ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

Discovering Locally Frequent Patterns from Temporal Data

Fokrul Alom Mazarbhuiya

College of Computer Science and IT, Albaha University, Albaha, KSA

Abstract: Mining patterns from large dataset is an interested data mining problem. Many methods have been developed for this purpose till today. Most of the methods considered the time attributes as one of the attributes like others. However taking the time attribute into account separately the patterns can be extracted which cannot be extracted by normal methods. These patterns are termed as temporal patterns A couple of works have already been done in mining temporal patterns. In this article, we propose an efficient method of mining locally frequent patterns from temporal datasets. The efficacy of our method is established using experiments carried on a real life datasets and a synthetic datasets. Then we make comparative studies with a well established method and found that our method outperforms at least one well-known method.

Keywords: Data Mining, Frequent patterns, temporal patterns, locally frequent patterns.

1. INTRODUCTION

The problem frequent item set mining is the most researched field of frequent pattern mining. The problem is associated with association rule mining in market basket data and is formulated by Agarwal et al [1]. There are a number of algorithms proposed till today for mining such datasets. A-priori algorithm [2], is one of the most popular algorithms. But market basket data are usually temporal in nature e.g. when a transaction happens the time of transaction is also recorded with the transaction. Considering the time features of such datasets, some interesting patterns can be extracted which otherwise cannot be extracted. In [3], Ale et al proposed a method of extracting association rules which hold throughout the life-span of an itemset where the life-span of an item set is defined as the time period between the first transaction and last transaction containing the item set. Here life-span of an item set may not be same as that of the dataset. The algorithm can extract much more rules than normal A-priori, but it has some limitations.

For example, the method works well if the items are uniformly distributed in the transactions throughout its life-span. However, in practice there may exist some items, which are not uniformly distributed in the transactions throughout their lifetime. For example in the super market data, we may find certain items, which are periodic in nature. For such items if the items life-span is taken into consideration, then the items may not turn out to be frequent because of the large time period when the item is not in the transaction is also considered or even if the items are frequent then they will have very small support value. Considering the seasonal behavior of certain items in the transactions, B. Ozden et al [4] put forward a method of finding cyclic association rules where they proposed algorithms to extract all such rules holding within a user-specified time period. Once the user chooses the time period it will be fixed throughout the execution of the algorithm. In [5, 6], authors tried to address the above shortcomings and proposed a method.

The frequent itemsets extracted by [5, 6] is known as locally frequent itemsets. However, In this paper, our discussions are mostly emphasized on the implementation side of [5, 6] and comparative studies with [3]. Our implementation is Trie-based implementation. This paper is organized as follows. In section-2, we discuss about definitions and notations used in [5, 6]. In section-3, we discuss the algorithm of [5, 6]. In section-4, we discuss about the implementation along with the comparative studies with [3]. Conclusion and Lines for future works are discussed in section-5.

2. TERMS, DEFINITIONS AND NOTATIONS USED

Let T = <to, t1…> be a sequence of time-stamps over which a linear ordering < is defined where ti < tj means ti denotes a time which is earlier than tj. Let I denote a finite set of items and the transaction dataset D is a collection of transactions where each transaction has a part which is a subset of the item set I and the other part is a time-stamp indicating the time in which the transaction had taken place. We assume that D is ordered in the ascending order of the time-stamps. For time intervals we always consider closed intervals of the form [t1, t2] where t1 and t2 are time-stamps. We say that a transaction is in the time interval [t1, t2] if the time-stamp of the transaction say t is such that t1 ≤ t ≤ t2.
We define the local support of an item set in a time interval [t1, t2] as the ratio of the number of transactions in the time interval [t1, t2] containing the item set to the total number of transactions in [t1, t2] for the whole dataset D. We use the notation $Supp_{[t_1,t_2]}$(X) to denote the support of the item set X in the time interval [t1, t2]. Given a threshold σ we say that an item set X is frequent in the time interval [t1, t2] if $Supp_{[t_1,t_2]}$(X) ≥ (σ/100)* tc where tc denotes the total number of transactions in D that are in the time interval [t1, t2].

