Implementation of Utility Mining Algorithms to find Frequent Itemsets from Transactional Database

Ch.Chakradhara Rao\textsuperscript{1}, A.V.Ramana\textsuperscript{2} & R.Soujanya\textsuperscript{3}

\textsuperscript{1}Assistant Professor, CSE Department, GMRIT, Rajam
\textsuperscript{2}Assistant Professor, IT Department, GMRIT, Rajam
\textsuperscript{3}IVth B.Tech, CSE Department, GMRIT, Rajam

Abstract: Mining high utility itemsets from a transactional database refers to the discovery of itemsets with high utility like profits. To overcome the problems of producing large number of candidate itemsets [4],[5], we are implementing three algorithms like Utility Pattern-Growth[2], Chernoff-based[3] and the Top-K Lossy Counting Algorithm[1]. UP Growth algorithm facilitates mining process by maintaining essential information in UP-Tree (Utility Pattern Tree). High-utility itemsets can be generated from UP-Tree efficiently with only two scans of original databases [2]. In case of Chernoff Bound Algorithm, user specifies a bound for better optimization of the dataset and to reduce the generated frequent itemsets [3]. In case of Top-K Lossy Counting Algorithm, the main parameter is frequency (quantity) of the item. Based on the frequency count and support of the item we generate unpromising items [1]. These algorithms produce small candidate itemsets unlike other algorithms that produce large candidate itemsets. These improve the mining performance in terms of execution time and space requirement. The objective of this work is to mine the frequent itemsets of a database using the above mentioned three algorithms. Based on the results, the most significant algorithm is judged.

1. Introduction.

Data mining of uncertain data has become an active area of research recently. Data mining is the process of revealing nontrivial, previously unknown and potentially useful information from large databases. Discovering useful patterns hidden in a database plays an essential role in several data mining tasks, such as frequent pattern mining, weighted frequent pattern mining, and high utility pattern mining. Among them, frequent pattern mining is a fundamental research topic that has been applied to different kinds of databases, such as transactional databases, streaming databases, and time series databases, and various application domains, such as bioinformatics, Web click-stream analysis, and mobile environments[2].

In frequent itemset mining, frequent sets play an essential role in many data mining tasks that try to find interesting patterns from databases, such as association rules[7], correlations, sequences, episodes, classifiers and clusters. A set is called frequent if its support is no less than a given absolute minimal support threshold $\text{min}_\text{sup}$ with $0<\text{min}_\text{sup}\leq|D|$. When working with frequencies of sets instead of their support, we use the relative minimal frequency threshold $\text{min}_\text{suprel}$, with $0<\text{min}_\text{suprel}<1$. The task of discovering all frequent sets is quite challenging. The search space is exponential in the number of items occurring in the database and the targeted databases tend to be massive, containing millions of transactions. Both these characteristics make it a worthwhile effort to seek the most efficient techniques to solve this task[9].

Mining frequent patterns (or itemsets) are probably one of the most important concepts in data mining. A lot of other data mining tasks and theories stem from this concept. It should be the beginning of any data mining technical training because, on one hand, it gives a very well-shaped idea about what data mining is and, on the other, it is not extremely technical.

The task of discovering all frequent itemsets is quite challenging. The search space is exponential in the number of items occurring in the database. The support threshold limits the output to a hopefully reasonable subspace. Also, such databases could be massive[8], containing millions of transactions, making support counting a tough problem. In this paper, we will analyze these two aspects into further detail. Throughout the last decade, a lot of people have implemented and compared several algorithms that try to solve the frequent itemsets mining problem as efficiently as possible. Unfortunately, only a very small selection of researchers put the source codes of their algorithms publicly available such that fair empirical evaluations and
comparisons[12] of their algorithms become very difficult[6],[2]. Moreover, we experienced that different implementations of the same algorithms could still result in significantly different performance results[6]. As a consequence, several claims that were presented in some articles[10] were later contradicted in other articles[11]. The original motivation for searching frequent sets came from the need to analyze so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Frequent sets of products describe how often items are purchased together. This paper shows the implementation of the above said three algorithms and the results are observed.

