CHAPTER- 5

INFREQUENT PATTERNS

5.1 Introduction

Patterns that are rarely found in database are often considered to be uninteresting and are eliminated using the support measure. Such patterns are known as infrequent patterns [1]. An infrequent pattern is an itemset or a rule whose support is less than the \( \minsup \) threshold.

Although a vast majority of infrequent patterns are uninteresting, some of them might be useful to the analysis, particularly those that correspond to negative correlations in data. For example, the sale of DVDs and VCRs together is low because any customer who buys a DVD will most likely not buy a VCR and vice versa. Such negative-correlated patterns are useful to help identify competing items, which are items that can be substituted for one another. Examples of competing items include tea versus coffee, butter versus margarine, regular versus diet soda, and desktop versus laptop computers.

Some infrequent patterns may also suggest the occurrence of interesting rare events or exceptional situations in the data. For example, if \{Fire = Yes\} is frequent but \{Fire = Yes, Alarm = On\} is infrequent, then the latter is an interesting infrequent pattern because it may indicate faulty alarm systems. To detect such unusual situations, the expected support of a pattern must be determined, so that, if a pattern turns out to have a considerably lower support than expected, it is declared as an interesting infrequent pattern.

Mining infrequent patterns is a challenging endeavor because there is an enormous number of such patterns that can be derived from a given data set. More specifically, the key issues in mining infrequent patterns are: (1) how to identify interesting infrequent patterns, and (2) how to efficiently discover them in large data sets. To get a different perspective on various types of interesting infrequent patterns, two related concepts are negative patterns and negatively correlated patterns.

**Negative Patterns:** Let \( I = \{i_1, i_2, ..., i_d\} \) be a set of items. A negative item, \( i_k \), denotes the absence of item \( i_k \) from a given transaction. For examples, coffee is a negative item whose value is 1 if a transaction does not contain coffee.

**Negative Itemset:** A negative itemset \( X \) is an itemset that has the following properties:
(1) \( X = A \cup B \), where \( A \) is a set of positive items, \( B \) is a set of negative items, \( |B| \geq 1 \), and (2) \( s(X) \geq \minsup \).

**Negative Association Rule:** A negative association rule is an association rule that has the following properties: (1) the rule is extracted from a negative itemset, (2) the support of the rule is greater than or equal to \( \minsup \), and (3) the confidence of the rule is greater than or equal to \( \minconf \).

The negative itemsets and negative association rules are collectively known as negative patterns. An example of negative association rule is tea \( \rightarrow \) coffee', which may suggest that people who drink tea tend to not drink coffee.
**Negatively Correlated Patterns:** An itemset is negatively correlated if its support is below the expected support computed using the statistical independence assumption. The smaller $s(X)$, the more negatively correlated is the pattern.

**Negatively Correlated Association Rule:** An association rule $X \rightarrow Y$ is negatively correlated if

$$s(X \cup Y) < \prod_i s(x_i) \prod_j s(y_j),$$

Where $X$ and $Y$ are disjoint itemsets; i.e., $X \cup Y = \emptyset$ and $x_i \subset X$ and $y_j \subset Y$.

The condition for negative correlation can also be expressed in terms of support for positive and negative itemsets. Let $X'$ and $Y'$ denote corresponding negative itemsets for $X$ and $Y$, respectively. Since

$$s(X \cup Y) - s(X)s(Y) = s(X \cup Y) - [s(X \cup Y) + s(X \cup Y')]\frac{s(X \cup Y)}{s(X \cup Y) + s(X' \cup Y')} - s(X \cup Y')s(X' \cup Y')$$

The condition for negative correlation can be stated as follows:

$$s(X \cup Y) s(X' \cup Y') < s(X \cup Y') s(X' \cup Y).$$

The negatively correlated itemsets and association rules are known as negatively correlated patterns.

**Comparison among Infrequent Patterns, Negative Patterns, and Negatively Correlated Patterns:**
Infrequent patterns, negative patterns, and negatively correlated patterns are three closely related concepts. Although infrequent patterns and negatively correlated patterns refer only to itemsets or rules that contain positive items, while negative patterns refer to itemsets or rules that contain both positive and negative items.