3. FINDING LOCALLY FREQUENT ITEMSETS WITH ASSOCIATED TIME INTERVALS

Here for the sake of convenience, we discuss the algorithm used in [5, 6] for finding locally frequent itemsets. While constructing locally frequent sets, with each locally frequent set a list of time-intervals is constructed in which the set is frequent. Two thresholds minthd1 and minthd2 are used and these are given as input. During execution, while making a pass through the database, if for a particular item set the time gap between its current time-stamp and the time when it was last seen (before the current time-stamp) is less than the value of minthd1 then the current transaction is included in the current time-interval under consideration; otherwise a new time-interval is started with the current time-stamp as the starting point. The support count of the item set in the previous time interval is checked to see whether it is frequent in that interval or not and if it is then it is added to the list maintained for that set. Also for the locally frequent sets a minimum period length is given by the user as minthd2 and time intervals of length greater than or equal to this value are only kept. If minthd2 is not used than an item appearing once in the whole database will also become locally frequent.
Procedure to compute L1, the set of all locally frequent item sets of size 1.
For each item while going through the database we always keep a time-stamp called lastseen that corresponds to the time when the item was last seen. When an item is found in a transaction and the time-stamp is tm and the time gap between lastseen and tm is greater than the minimum threshold given, then a new time interval is started by setting start of the new time interval as tm and end of the previous time interval as lastseen. The previous time interval is added to the list maintained for that item provided that the duration of the interval and the support of the itemset in that interval are both greater than or equal to the minimum thresholds specified for each. Otherwise lastseen is set to tm, the counters maintained for counting transactions are increased appropriately and the process is continued. Following is the algorithm to compute L1, the list of locally frequent sets of size-1. Suppose the number of items in the dataset under consideration is n and we assume an ordering among the items.

Algorithm 3.1

```
C1 = {(ik,tp[k]) : k = 1,2,…..,n}
 where ik is the k-th item and tp[k] points to a list of time intervals initially empty
for k = 1 to n do
    set lastseen[k], icount[k] ctcount[k] and ptcount[k] to zero
for each transaction t in the database with time stamp tm do
  {for k = 1 to n do
     {if ({ik} ⊆ t) then
        {if(lastseen[k] == 0)
             {lastseen[k] = firstseen[k] = tm;
              icount[k] = ptcount[k]=ctcount[k] =1;
             }
         else if (|tm – lastseen[k]| < minthd1)
                 {lastseen[k] =tm; itemcount[k]++;
                  ctcount[k] ++; ptcount[k]=ctcount[k];
                 }
             else
                 {if ((|lastseen[k] – firstseen[k]| ≥ minthd2)
                           &&(icount[k]/ptcount[k]*100 ≥ σ))
                      add(firstseen[k], lastseen[k]) to tp[k];
                  icount[k] = ptcount[k] = ctcount[k]= 1;
                  lastseen[k] = firstseen[k] = tm;
                 }
        }
      else ctcount[k]++;
     }          // end of k-loop //
  }    // end of do loop //
for k = 1 to n do
  {if ((|lastseen[k] – firstseen[k]| ≥ mintdh2) and
         (icount[k] /ptcount[k] * 100 ≥ σ))
             add (firstseen[k], lastseen[k] ) to tp[k];
   if(tp[k] != 0) add (ik, tp[k]) to L1
  }
```

Three support counts icount, ctcount and ptcount are maintained with each item. When an item is first seen then these are initialized to 1. For each item while making a pass through the dataset when a transaction containing the item is found then icount for that item is increased. To see whether an item is frequent in an interval the total number of transactions in that interval will have to be counted. For this with each item two counts ptcount and ctcount are kept. The value of ctcount increases with each transaction but ptcount changes its value only when a transaction containing an item is found within minthd1 from current value of lastseen and then it takes the value of ctcount. When an item is not seen for more than minthd1 time distance from lastseen then the value of ptcount is used to compute the percentage support count of the item between firstseen and lastseen. If the count percentage of an item in a time interval is greater than the minimum threshold then only the set is considered as a locally frequent set and the locality is the time interval. When a new interval is started for an item then the three counts again start from 1.
After this, A-priori candidate generation algorithm is used to find candidate frequent sets of size-2 and then pruning is applied. With each candidate frequent set of size-2 we associate a list of time intervals. In the candidate generation phase this list is empty. During the pruning phase this list is constructed. The procedure of construction is that when the first subset of an item set appearing in the previous level is found then that list is

taken as the list of time intervals associated with the set. When subsequent subsets are found then the list is reconstructed by taking all possible pair wise intersection of subsets one from each list. If this list becomes empty at any point of time or when a particular subset of the item set under consideration is not found in the pervious level then the set is pruned. Pair-wise intersection of the interval lists are taken for the following reason. If in an interval say [t, t'] an item set say {A,B} is frequent then there exits two time periods [t1, t1'] and [t2, t2'] in which the item sets {A} and {B} are respectively frequent and [t, t'] $\subseteq$ [t1, t1'] $\cap$ [t2, t2']. Using this concept we describe below the modified A-priori algorithm for the problem under consideration.