2. Utility Pattern Growth.

UP Growth algorithm is a weighted association rule mining technique. In this framework, weights of items, such as unit profits of items in transaction databases, are considered. With this concept, even if some items appear infrequently, they might still be found if they have high weights. Mining high utility itemsets from databases refers to finding the itemsets with high profits. Here, the meaning of itemset utility is interestingness, importance, or profitability of an item to users. Utility of items in a transaction database consists of two aspects: 1) the importance of distinct items, which is called external utility, and 2) the importance of items in transactions, which is called internal utility. Utility of an itemset is defined as the product of its external utility and its internal utility. An itemset is called a high utility itemset if its utility is no less than a user-specified minimum utility threshold; otherwise, it is called a low-utility itemset. Mining high utility itemsets from databases is an important task with a wide range of applications such as website clickstream analysis, business promotion in chain hypermarkets, cross marketing in retail stores, online e-commerce management, and mobile commerce environment planning, and even finding important patterns in biomedical applications[2],[6]. This method suffers from the problems of a large search space, especially when databases contain lots of long transactions or a low minimum utility threshold is set. UP-Growth algorithm identifies high utility itemsets using a compact tree structure, called UP-Tree. UP-Tree maintains the importance of the transaction database related to utility pattern. High utility itemsets are then generated from the UP-Tree efficiently with only two scans of the database[2]. The main framework of this proposed method consists of 3 steps as:

(1) Construction of UP-Tree
(2) Generation of potential high utility itemsets from the UP-Tree by UP-Growth, and
(3) Identification of high utility itemsets from the set of potential high utility itemsets.

The UP-tree is constructed in two scans of the database. The first scan collects and sorts the set of high utility items, and the second constructs the UP-Tree.

Definitions

I. Utility of an item ip in a transaction Td is denoted as u(ip;Td) and defined as pr(ip) * q(ip;Td).
II. Utility of an itemset X in Td is denoted as u(X;Td) and defined as \( \sum_{ip} \times X \_Td \_u(ip;Td) \).
III. Utility of an itemset X in D is denoted as u(X) and defined as \( \sum_{X \_Td}^T \_Td2D u(X;Td) \).
IV. An itemset is called a high utility itemset if its utility is no less than a user-specified minimum utility threshold which is denoted as min_util. Otherwise, it is called a low-utility itemset.
V. Transaction utility of a transaction Td is denoted as TU(Td) and defined as u(Td;Td).
VI. Transaction-weighted utility of an itemset X is the sum of the transaction utilities of all the transactions containing X, which is denoted as TWU(X) and defined as \( \sum_{X \_Td}^T \_Td2D u(TU(Td)) \).
VII. An itemset X is called a high-transaction weighted utility itemset (HTWUI) if TWU(X) is no less than min_util.
VIII. An item ip is called a promising item if TWU(ip) < min_util. Otherwise it is called an unpromising item. Without loss of generality, an item is also called a promising item if its overestimated utility is no less than min_util. Otherwise it is called an unpromising item.

UP Growth algorithm

Input: A transaction database DB, profit table p and a minimum utility threshold min_util.
Output: Unpromising items.
Steps:
Step 1: for each item Ip in Transaction Td compute
Step 2: u(Ip, Td)= pr(Ip) * q(Ip;Td);
//p(Ip) value is given in profit table and q(Ip,Td) value is in DB
Step 3: end for
Step 4: for each transaction Td in DB compute
Step 5: TU(Td) = u(Td, Td);
//where u(X, Td) = sum of u (Ip, Td), Ip belongs to X and X is subset of Td
Step 6: end for
Step 7: for each item Ip in DB compute
Step 8: TWU(Ip) = sum of TU(Td);
//where Ip is subset of Td, Td belongs to DB
Step 9: end for
Step 10: Discarding all unpromising items
//TWU(ip) < min_util are unpromising items.
Step 11: Unpromising items are removed from
transactions in DB and reorganized TU vales RTU are computed.
Step 12: The remaining promising items are sorted in
decreasing order of the TWU values.
The results for Upgrowth is as shown below:


Top-K Lossy Counting Algorithm does not make any
assumption about data independence and data
distribution. The data are divided into a number
of batches of R transactions each for efficiency purpose.
In each batch, we further divide the transactions into
a number of buckets.
The data are divided into a number of batches of R
transactions each for efficiency purpose. In each
batch, we further divide the transactions into
a number of buckets[1]. Let β be the number of
buckets in a batch. The algorithm stores the itemsets
set in the pool Fl find all itemsets with frequency
greater than or equal to β in the current batch and
store them in the pool Pl. Then, we merge the entries
in pool Pl to the pool Fl. We update the support of
each entry stored in Fl. We repeat the above process
for the remaining batches.