---

**Fig 5.1. Comparisons among infrequent, negative and negatively correlated patterns**
5.2 Techniques for Mining Interesting Infrequent Patterns

In principle, infrequent itemsets are given by all itemsets that are not extracted by standard frequent itemset generations algorithms such as Apriori and FP-growth. Since the number of infrequent patterns exponentially large, especially for sparse, high-dimensional data, techniques developed for mining infrequent patterns focus on finding only interesting infrequent patterns. In the figure 5.2 all the colored nodes represent infrequent itemsets.

Fig 5.2 Frequent and infrequent itemsets

Techniques Based on Mining Negative Patterns: Transaction data can be binarized by augmenting it with negative items. Figure 5.3 shows an example of transforming the original data into transactions having both positive and negative items. By applying existing frequent itemset generation algorithm such as Apriori on the augmented transactions, all the negative itemsets can be derived. Such an approach is feasible only if a few variables are treated as symmetric binary.

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{A,B}</td>
</tr>
<tr>
<td>2</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>3</td>
<td>{C}</td>
</tr>
<tr>
<td>4</td>
<td>{B,C}</td>
</tr>
<tr>
<td>5</td>
<td>{B,D}</td>
</tr>
</tbody>
</table>
Another class of techniques considers an infrequent pattern to be interesting only if its actual support is considerably smaller than its expected support. For negatively correlated patterns, the expected support is computed based on the statistical independence assumption. Two alternative approaches for determining the expected support of a pattern using (1) a concept hierarchy and (2) a neighborhood-based approach known as indirect association.

**Support Expectation Based on Concept Hierarchy:** Objective measures alone may not be sufficient to eliminate uninteresting infrequent patterns. For example, support bread and laptop computer are frequent items. Even though the itemset \{bread, laptop computer\} is infrequent and perhaps negatively correlated, it is not interesting because their lack of support seems obvious to domain experts. Therefore, a subjective approach for determining expected support is needed to avoid generating such infrequent patterns.

In the preceding example, bread and laptop computers belong to two completely different product categories, which is why it is not surprising to find that their support is low. This example also illustrates the advantage of using domain knowledge to prune uninteresting patterns. For market based data, the domain knowledge can be inferred.
Support Expectation Based on Indirect Association: Consider a pair of items, \((a, b)\), that are rarely bought together by customers. If \(a\) and \(b\) are unrelated items such as bread and DVD player, then their support is expected to be low. On the other hand, if \(a\) and \(b\) are related items, then their support is expected to be high. The expected support was previously computed using a concept hierarchy. Here, an approach for determining the expected support between a pair of items by looking at other items commonly purchased together with these two items.

For example, suppose customers who buy a sleeping bag also tend to buy other camping equipment, whereas those who buy a desktop computer also tend to buy other computer accessories such as optical mouse or printer. Assuming there is no other item frequently bought together with a sleeping bag and desktop computer, the support for these unrelated items is expected to be low. On the other hand, suppose diet and regular sodas are often bought together with chips and cookies. Even without using a concept hierarchy, both items are expected to be somewhat related and their support should be high. Such patterns are known as indirect association patterns.

Indirect association has many potential applications. In the market basket domain, \(a\) and \(b\) may refer to computing items such as desktop and laptop computers. In text mining, indirect association can be used to identify synonyms, antonyms, or words that are used in different contexts. For example, given a collection of documents, the word data may be indirectly associated with gold via the mediator mining. This pattern suggests that the word mining can be used in two different contexts – data mining versus gold mining.

Indirect association can be generated in the following way. First, the set of frequent itemsets is generated using standard algorithm such as Apriori or FP-growth. Each pair of frequent \(k\)-itemsets are then merged to obtain a candidate indirect association \((a, b, Y)\), where \(a\) and \(b\) are a pair of items and \(Y\) is their common mediator. For example, if \(\{p, q, r\}\) and \(\{p, q, s\}\) are frequent-3 itemsets, then the candidate indirect association \((r, s, \{p, q\})\) is obtained by merging the pair of frequent itemsets. Once the candidates have been generated, it is necessary to verify that they satisfy the itempair support and mediator dependence conditions.

Algorithm 5.1 : Algorithm for mining indirect associations:
1: Generate \(F_k\), the set of frequent itemsets.
2: for \(k = 2\) to \(k_{\text{max}}\) do
3: \(C_k = \{(a, b, Y) \mid \{a\} \cup Y \subseteq F_k, \{b\} \cup Y \subseteq F_k, a \neq b\}\)
4: for each candidate \((a,b,Y) \in C_k\) do
5: if \(s(a, b) < t_s \land d(a, Y) \geq t_d\) then
5. 3 Outliers

Outliers have been informally defined as observations in a data stream which appear to be inconsistent with the remainder of the data stream, deviate so much from other observations so as to arouse suspicions that they were generated by a different mechanism. Searching for outliers in data stream is an important area of research in the world of data mining with numerous applications, including credit card fraud detection, discovery of criminal activities in electronic commerce, weather prediction, marketing and customer segmentation [3]. Consequently the problem of efficient outlier detection has recently drawn increased attention.

