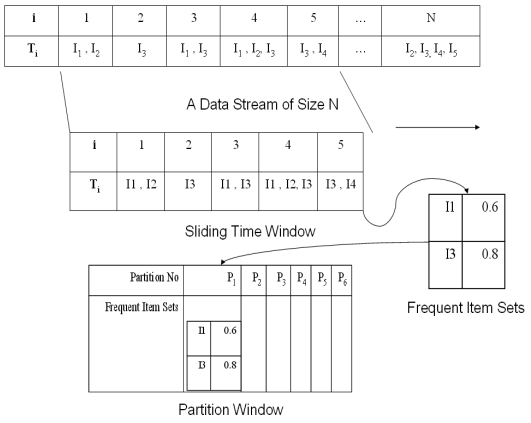

Fig.1- Data stream, sliding window and partition window

If all the partitions of *PW* are full then, the process of merging (section 3.1) is executed on *PW* to accommodate the frequent itemsets of the subsequent segments of the data stream *D*.
At any point of time, frequent itemsets, for the entire data stream or a part of a stream can be generated from the data available in the partition window *PW* using the technique given in section III.D.

Merging
Merging of partitions in *PW* is done when all *p* partitions of *PW* are filled in and there is no empty partition available in *PW* to store the frequent itemsets of the next *w* elements of D, i.e.of the segment *S [p*w+1,w*(p+1)]*, which are present in *SW*.
Merging begins by selecting two adjacent partitions, P_i and P_{i+1}, from partition window. The resultant partition, *P*, has all the itemsets that are present in either or both of the selected partitions. *P* has two kinds of itemsets, those which are present in both, and those which are present in only one of the partitions, i.e. P_i or P_{i+1}. The support of each itemset in *P* is calculated in one of the following two ways.

Itemset present in both partitions
Let $X \epsilon P_i, P_{i+1}$ be an itemset that is present in both partitions. Let $Supp\_P_i(X)$ be the support of *X* in P_i. The support of *X* in *P*, *Supp_P(X),* is calculated as

$$(|P_i| * Supp\_P_i(X) + |P_{i+1}| * Supp\_P_{i+1}(X)) / (|P_i| + |P_{i+1}|) \quad (1)$$

Itemset present in only one partition
Let *X* be an itemset that is present in only one of the selected partitions. Let P_j be the partition not containing *X.* In such a case, *j* is either, *i* or *i+1.* Support of *X* in P_j is not known, which means it is less than *minsup*. Let $Supp\_P_j(X)$ be the support of *X* in P_j, which is not known and is to be estimated.\

Support Estimation
We calculate value of $Supp\_P_j(X)$ by seeing the trends of *X* in the neighboring partitions using the following expression -

$$Supp\_Pj(X) = minsup * trend \quad (2)$$

The term *trend* describes the average support of *X* in the neighbouring partitions. The value of *trend* is calculated as

$$trend = (|Pj-1| * Supp\_Pj-1(X) + |Pj+1| * Supp\_Pj+1(X)) / (|Pj-1| + |Pj+1|) \quad (3)$$

The neighbouring partitions, P_{j-1} or P_{j+1} may support *X*. In this case, support of X in the partitions not supporting X is taken as *minsup/2*.

Final Support Calculation
Supp_P(X) is calculated using the method mentioned in III.A.1.

Update Partition Window
After merging partitions P_i and P_{i+1} to obtain *P* as the new partition, P_i and P_{i+1} are removed from *PW* and are replaced by *P.* All the partitions followed by P_{i+1} in *PW* are shifted one partition left resulting in empty partition P_p. The frequent itemsets of the new segment are stored in partition P_p of *PW*.

Selection of partitions for merging
Two adjacent partitions are selected for merging. Selection of partition is done in the following way.
The oldest pair of adjacent partitions that has not been merged is selected for merging, i.e. partitions at same level are selected. Two partitions *Pi* and *Pj* are said to be at the same level if $|Pi|=|Pj|$. In this method merging begins first for the oldest partitions and continues for every pair of partitions that has not been considered for merging for that level until the last partition in *PW*. After the last partition, merging begins with the old partitions and continues till the end. This process is repeated until the data stream gets over.
Sometimes, the last partition may have to be merged directly with the newly generated results of the latest itemsets in *SW*. This is done until the level of last partition becomes the same as the level of all the other partitions in *PW*.
In this kind of merging, the older partitions in *PW* will have interval twice that of the later partitions, except the last one, in *PW*. Merging done is uniform and gives equal importance to the partitions of *PW*.