Top-K-Lossy Algorithm

Input: A transaction database DB, batch size, minimum support.
Steps:
Step 1: for each item Ip in Transaction Td compute
Step 2: Scan the data and calculate the frequency
count of each and every item.
Step 3: freq_count (id, Td) = freq_count (id, Td-1) +
quantity (id, Td).
Step 4: end for
Step 5: for each transaction Td in DB compute
Step 6: Calculate the min frequency count of all the
transactions.
Step 7: Min frequency(β) = ∑ (minimum quantity in
each transaction) / batch size.
Step 8: end for
Step 9: for each item Ip in DB compute
Step 10: if freq count< minfreq, update pool as 0
otherwise as 1.
Step 11: end for
Step 12: for each item Ip in DB compute
Step 13: Support of item id calculated as:
Support (id, Td) = (freq_count (id) in transaction Td) / d
Step 14: if support<min_sup make item as
unpromising.
Step 15: end for
Step 16: Unpromising items are removed from
transactions in DB and frequency count of the items
are made to 0.
Step 17: The remaining promising items are stored in
global pool.
Output: Unpromising items.
The results for Top-K-Lossy is as shown below:
4. Chernoff Bound Algorithm.

The Chernoff-based Algorithm is controlled by two parameters: minimum frequency and minimum support for error bounds and reliability. Minimum frequency is determined automatically when data arrives. It will be close to zero when more and more data is read. The other parameter minimum support determines the confidence that the results returned are the true top K frequent itemsets. It is expected that the Chernoff-based Algorithm returns accurate results when the data is independent. In our algorithm, we divide the itemsets into two groups—potential K-frequent itemsets and unpromising itemsets[3]. Unpromising itemsets will be pruned regularly to keep the memory usage low.

For efficiency, we process the data in batches. For every batch B of R transactions, we keep in the pool Pl the potential K-frequent l-itemsets in terms of R in batch B with respect to minimum frequency found in B. Let n be the number of transactions examined so far in the data. Then, we combine the pool Pl with a global pool Fl. We update the support count of each entry in the global pool Fl. If the number of entries in the global pool Fl is greater than the storage capacity given by n0, l, then we prune the unpromising itemsets in terms of n in Fl according to the minimum frequency value found in the n transactions read so far. We repeat the process for the remaining batches.

**Chernoff Bound Algorithm**

**Input:** A transaction database DB, batch size, minimum support and minimum frequency.

**Steps:**
- Step 1: for each item Ip in Transaction Td compute
- Step 2: Scan the data and calculate the support count of each and every item.
- Step 3: s(id,Td)=quantity of id in Td/sum of quantity of all items in Td.
- Step 4: end for
- Step 5: Obtain the value of minimum frequency.
- Step 6: for each item Ip in DB compute.
- Step 7: if support count< min_freq, update pool as 0 otherwise as 1.

5. Experiments.

For testing efficiency of Utility Pattern Growth, Top-K-Lossy Counting and Chernoff algorithms, various
datasets are available at UCI repository of machine learning databases. This paper uses transactional dataset and a brief description of dataset used in experiment evaluation Table 1 shows some characteristics of dataset as the test datasets.

<table>
<thead>
<tr>
<th>Dataset-1</th>
<th>Number of Items</th>
<th>Number of records/Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

In this paper, we have used NetBeans IDE to implement the above stated algorithms and tested the efficiency i.e, running time and memory taken by each algorithm on given datasets. The results are as shown in Table 2.

<table>
<thead>
<tr>
<th>Efficiency Type</th>
<th>UP-Growth</th>
<th>Top-K-Lossy</th>
<th>Chernoff Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Time (ms)</td>
<td>0.1870</td>
<td>0.2820</td>
<td>0.3120</td>
</tr>
<tr>
<td>Memory Used (kb)</td>
<td>7955.91</td>
<td>8215.85</td>
<td>8235.64</td>
</tr>
</tbody>
</table>

6. Conclusion.

Frequent sets play an essential role in many Data Mining tasks that try to find interesting patterns from databases. The original motivation for searching frequent sets came from the need to analyze so called supermarket transaction data. Mining frequent itemsets from large datasets degrades the performance in terms of execution time and space requirement. To overcome this drawback we have implemented three algorithms like UP-Growth, Chernoff based and the Top-K Lossy Counting Algorithm. The main aim of this paper is to find the efficient algorithm to find frequent itemsets using utility of the items. Every algorithm has their own importance and we use them on the behavior of the data, but on the basis of this research we found that Utility Pattern Growth algorithm is simplest algorithm as compared to other algorithms because it takes less time and space to get the results.

7. References.