**K-NN based outlier detection:** In [2] ODABK (outlier detection algorithm based KNN), is proposed to improve the efficiency for detecting outliers in data stream. K-NN method is a kind of classification methods based on statistics which is instance-based since it stores all of the training samples [4]. The k-NN method can also be used for detecting outliers. However, it can incur expensive computational costs when the number of potential neighbors (i.e., stored training samples) with which to compare a given unlabeled sample is great. Therefore, efficient indexing techniques are required. Unlike decision tree induction and back-propagation, K-NN assigns equal weight to each attribute. This may cause confusion when there are many irrelevant attributes in the data. Thus need more flexibility in the outlier definition over data stream. So [2] proposed an algorithm which is based on K-NN method in accordance with the properties of data stream. They named their algorithm as ODABK (outlier detection algorithm based KNN). ODABK algorithm, can efficiently mine outliers over data stream and scales well with increase of data size. Hash Structure, mutual neighborhood and optimized logical operations are used to speed up the process.

**Frequent Pattern Based Outlier Detection** :The problem of finding all frequent itemsets in \( D \) is then traditionally defined as follows. Given user defined threshold for the permissible minimal support, find all itemsets with support greater or equal to \( \text{minisupport} \). Frequent itemsets are also called frequent patterns. From the viewpoint of knowledge discovery, frequent patterns reflect the “common patterns” that apply to many objects, or to large percentage of objects, or to large percentage of objects, in the dataset. In contrast, outlier detection focuses on a very small percentage of data objects. Hence, the idea of making use of frequent patterns for outlier detection is very intuitive.

If a data object contains more frequent patterns, its \( \text{FPOF} \) (Frequent Pattern Outlier Factor) [5] value will be larger, which indicates that it is unlikely to be an outlier. In contrast, objects with smaller \( \text{FPOF} \) values will have greater outlying-nesses.
the $FPOF$ value is between 0 and 1. To *describe* the reasons why identified outliers are abnormal, the itemsets those are not contained in the transaction (it is said that the itemset is *contradict* to the transaction) are good candidates.

The algorithm $DSFindFPOF$ was proposed by [5] for detecting outliers from data stream. For each transaction in the stream, the value of $FPOF$ is computed. And, updating the *top-*$n$ FP-outliers with their corresponding *top-*$k$ contradict frequent patterns.

**Algorithm 5.2 $DSFindFPOF$**

**Input:** $DS$  //the transactional data stream  
    $minisupport$  //threshold for the permissible minimal support  
    $top-n$  // threshold value for *top-*$n$ FP-outliers  
    $top-k$  //threshold value for *top-*$k$ contradict frequent patterns  
    $\varepsilon$  //error parameter for frequent pattern  

**Output:** The *top-*$n$ FP-outliers with their corresponding TKCFPs so far

1: begin
2:   foreach transaction $t$ in $DS$ do begin
3:      update entries of the form $(e,f,\text{maximal error})$
4:      compute the value of outlier factor of $t$ using current frequent patterns
5:      if $t$ is in the *top-*$n$ FP-outliers then
6:         finds its *top-*$k$ contradict frequent patterns
7:         output $t$ and its *top-*$k$ contradict frequent patterns
8:   end
9: end
10: end

**Incremental LOF (Local Outlier Factor):** Outlier detection algorithms determine outliers once all the data records (samples) are present in the dataset. They are referred [7] as *static outlier detection algorithms*. In contrast, *incremental* outlier detection techniques identify outliers as soon as new data record appears in the dataset. An incremental outlier detection algorithm based on computing the densities of local neighborhoods is proposed by [7].