Deducing Frequent Itemsets from Partition Window PW
At any instance, each partition P_i in *PW* has frequent itemsets for some part of *D*.
At this moment the number of partitions in the partition window *PW,* are either *p* or less than *p* where *p* is the size of partition window *PW*. The partitions in *PW* provide frequent itemsets along with their supports for their corresponding segments. Each partition $P_i[a,b]$ in *PW* provides frequent itemsets for the segment *S[a,b]* of *D*.
Besides this, frequent itemsets can also be generated for the entire or a part of *D*. Let *Interval[a,b]* denote the range of data elements of *D* for which frequent itemsets are to be generated. This is done from the data stored in partitions corresponding to the segments of the data stream which fall in the *Interval[a,b].* If *a*=1 and *b*=*n*, then the frequent itemsets are generated for the entire data stream *D*.
Let the partition window *PW* have *p* partitions where each partition has frequent itemsets generated for their corresponding segments. Let *Interval[i,j]* be the range for which the frequent itemsets are to

be generated. Let Global Frequent Itemset(*GF)* be the set of itemsets present in atleast one of the partitions, for the *Interval[i,j]*, of the partition window *PW*. Thus *GF* has the itemsets that were frequent for atleast 1 stream segment *S[a,b],* where *i<=a<=b<=j*. *GF* has the potential frequent itemsets for *Interval[i,j]* of the data stream *D*.
For every itemset *I* in *GF*, its support for *Interval[i,j]* is calculated in either of the following ways.

Itemset Present in all Partitions
If the itemset *I* is present in all the partitions in the *Interval[i,j]* then its support is the average of all its supports in the partitions for *Interval[i,j]*. It is calculated using the formula
Suppt(I)=∑SupptPi(I) / Np (4)
for all *Pi* for *Interval[i,j]*. Where $SuppP_i(I)$ is the support of itemset *I* in the partition P_i and N_p is the number of partitions for the *Interval [i,j]* .

temset present in not all Partitions
If the itemset is absent in atleast one partition it means that the itemset *I* is not frequent for that partition. Hence we have no idea about the support of the itemset for that partition that does not contain the itemset *I*. In order to calculate the support of the itemset *I* for the *Inteval[i,j]* it is required to know the support of the itemset *I* even in the partitions that do not contain *I*, which is not known as *I* was not frequent in the segment corresponding to those partition. Thus it is necessary to estimate the support of *I* in non-supporting partitions. The support of itemset *I* in the non-supporting partitions can be estimated to be less than the minimum support threshold minsup. But the exact and precise support of itemset *I* in non supporting partitions cannot be known. In order to estimate the support of itemset *I* in such partitions we use the trends that are shown by the itemset in other partitions containing the itemset *I*. The support of itemset *I* in non supporting partitions is calculated as the product of minsup and the average of all the supports of itemset *I* in the partitions, containing *I*, using the following formula.
$Supp(I)=minsup*\sum Supp_i(I)/N_p$ (5)
for all P_i, in *PW*, containing the itemset *I*. Where $SuppP_i(I)$ is the support of itemset *I* in the partition P_i and N_p is the number of partitions, in *PW*, containing *I*.
The partition window *PW* now has supports in every partition for all the itemsets of *GF*. The over all supports for itemsets for the entire stream can be calculated from the values in the partition window *PW* using the method specified above in III.D.1.

Expermental Results
In this section, we show preliminary experimental results to test the effectiveness of our proposed approach. We show some preliminary experiments on synthetic dataset *D1* and real dataset *D2*. Dataset *D2* has its itemsets as subject combinations opted by students of an institution. Each item is the subject opted by the student. Because our experiment goal is to demonstrate the effectiveness of our approach, all datasets are treated as static.
The characteristics of these datasets is given in table 1 -

Table 1- Characteristics of Datasets

Dataset	# of items	# of records
D1	5	1.5K
D2	5	1.0K

Error Calculation
In our experiment, we set size of sliding window *w*=10, number of partitions in *PW*, *p*= 10, *minsup*=0.4. The results are summarized in table 2. Error for each itemset was calculated as the difference between the support of itemset calculated by our approach and the actual support of the itemset calculated by applying apriori algorithm to the entire data as whole.