Algorithm 3.2

```
Modified A priori
Initialize
  k = 1;
C1 = all item sets of size-1
L1 = { frequent item sets of size-1 where with each itemset {ik} a list tp[k] is maintained which gives all time intervals in which the set is frequent}
L1 is computed using algorithm 3.1 */
for(k = 2; Lk-1 ≠ ϕ ; k++) do
      { Ck = apriorigen(Lk-1)
/* same as the candidate generation method of the A-priori algorithm setting tp[i] to zero for all i*/
        prune(Ck);
        drop all lists of time intervals maintained with the sets in Ck
      Compute Lk from Ck.
//Lk can be computed from Ck using the same procedure used for computing L1 //
      k = k + 1
      }
```

Answer = $\bigcup_k L_k$

```
Prune(Ck)
{Let m be the number of sets in Ck and let the sets be s1, s2,…, sm. Initialize the pointers tp[i] pointing to the list of time-intervals maintained with each set si to null
for i = 1 to m do
    {for each (k-1) subset d of si do
         {if d ∉ Lk-1 then
              {Ck = Ck - {si, tp[i]}; break;}
         else
           {if (tp[i] == null) then set tp[i] to point to the list of time intervals maintained for d
                     else {take all possible pair-wise intersection of time intervals one from each list,
                     one list maintained with tp[i] and the other maintained with d and take this as the list
                     for tp[i]
                 delete all time intervals whose size is less than the value of minthd2
               if tp[i] is empty then {Ck = Ck - {si,tp[i]};
                                   break;  }
                }
            }
          }
        }
 }
```

4. IMPLEMENTATION

4.1 Data structure used

The candidate generation and the support counting processes require an efficient data structure in which all candidate item sets are stored since it is important to efficiently find the item sets that are contained in a transaction or in another item set. In our purpose here we used Trie (or Prefix-tree) data structure .

4.2 Datasets

For the experiments we performed here, we used two datasets with different characteristics. We have experimented using one retail dataset [7], and one synthetic dataset available at **http://fimi.cs.helsinki.fi/testdata.html**. The features of datasets are given in table-4.1.

4.3 Analysis of Obtained Results

In this section we discuss about the results along comparative study with results obtained by Ale et al's method and the method. The dataset used for this purpose are the two datasets mentioned above. We kept the parameters like min_sup same for both the methods. We found some interesting results.
For retail datasets comparative study between Ale;s method [3] and method [5, 6] is presented using graph in figure-4.1. From the graph it is clearly observed that method [5, 6] outperforms Ale et al method [3] if other parameters are kept same. That means the method in [5, 6] extract more frequent itemsets than that in [3].

For the dataset T10I4D100K comparative study between Ale;s method [3] and method [5, 6] is presented using graph in figure-4.2. From the graph it is clearly observed that method [5, 6] outperforms Ale et al [3] method if other parameters are kept same. That means method in [5, 6] extract more frequent itemsets than the method in [3].

The results containing the comparative studies with the two datasets are also presented using ba-diagram given in figure-4.3 and figure-4.4.

Table 4.1: Dataset Characteristics

Dataset	#Items	#Transactions	Min |T|	Max |T|	Avg |T|
T10I4D100K	942	100 000	4	77	39
Retail dataset	17000	88162	1	52	9