The main idea of the LOF algorithm [6] is to assign to each data record a degree of being outlier. This degree is called the *local outlier factor (LOF)* of a data record. Data records (points) with high LOF have local densities smaller than their neighborhood and typically represent stronger outliers, unlike data points belonging to uniform clusters that usually tend have lower LOF values. The algorithm for computing the LOFs for all data records has the following steps:

1. For each data record $q$ compute *k-distance*($q$) as distance to the *k*th nearest neighbor
2. Compute reachability distance for each data record \( q \) with respect to data record \( p \) as follows: 
\[
\text{reach-dist}_k(q,p) = \max(d(q,p), k\text{-distance}(p)) \quad \text{where } d(q,p) \text{ is Euclidean distance from } q \text{ to } p.
\]

3. Compute **local reachability density (lrd)** of data record \( q \) as inverse of the average reachability distance based on the \( k \) nearest neighbors of the data record \( q \).
\[
lrd(q) = \frac{1}{\sum_{p \in \text{NN}(q)} \text{reach-dist}_k(q, p) / k}.
\]

4. Compute **LOF** of data record \( q \) as ratio of average local reachability density of \( q \)’s \( k \) nearest neighbors and local reachability density of the data record \( q \).
\[
\text{LOF}(q) = \frac{1}{k} \frac{\sum_{p \in \text{NN}(q)} lrd(p)}{lrd(q)}.
\]

The main advantages of LOF approach over other outlier detection techniques include:
- It detects outliers with respect to density of their neighboring data records; not to the global model.
- It is able to detect outliers regardless the data distribution of normal behavior, since it does not make any assumptions about the distributions of data records.

Applying static LOF outlier detection algorithms to data streams would be extremely computationally inefficient and/or very often may lead to incorrect prediction results. The incremental LOF algorithm [7] computes LOF value for each data record inserted into the data set and instantly determines whether inserted data record is outlier. In addition, LOF values for existing data records are updated if needed.

**i. Insertion.** In the insertion part, the algorithm performs two steps:
- a) Insertion of new record, when it computes reach-dist, lrd and LOF values of a new point;
- b) Maintenance, when it updates \( k \)-distances, reach-dist, lrd and LOF values for affected existing points.

**ii. Deletion.** In data stream applications it is sometimes necessary to delete certain data records (e.g., due to their obsoleteness). Very often, not only a single data record is deleted from the data set, but the entire data block that may correspond to particular outdated behavior. Similarly like in an insertion, upon deleting the block of data records \( S_{\text{delete}} \) there is a need to update parameters of the incremental LOF algorithm.

This algorithm has the same detection performance as the static “iterated” LOF algorithm that is applied after insertion of each data record, but it is much more computationally efficient.

**Clustering based outlier detection:** Clustering based outlier detection is used for intrusion detection [8]. In anomaly (outlier) intrusion detection, how to model the normal behavior of activities performed by a user is an important issue. To extract the normal behavior as a profile, conventional data mining techniques are widely applied to a finite audit data set. However, these approaches can only model the static behavior of a
user in the audit data set. This drawback can be overcome by viewing the continuous activities of a user as an audit data stream. [8] propose an anomaly detection method based on clustering a data stream.

**Clustering algorithm:** In this algorithm[8], input parameters are a data stream, a minimum deviation, and an initial cluster number. Each object is generated as a cluster with respect to the initial cluster number. And then, when a new object occurs, the following processes are performed. A cluster whose center is closest to a new object is selected from the cluster set and the properties of the cluster are updated. When the standard deviation of all the objects in any two adjacent clusters is less than or equal to the minimum deviation, two clusters are merged. When the standard deviation of any cluster becomes larger than the minimum deviation, the cluster is split into two clusters and then newly generated clusters are inserted into the cluster set $X^k$.

**Anomaly Detection:** For each feature[8], the on-going result of clustering is summarized in a profile, which is composed of the two properties of each cluster, a center and a standard deviation. An anomaly in a newly occurring object can be identified by comparing the new object with the current profile of each feature. For this purpose, as a new object occurs from a data stream, if the difference between the object and its closest cluster becomes large, this object is considered as an anomaly. Here new objects are continuously reflected to both the on-going result of clustering and the profile at the same time. Therefore, an anomaly can be detected easily without additional processes.

**Outlier detection on probabilistic data stream:** Distance-based outlier detection techniques on deterministic data have been extensively studied in the areas of network intrusion detection, event detection in wireless sensor networks and so on. As one of the emerging database techniques, uncertain data, this concerns the uncertainty of data and can reflect the real world better. Due to the intrinsic differences between uncertain and deterministic data, the existing techniques on deterministic data cannot be applied on outlier detection on uncertain data directly. Moreover, in the sliding window computation model, when a new element arrives, it should be inserted into the window and the expired element should be removed from the window.

