A NOVEL APPROACH TOWARDS MINING LOCALLY FREQUENT ITEMSETS

Md Husamuddin¹, J.Jeya.A.Celin²
College of Computer Science and IT¹; Dept of IT²
AL-Baha University¹, Noor-ul-islam University²
AL-Baha¹, Tamil nadu²,
Kingdom of Saudi Arabia¹, India²

ABSTRACT

Mining patterns that occur in a certain time frame is a well defined data mining problem. Many researchers have proposed different methods for the same. Taking time factor into consideration, different patterns can be extracted. These patterns cannot be extracted normally. These patterns are known as temporal patterns. Here we propose an efficient method of mining locally frequent patterns from temporal datasets. The real time databases are used to prove the efficacy of the method. The comparative studies prove that our method gives more results than the other well known method.

Keywords: Temporal Data, Frequent Itemsets, Lifespan of an Itemset, Local patterns, Apriori candidate generation, A-priori pruning, Trie-datastructures.

1. INTRODUCTION

The problem frequent item set mining is one of the most discussed problems among the researchers. The problem is associated with association rule mining in market basket data and is formulated by Agrawal et al [14]. Researchers came up with many algorithms for mining such datasets. A-priori algorithm is one of the most popular algorithms. It is also called the level-wise algorithm and was proposed by Agrawal and Srikant [13] in 1994. But market basket data are usually temporal in nature e.g. with each transaction, the time of transaction is also recorded. Considering the time frames of such datasets, some interesting patterns can be extracted which normally cannot be extracted. In [10], Ale et al proposed a method of extracting association rules which hold throughout the life-span of an item set 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 be different from the dataset. Their algorithm
is a modification of well-known \textit{A-priori} algorithm and it can extract much more rules than normal \textit{A-priori}. But it has some limitations. For example, the items should be uniformly distributed in the transactions throughout its life-span. Practically it is not possible as there may exist some items, which are not uniformly distributed in the transactions throughout their lifetime. For example in the \textit{super market data}, we may find certain items, which are periodic in nature. An example of such items is \textit{Ice cream}, which is present in the transactions that happen in the summer season when the temperature is high and seasons are periodic or repetitive in general i.e. there is a large time gap between two consecutive seasons of same type when the items may not present in the transaction. 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. Although the method proposed by \textit{Ale et al} is a noble method and work good in many cases, it cannot extract various types of pattern that may exist in the dataset as it does not consider the time gap between two successive transactions containing an item. Considering the seasonal behavior of certain items in the transactions, \textit{B. Ozden et al} [3] proposed a method of finding cyclic association rules where they proposed algorithms to extract all such rules holding within a specified time period. In their method the user has to specify the time period. The time period will be fixed throughout the execution of the algorithm.

In [1], authors tried to address the above shortcomings and proposed a method. The frequent itemsets extracted by [1] is known as locally frequent itemsets. However, In this paper, we discuss the implementation of [1] and make comparative studies with [10]. Our implementation is a trie-based implementation. Although, we are not claiming that the implementation is the most efficient one however it establishes our claim made in [1].

The paper is organized as follows. In section-2 we discuss about the related works. In section-3 we discuss about definitions and notations used in the algorithm [1]. In section-4, we discuss the algorithms for mining locally frequent itemsets. In section-5, we discuss about details of the implementations along with the details findings in tabular and comparative studies with \textit{Ale et al} work. In section-6 we give brief conclusion.

\section{RELATED WORKS}

The problem of discovery of association rules was first formulated by Agrawal \textit{et al} [14] in 1993. Given a set \(I\), of items and a large collection \(D\) of transactions involving the items, the problem is to find relationships among the items i.e. the presence of various items in the transactions. A transaction \(t\) is said to support an item if that item is present in \(t\). A transaction \(t\) is said to support an item set if \(t\) supports each of the items present in the item set. An association rule is an expression of the form \(X \Rightarrow Y\) where \(X\) and \(Y\) are subsets of the item set \(I\). The rule holds with confidence \(\tau\) if \(\tau\%\) of the transaction in \(D\) that supports \(X\) also supports \(Y\). The rule has support \(\sigma\) if \(\sigma\%\) of the transactions supports \(X \cup Y\). A method for the discovery of association rules was given in [13], which is known as the \textit{A-priori} algorithm. This was then followed by subsequent refinements, generalizations, extensions and improvements. The association rule discovery process is also extended to incorporate temporal aspects. In temporal association rules each rule has associated with it a time interval in which the rule holds. The problems associated are to find valid time periods during which association rules hold, the discovery of possible periodicities that association rules have and the discovery of association rules with temporal features. In
[1], [10], the problem of temporal data mining is addressed and techniques and algorithms have been developed for this. In [10] an algorithm for the discovery of temporal association rules is described where for each item (which is extended to item set) a lifetime or life-span is defined which is the time gap between the first occurrence and the last occurrence of the item in the transaction in the database. Supports of items are calculated only during its life-span. Thus each rule has associated with it a time frame corresponding to the lifetime of the items participating in the rule. In [1], the works of [10] is extended by incorporating time gap between two consecutive transactions containing an item and it is shown that the number of frequent item sets (called locally frequent item sets) depends upon two thresholds $minh1$ and $minh2$ besides minimum support sigma.