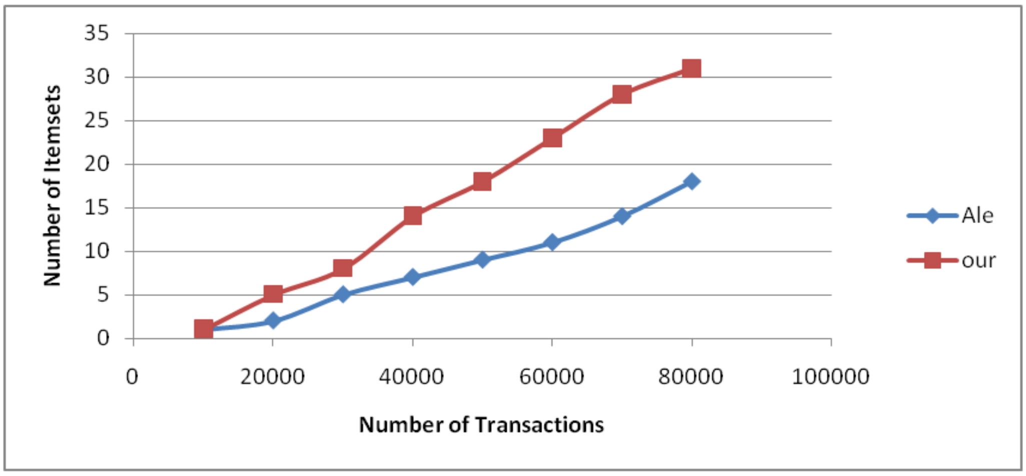

Figure 4.1: Graph of frequent itemsets for retail dataset

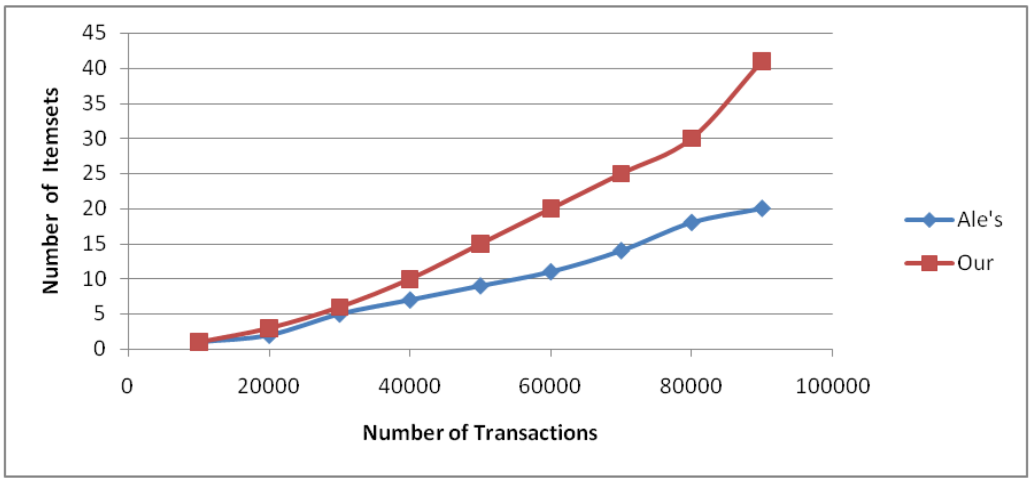

Figure 4.2**:** Graph of frequent itemsets for T10I4D100K dataset

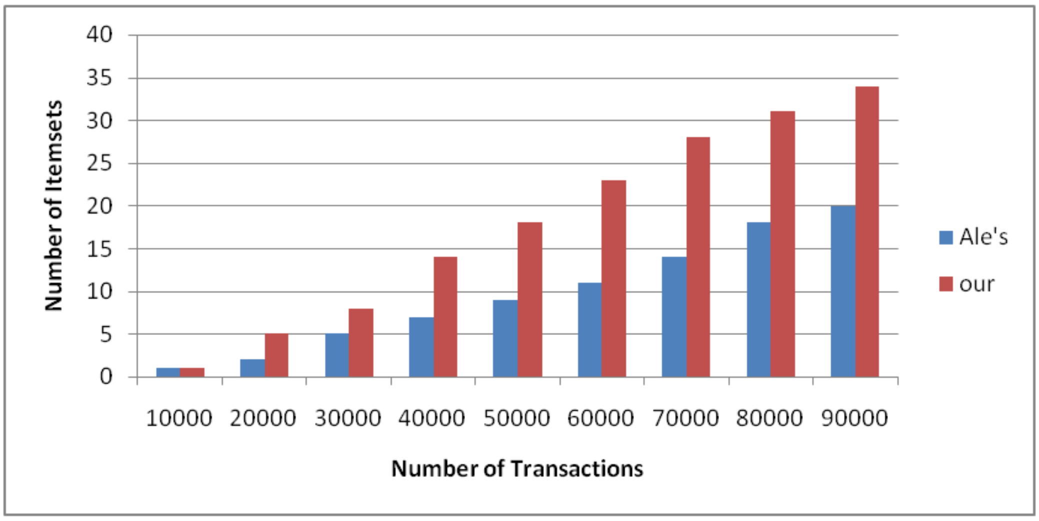

Figure 4.3**:** Retail dataset

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

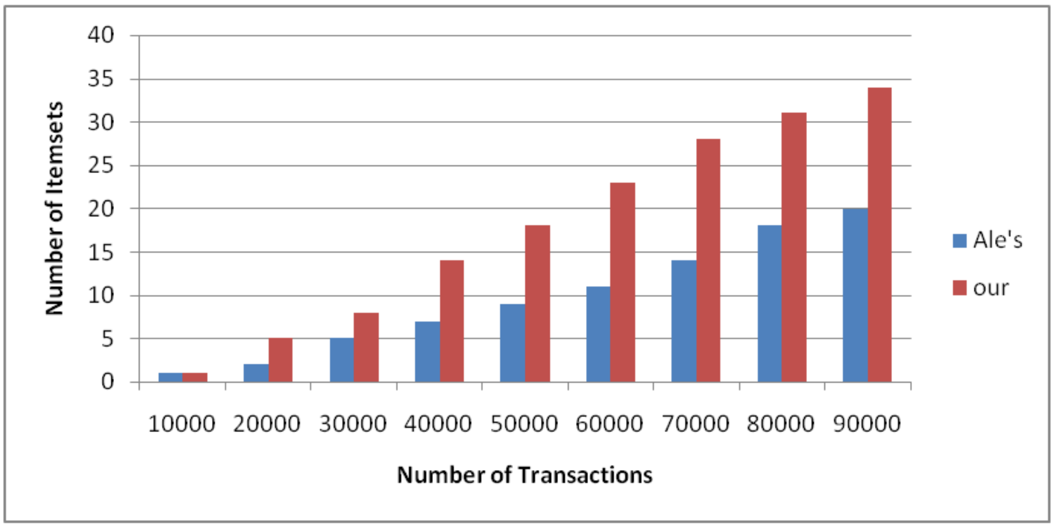

Figure 4.4**:** T10I4D100K dataset

5. CONCLUSION

In this paper, we present some results of the algorithms discussed in [5, 6] and [3]. We have made a comparative study on the both methods and found that algorithm [5, 6] is much efficient than that of Ale et al [3] and it can extract various types of frequent item sets that may present in a dataset and cannot be extracted by [3]. The datasets used here are basically non-temporal. The temporal (calendric) features are incorporated in the datasets so that they can be looked like temporal datasets and can be handle by the methods. We are not claiming that the implementation discussed in this paper is the most efficient one but it serves the purpose of justifying the claim made in [5, 6] and there is an ample scope of improving the work. Here we mainly made comparative studies with Ale et al algorithm [3]. The detailed results are reported using bar diagrammatic and graphical form. In future, we will go better implementation using other types of datastructures viz. hesh-tree.