When considering deterministic data, an element is a distance-based outlier if the number of its neighbor elements (including itself) within distance $d$ is below a threshold $k$. For example, a window contains the elements $e_1, e_2, e_3, e_4$ and $e_5$, if the probabilities of all elements are 1, all elements can be regarded as deterministic data. Then $e_5$ has four neighbor elements including $e_1, e_2, e_4$ and $e_5$ within $d = 50$ from it. Thus, when $k$ is 5, $e_5$ is an outlier. However, on an uncertain data stream, each element has a probability value, which result in the number of neighbor elements is uncertain. Thus, distance-based outlier on uncertain data stream cannot be defined just based on the number of neighbor elements.

A new definition of distance-based outlier on an uncertain data stream was proposed by [9]. Keeping the basic idea of the traditional definition of distance-based outlier; they employed probability in new definition. After each sliding step, an element $e$ is regarded as a distance-based outlier if the probability that the number of neighbor elements of $e$ in current window is less than or equal to a threshold $\lambda$.

Outliers in a single window are detected by processing all elements in current window sequentially. For each item they unfolded all possible worlds of its neighborhoods, and sum up the probabilities of possible worlds containing at least $k$
elements. \( k \) is used to denote the minimum number of enough neighbor elements. If this summation is not larger than \( \lambda \), then the element is an outlier.

The above naive approach is infeasible in realistic due to the following two reasons: (i) when the size of sliding window is large, it costs too much time to process all the elements contained in the window; (ii) the number of possible worlds for each element could be exponentially growing with the increasing number of its neighborhoods and it will be too expensive to process the element by unfolding all the possible worlds. In order to detect such distance-base outliers on uncertain data streams, a more efficient approach is expected. Moreover, [9] proposed a dynamic programming algorithm named DPA, which can process each single element in linear time, avoiding expensively unfolding the possible worlds of its neighborhoods.

The algorithm DPA was designed to support online computation in the presence of rapid updates of data elements. In the sliding window computation model against data streams, they considered the most \( m \) data elements, that is, the oldest \( l \) elements in an \( l \)-sliding window should be deleted and the arriving \( l \) elements should be inserted. Detected elements were utilized to incrementally detect outliers when sliding the window dynamically. When we slide the window for 1 step, the non-expired elements in the previous window will share the new window, for example, \( e_2; e_3; e_4 \), and \( e_5 \) are non-expired elements in the 1-sliding window in then such elements are called as joint elements. A pruning-based approach (PBA) was used to filter non-outliers on single window and dynamically detect recent \( m \) elements.

### 5.4 Frequent pattern based infrequent pattern mining

In this section a new algorithm is proposed to mine infrequent patterns over data stream which based on the concept frequent pattern mining. Here Dynamic FP-tree (section 4.6), introduced in chapter-4 is used for mining purpose. A support factor is used to mine infrequent patterns. As compared to frequent patterns length of infrequent patterns is quite small. On-line mining of transaction records of each customer to determine the least purchased items can be a very good example of infrequent mining. Proposed work is based on the same application. Transaction streams are mined to output association between least bought items. Here a data stream arriving as a time ordered series of transitions is considered for analysis and is denoted as \( D = \{ t_1, t_2, \ldots, t_n, \ldots \} \). Each transaction \( t_i \) contains a set of items \( a_i \) and \( a \in I \), where \( I = \{ a_1, a_2, \ldots, a_3 \} \) is a set of items or objects and \( t_n \) is called the current transaction arriving on the stream. Here let \( TSI_0, TSI_1, \ldots, TSI_{k+1}, \ldots, TSI_i \) denote the time period or time slot which contains multiple transactions arriving in that interval and thus they form a partition of transactions in the stream. Given an integer \( k \), the time based sliding window is defined as the set of transactions arriving in the last \( k \) time periods and denoted by \( SW= \{ TSI_i, TSI_{k+1}, \ldots, TSI_{i-1}, TSI_i \} \). \( TSI_i \) is called the latest time slot and \( TSI_{i-k} \) is called the expiring one in the sliding window. When the time shifts to the new slot \( TSI_i \) the effect of all transactions in \( TSI_{i-k} \) will be eliminated from the mining model.