In [[2], [5], [6]] author uses the work of [1] to extract calendar-based periodic patterns, clustering of patterns and sequential patterns respectively. In [11], author made a survey on local patterns for mining medical data.

In [8], an implementation of A-priori algorithm is given which theoretically and experimentally analyze A-priori frequent item set mining algorithm and outperforms others. The authors have shown that data structure is the main factor in the efficiency of their implementation. In [9], a trie-based implementation is given which uses a trie to store the counters for the different item sets. This tree is grown top to down level by level, pruning those branches that cannot contain a frequent item set. This also makes counting efficient. Although couple implementations have come out till today [[4], [12]] along with complexity analysis [7], very few have addressed the temporal features that may present in a transaction dataset. In this paper we give a trie-based implementation of algorithm discussed in [1]. We made a comparative study with Ale et al’s algorithm [10]. We show that the algorithm discussed in [1] outperforms Ale et al’s algorithm.

3. TERMS, DEFINITIONS AND NOTATIONS USED

Let $T = \langle t_0, t_1, \ldots \rangle$ be a sequence of time-stamps over which a linear ordering $<$ is defined where $t_i < t_j$ means $t_i$ denotes a time which is earlier than $t_j$. 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 $[t_1, t_2]$ where $t_1$ and $t_2$ are time-stamps. We say that a transaction is in the time interval $[t_1, t_2]$ if the time-stamp of the transaction say $t$ is such that $t_1 \leq t \leq t_2$.

We define the local support of an item set in a time interval $[t_1, t_2]$ as the ratio of the number of transactions in the time interval $[t_1, t_2]$ containing the item set to the total number of transactions in $[t_1, t_2]$ 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 $[t_1, t_2]$. Given a threshold $\sigma$ we say that an item set $X$ is frequent in the time interval $[t_1, t_2]$ if $Supp_{[t_1, t_2]}(X) \geq (\sigma/100) \times tc$ where $tc$ denotes the total number of transactions in $D$ that are in the time interval $[t_1, t_2]$.

4. FINDING LOCALLY FREQUENT ITEMSETS WITH ASSOCIATED TIME INTERVALS

Here for the sake of convenience, we discuss the algorithm used in [1] 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 $minh1$ and $minh2$ are used and these are
A NOVEL APPROACH TOWARDS MINING LOCALLY FREQUENT ITEMSETS

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 4.1
C1 = \{(i_k, tp[k]) : k = 1, 2, ..., n\}
where i_k 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], ccount[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 \{i_k\} \subseteq t then
      if (lastseen[k] == 0)
        lastseen[k] = firstseen[k] = tm;
        icount[k] = ptcount[k] = ccount[k] = 1;
      else if (|tm - lastseen[k]| < minthd1)
        lastseen[k] = tm; itemcount[k]++; icount[k] = ptcount[k] = ccount[k] = 1;
      else if ((|tm - lastseen[k]| \geq minthd2) && (icount[k]/ptcount[k]*100 \geq \sigma))
        add(firstseen[k], lastseen[k]) to tp[k];
        icount[k] = ptcount[k] = ccount[k] = 1;
        lastseen[k] = firstseen[k] = tm;
    else ccount[k]++;
  } // end of k-loop //
} // end of do loop //
for $k = 1$ to $n$ do
  \{ if ($|lastseen[k] - firstseen[k]| \geq \text{mintdh}2$) and
    $(icount[k] / ptcount[k] \times 100 \geq \sigma)$
    add ($firstseen[k]$, $lastseen[k]$) to $tp[k]$;
  \}

  if ($tp[k] \neq 0$) add ($i_k$, $tp[k]$) to $L_1$

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 $\text{mintdh}1$ from current value of $lastseen$ and then it takes the value of $ctcount$. When an item is not seen for more than $\text{mintdh}1$ 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 $[t_1, t_1']$ and $[t_2, t_2']$ in which the item sets $\{A\}$ and $\{B\}$ are respectively frequent and $[t, t'] \subseteq [t_1, t_1'] \cap [t_2, t_2']$. Using this concept we describe below the modified A-priori algorithm for the problem under consideration.