Table 2- Experiment Results for minsup=0.4

Dataset	# of frequent itemsets	Average error
D1	6	0.05648
D2	5	0.01

Average Error versus minimum support
Experiment was performed to check the variation in average error in support of all itemsets with respect to the change in minimum support (Fig. 2) for the same values of other parameters as in section IV.1.
Initially for small values of minimum support the average error is small. As the value of minimum support increases the average error in supports of itemsets also increases. We see an overall increase in error but average error is seen to be decreasing for small range of minimum support. In all the cases the average error is less than the minimum support. The trends shown need not to be always true, as the error entirely depends upon the actual values in the data stream. But always the error will be less than the minimum support

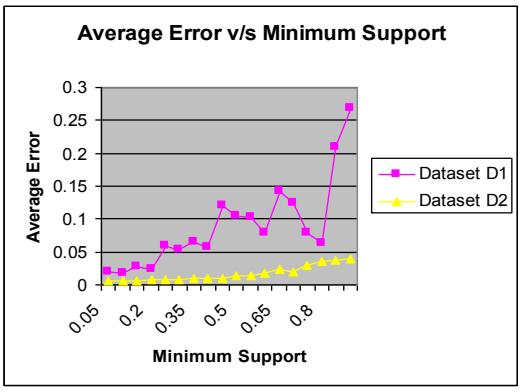

Fig. 2- Change in average error with change in minimum support

Error versus number of partitions
Experiment was done to calculate the variation in error in support of an itemset with respect to the change in the number of partitions (Fig. 3) for the same values of other parameters as in section IV.1.

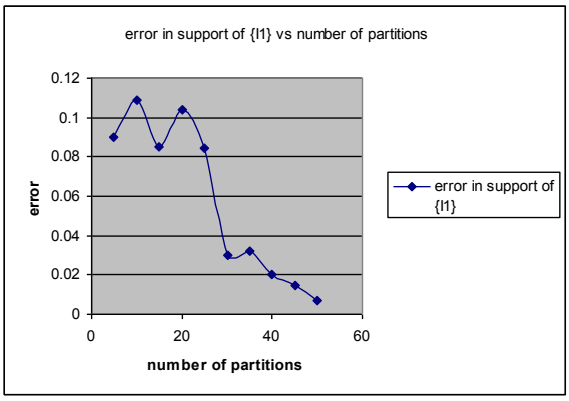

Fig. 3- Change in average error with change in number of partitions

The above curve shows the changes in average error caused by the change in the number of partitions in the partition window. For small number of partitions the average error in support is high. This is the result of large number of estimates done due to more merging of partitions. It is because of the smaller number of partitions, that merging operation is more frequently done. As the number of partitions increases, the number of estimates reduces which results in smaller values of average error.

Conclusion

Most research efforts on dealing with large database in data mining have concentrated on static databases. The approach presented in this paper is suitable for data streams.

To find out frequent itemsets in data streams we have proposed the method by dividing the data stream into segments of record that can be stored in the memory for analysis. Only the frequent itemsets generated for every segment of the stream are stored for further analysis. The method uses information about the itemset from those segments (partitions) of data stream that have the itemset as frequent and estimate its probability in the segments that have not generated the itemset as frequent. Due to these estimations some error is introduced in the final results.

References

[1] Agrawal R. Imielinski T. and Swami A. (1993) *ACM SIGMOD Conf. Management of Data*, pp. 207-216.

[2] Babcock B., Babu S., Datar M., Motwani R. and Widom J. (2002) *PODS*, pp. 1–16.

[3] Beringer J. and Hullermeier E. *Data & Knowledge Engineering*.

[4] Considine J. Li F., Kollios G. and Byers J.W. (2004) 20*th IEEE Int'l Conference on Data Engineering*.

[5] Chen M., Han J. and Yu P. *IEEE Trans Knowledge and Data Eng*., vol. 8, no. 6, pp. 866-881.

[6] Das A., Gehrke J. and Riedewald M. (2003) *ACM SIGMOD International Conference on Management of Data*, pp. 40–51.

[7] Giannella C., Han J., Pei J., Yan X., Yu P.S., *Mining frequent patterns in data streams at multiple time granularities*.

[8]Golab L. and Ozsu M.T. (2003) *SIGMOD*., 32(2):5–14.

[9]Srikant R. and Aggrawal R. (1997) *Future Generation Computer Systems*, vol. 13, pp. 160-180.