REFERENCES

1. R. Agrawal, T. Imielinski and A. N. Swami, Mining association rules between sets of items in large databases, In Proc. of 1993 ACM SIGMOD Int’l Conf on Management of Data, Vol. 22(2) of SIGMOD Records, ACM Press, (1993), pp 207-216.
2. R. Agrawal and R. Srikant; Fast Algorithms for Mining Association Rules, In Proc. of the 20th VLDB Conf., Santiago, Chile, 1994.
3. J. M. Ale and G. H. Rossi; An approach to discovering temporal association rules, In Proc. of 2000 ACM symposium on Applied Computing (2000).
4. B. Ozden, S. Ramaswamy and A. Silberschatz; Cyclic Association Rules, In Proc. of the 14th Int’l Conf. on Data Engineering, USA (1998), pp. 412-421.
5. A. K. Mahanta, F. A. Mazarbhuiya and H. K. Baruah; Finding Locally and Periodically Frequent Sets and Periodic Association Rules, In Proc. of 1st Int’l Conf. on Pattern Recognition and Machine Intelligence, LNCS 3776 (2005), pp. 576-582.
6. A. K. Mahanta, F. A. Mazarbhuiya and H. K. Baruah, Finding calendar-based periodic patterns, Pattern Recognition Letters, vol.29, no.9, pp.1274-1284, 2008.
7. Tom Brijs, G. Swinnen, K. Vanhoof and G. Wets; using association rules for product assortment decisions: A case study. In Knowledge Discovery and Data Mining (1999), pp.254-260.