**ALGORITHM 4.2**
Modified A priori

Initialize

$k = 1$;

$C_1 = \{\text{all item sets of size-1}\}$

$L_1 = \{\text{frequent item sets of size-1 where with each itemset } \{i_k\} \text{ a list } tp[k] \text{ is maintained which gives all time intervals in which the set is frequent}\}$

$L_1$ is computed using algorithm 4.1 */

for ($k = 2$; $L_{k-1} \neq \phi$; $k++$) do
  \{ $C_k = \text{apriorigen}(L_{k-1})$ */
  
  same as the candidate generation method of the A-priori algorithm setting $tp[i]$ to zero for all $i$*/
  
  prune($C_k$);
  
  drop all lists of time intervals maintained with the sets in $C_k$
  
  Compute $L_k$ from $C_k$.

Md Husamuddin and J. Jeya .A. Celin
//\textit{L}_k \text{ can be computed from } \textit{C}_k \text{ using the same procedure used for computing } \textit{L}_1; //
\text{ } 
k = k + 1 
\text{ } 
\) \text{Answer} = \bigcup_k \text{ } \textit{L}_k
\) 

\text{Prune}(\textit{C}_k)
\{Let \text{ } m \text{ } be \text{ } the \text{ } number \text{ } of \text{ } sets \text{ } in \text{ } \textit{C}_k \text{ } and \text{ } let \text{ } the \text{ } sets \text{ } be \text{ } s_1, s_2, \ldots, s_m. \text{ Initialize \text{ } the \text{ } pointers \text{ } } \text{tp}[i] \text{ } pointing \text{ } to \text{ } the \text{ } list \text{ } of \text{ } time-intervals \text{ } maintained \text{ } with \text{ } each \text{ } set \text{ } s_i \text{ } to \text{ } null \text{ for } i = 1 \text{ } to \text{ } m \text{ } do \text{ } 
\text{ } 
\text{ } \{\text{for each } (k-1) \text{ subset } d \text{ of } s_i \text{ do} \text{ } 
\text{ } \{\text{if } d \not\subseteq \textit{L}_{k-1} \text{ then} \text{ } 
\text{ } \{\text{ } \textit{C}_k = \textit{C}_k \setminus \{s_i, \text{tp}[i]\}; \text{break;} \}
\text{ } \text{else} \text{ } 
\text{ } \{\text{if } (\text{tp}[i] == \text{null}) \text{ then set } \text{tp}[i] \text{ to point to the list of time-intervals maintained for } d \text{ } \text{else } \{\text{take all possible pair-wise intersection of time intervals one from each list, one list maintained with } \text{tp}[i] \text{ and the other maintained with } d \text{ and take this as the list for } \text{tp}[i]\} \text{ } 
\text{ } \text{delete all time intervals whose size is less than the value of } \text{minthd2} \text{ } \text{if } \text{tp}[i] \text{ is empty then } \{\textit{C}_k = \textit{C}_k \setminus \{s_i, \text{tp}[i]\}; \text{break; } \}
\text{ } \text{\} \}
\text{ } \text{\} \}
\text{ } \text{\} \}
\text{ } \text{\} \}
\text{ } \text{\} \}
\text{ } \text{\} \}
\text{ } \text{\} \}
\text{ } \text{\} \}
\text{5. IMPLEMENTATION}
\text{5.1 DATASTRUCTURES 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 general two data structures namely Hash-tree and Trie (or Prefix-tree) are used for this purpose. In our work, we have used the Trie-data structure.

\text{5.1.1 TRIE-DATASTRUCTURES}

All the items under consideration are marked as 1, 2, \ldots, n where n is the total number of items. We also assume that the items in the transactions are ordered in the same ordering. In a trie, every k-item set has a node associated with it, as does its (k-1)-prefix. The empty item set is the root node. All the 1-item sets are attached to the root node as its children. Every other k-item set is attached to its (k-1)-prefix. Every node represents an item set. The node stores the last item in the item set it represent, pointer to childlist, pointer to parent, pointer to list of time intervals where the item set is frequent and pointer to its right sibling. The siblings of each node are implemented as linked list. So, every level consists of a collection of lists. In our implementation all the list in a level are again maintained as a list of lists. The level-1 is a single list consisting of all the 1-item sets.