**Algorithm 5.3** : Proposed algorithm works for both dense as well sparse streams of transactions. It does not require knowing number of items per transaction .It works for fixed as well as variable length transactions.
Input: an incoming batch of transactions arriving one at a time from the buffer and min_support threshold $\sigma$
Output: Infrequent patterns

4. Initialize the dynamic FP tree by assigning null value to first pointer.
5. Insert each incoming transaction as a link list without pruning any item.
   a. initialize the first pointer with the first item of first transaction.
   b. If the first pointer is not null then if the prefix of the transaction matches with the existing prefix of the tree then increase the counters of the prefixes.
   Else
      Add new link list of transaction at that level.
6. Traverse the tree by adding its data to stack and storing only those items to the file which falls below the min_support threshold accepted by user. As the data is popped out of stack it is deleted by freeing the memory.

5.5 Performance Study and Experiments

All the experiments are performed on a Pentium PC machine with 2 GB main memory, running Microsoft Windows-XP. All the methods are implemented using Microsoft Visual C++. Program details are available in Appendix-E. As the tree is dynamic it does not need to know in advance number of items. As a new item comes a node is created and inserted. Once the node is traversed it is deleted by freeing the memory, so data structure used is memory efficient can create patterns of any length. It is also time efficient as it uses only one data structure as compared to [11] which uses FP-tree as well FP stream. There is no need to create a flist as it is done in [11] items are directly inserted into the tree no intermediate data structure required.

For experiments two datasets are downloaded from fimics.helsinki.fi/data, Frequent Item set mining dataset repository, and are treated as transaction stream to test the data structure as well as algorithm.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#items</th>
<th># Avg. Length</th>
<th># Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>75</td>
<td>37</td>
<td>3,196</td>
</tr>
<tr>
<td>Pumsb</td>
<td>2113</td>
<td>74</td>
<td>49,046</td>
</tr>
</tbody>
</table>

Table 5.1 Data stream Characteristics

The details of selected datasets are summarized in above table. The infrequent pattern mining programs have been tested on each data stream with varied min_sup. Two user-specified parameters are accepted: a support threshold $s \in (0, 1)$, and an error parameter $e \in (0, 1)$ such that $e \ll s$. The error bound and the threshold parameter is used in
determining which pattern to archive. A pattern is archived if the sum of its frequency is less than or equal to difference between error and support threshold multiplied by the number of elements in the time window i.e. \((f \geq (s-e)N)\).

Experiments are conducted using support value 0.1 and error bound is kept 0.01 throughout. Window size i.e. number of transactions per window are kept 50.

The following graph shows infrequent patterns generated of chess dataset with support threshold 0.1 and error bound = 0.01.

![Infrequent patterns over the dataset Chess](image)

Fig-5.5 Infrequent Patterns over Chess transaction stream

Pattern p1 (6-7-9-11) occurs only three times among 150 transactions, and pattern p2 (64-66-68-70-71-74) occurs only five times among 250 transactions and pattern p3 (70-73-74) occurs only five times among 250 transactions.

In case of Pumsb dataset item id 7114 has occurred only ten times among 37,150 transactions. As shown in the fig-5.6 and pattern p1 (75-84-111) has occurred only once over the time window \(t_1\) to \(t_3\). And pattern p2- (155-161-163-168-170-180-184-188-197) occurred only thrice over the time window \(t_1\) to \(t_{10}\) and pattern p3 (161-163-168-170-180-184-188) has occurred 8 times respectively. Following graph fig 5.7 gives the frequency of these patterns over the dataset with support= 0.1 and error bound= 0.01.
Results in the experiments reported above are obtained by executing the program on windows XP platform which gives the run time in seconds. Infrequent patterns are stored on a secondary storage with each line pertaining to one time window. Frequent patterns are discarded by freeing the memory.

5.6 Conclusion

In the past, frequent pattern mining has been investigated in detail with little research being done in infrequent pattern mining. Mining of infrequent items has its good application in intrusion detection, market basket analysis, outlier detector etc. Above work is based on the Dynamic FP-tree introduced in chapter 4. Here pointers are used as main programming component requiring some time for allocation and deallocation of memory. But as the data structure is non recursive it is memory efficient and can work efficiently with variable as well as fixed length transactions. Algorithm is time efficient as the whole tree is not traversed only part of it satisfying the min_support threshold is traversed and rest is discarded.

An article based on this work is published in an international journal. [10].
5.7 References


[11] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, Philip S. Yu “ Mining Frequent Patterns in Data Streams at Multiple Time Granularities”, Data Mining, Next Generation Challenges and Future Directions, PHI