At a certain iteration k, all the candidate k-item sets are stored at depth k in the trie. In order to count the supports of item sets for a particular level, the item sets represented by the nodes are found by moving upwards to the root using parent pointers.

Also the join step of the candidate generation procedure becomes very simple when a trie is being used. Since all item sets of size-k with the same (k-1)-prefix are represented as a linked list (all are children of the same node), to generate all candidate k-size item sets with (k-1)-prefix X we simply copy
all right siblings of the node representing \( X \) and add them as child of \( X \). Candidate generation procedure also computes pair-wise intersection of the list of time intervals associated with the two item sets that are joined to get the candidate. If intersection of the lists of time intervals is found to be empty or the length of all intervals in the list is found to be less than \( \text{minthd}2 \) then the newly added node is removed. It also includes \( A\text{-priori} \) pruning step, which is required to check whether all the subsets of the candidates are present in the previous level.

To illustrate the structure we consider the items A, B, C, D.

![Trie Structure](image)

---

**5.2 DESCRIPTIONS OF DATASETS**

To compute the collection of item sets, we need to access the dataset. Since the datasets tend to be very large, it is not always possible to store them in the main memory.

Most algorithms is the representation of the transaction dataset. Such a dataset can be represented by a binary two-dimensional matrix in which every row represents an individual transaction and columns represent the items in \( I \). Such a matrix can be implemented in several ways viz. horizontal data layout and vertical data layout. But here each transaction is attached with time-stamp e.g. calendar-date. So conceptually the dataset consists of numeric as well binary attributes where the time-stamp consists of three columns viz. day, month and year having numeric values within a specified ranges and rest of the attributes are as usual binary. A sample of such dataset is given below:

<table>
<thead>
<tr>
<th>Time-stamps</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Day</strong></td>
<td><strong>Month</strong></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>

Md Husamuddin and J. Jeya .A. Celin
5.3 EXAMPLES OF DATASETS

For the experiments we performed here, we used two datasets with different characteristics. We have experimented using one real dataset, and one synthetic dataset generated by the program provided by the Quest research group at IBM Almaden. This generator can be downloaded from http://www.almaden.ibm.com. The synthetic dataset T4015D200K (5.0 MB) is available from FIMI’03 website at http://fimi.cs.helsinki.fi/testdata.html. The retail dataset was donated by Tom Brijs [15] and contains the retail market basket data from an anonymous Belgian retail store. These datasets are non-temporal so they cannot be directly used by our method. We have incorporated the temporal feature in the datasets so that they can be temporal datasets and be handled by our method. A program was developed for this. The program takes as input a starting date and two values for the minimum and maximum number of transactions per day. A number between these two limits are selected at random and that many consecutive transactions are marked with the same date. So that many transactions have taken place on that day. This process starts from the first transaction to the end by marking the transactions by consecutive dates (assuming that the market remains open on all week days). The datasets used, the number of items, the number of transactions in each dataset and the minimum, maximum and average length of transactions are reported in the table below:

| Dataset       | #Items | #Transactions | Min $|T|$ | Max $|T|$ | Avg $|T|$ |
|---------------|--------|---------------|------|------|------|------|
| T4015D200k    | 852    | 100 000       | 5    | 79   | 41   |
| Retail dataset| 15000  | 88162         | 2    | 54   | 11   |

5.4 RESULTS OBTAINED

In this section we make comparative study with our results and with results obtained by Ale et al’s method. The dataset used for this purpose are the two datasets mentioned above.

With Retail dataset and thresholds $minthd1 = 10$ days, $minthd2 = 30$ days and $min\_sup\_sigma = 4\%$, we obtained the following results:

1) By Ale et al’s method

<table>
<thead>
<tr>
<th>Levels</th>
<th>Frequent item sets</th>
<th>Time intervals</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level-1</td>
<td>42</td>
<td>[3-1-2010, 23-4-2013]</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>[3-1-2010, 23-4-2013]</td>
<td>15.5%</td>
</tr>
<tr>
<td></td>
<td>49</td>
<td>[3-1-2010, 23-4-2013]</td>
<td>55.3%</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>[3-1-2010, 23-4-2013]</td>
<td>14.7%</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>[3-1-2010, 23-4-2013]</td>
<td>45.6%</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>[3-1-2010, 23-4-2013]</td>
<td>3%</td>
</tr>
</tbody>
</table>

<p>| Level-2| 42, 49             | [3-1-2010, 23-4-2013]| 7.4%   |
|        | 42, 58             | [3-1-2010, 23-4-2013]| 7%     |
|        | 48, 49             | [3-1-2010, 23-4-2013]| 10.5%  |
|        | 48, 58             | [3-1-2010, 23-4-2013]| 7%     |
|        | 49, 51             | [3-1-2010, 23-4-2013]| 10.7%  |</p>
<table>
<thead>
<tr>
<th>Levels</th>
<th>Frequent item sets</th>
<th>Time intervals</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level-1</td>
<td>42 [3-1-2010, 23-4-2013]</td>
<td>15%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>48 [3-1-2010, 23-4-2013]</td>
<td>15.5%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>49 [3-1-2010, 23-4-2013]</td>
<td>55.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>51 [3-1-2010, 30-6-2011]</td>
<td>25.4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1-8-2012, 23-4-2013]</td>
<td>26.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>58 [3-1-2010, 23-4-2013]</td>
<td>45.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>75 [3-1-2010, 23-4-2013]</td>
<td>3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13725 [11-3-2012, 25-5-2012]</td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>Level-2</td>
<td>42, 49 [3-1-2010, 23-4-2013]</td>
<td>7.4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>42, 51 [3-1-2010, 26-6-2011]</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1-8-2012, 23-4-2013]</td>
<td>5.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>42, 58 [3-1-2010, 23-4-2013]</td>
<td>7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>48, 49 [3-1-2010, 23-4-2013]</td>
<td>10.5%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>48, 51 [3-1-2010, 30-6-2011]</td>
<td>7.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1-8-2012, 23-4-2013]</td>
<td>7.1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>48, 58 [3-1-2010, 23-4-2013]</td>
<td>7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>49, 51 [3-1-2010, 30-6-2011]</td>
<td>20%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1-8-2012, 23-4-2013]</td>
<td>20.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>49, 58 [3-1-2010, 23-4-2003]</td>
<td>31%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>49, 13725 [14-3-2012, 25-5-2012]</td>
<td>3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>51, 58 [3-1-2010, 30-6-2011]</td>
<td>14.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1-8-2012, 23-4-2013]</td>
<td>15.5%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>58, 13725 [11-3-2012, 25-5-2012]</td>
<td>5.5%</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.3: Results obtained by Ale et al’s method

2) By our method
If we closely observe the tables 1.3 and 1.4, we notice the following interesting differences:

i) Besides all the frequent item sets extracted by Al et al’s method, our method extracts some extra frequent item sets e.g. \{13, 72, 5\}, \{42, 51\}, \{49, 13, 72, 5\}, \{58, 13, 72, 5\}, \{48, 49, 51\}. Thus our method gives much more frequent sets than Al et al’s method.

ii) Again the item set \{51\} is associated with the time interval [3-1-2010, 23-4-2013] with support 16.9% (Al et al’s method), however by our method we get the same item set associated with two different time intervals [3-1-2010, 30-6-2011] and [1-8-2012, 23-4-2013] with support 25.4% and 26.3% respectively that means there is a large time gap from 31-7-2011 to 2-10-2012 when no transaction contains the item set \{51\} again supports with respect to the individual time interval are much high. Similar observations are recorded for the item sets \{42, 51\}, \{48, 51\}, \{49, 51\}, \{51, 58\}, \{48, 49, 51\} and \{49, 51, 58\}.

Again with the synthetic dataset T4015D200K and \textit{minthd1} = 15 days, \textit{minthd2} = 45 days and \textit{min_sup sigma} = 5% we obtained the following results

3) By Al et al’s method

4) By our method
If we closely observe the tables 1.5 and 1.6, we notice the following interesting differences:

The item set \{7\} is not extracted by Ale et al.'s method and so it is infrequent if its whole life-span is considered together. However by our method it has come as 8 times frequent within its life-span, so it is locally frequent 8 times within its life-span and rest of the frequent item sets extracted by both the methods are same.

6. CONCLUSION

In this paper, we present some results of the algorithms discussed in [1] and [10]. We have made a comparative study on the both methods and found that algorithm [1] is much efficient than that of Ale et al [10] and it can extract various types of frequent item sets that may present in a dataset and cannot be extracted by [10]. The implementation is a trie-based implementation. The datasets used here are basically non-temporal. The temporal (calendric) features are incorporated in the datasets to make it look like temporal datasets and can be used by the methods. The results are displayed in tabular form.

In this paper we are not claiming that the implementation discussed is the most efficient one but it serves the purpose of justifying the claim made in [1] and there is an ample scope of improving the work. In future better implementation techniques can be applied. In future comparative studies can be made with other known methods.

REFERENCES

Md Husamuddin and J. Jeya .A. Celin


