International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-1 Issue-5, June 2012

Performance based Frequent Itemset Mining Techniques for Data Mining

Sanjaydeep Singh Lodhi, Ghanshyam Rathore, Premnarayan Arya

Abstract*— Data mining tasks that try to find interesting patterns from databases, such as association rules, correlations, sequences, episodes, classifiers, clusters and many more of which the mining of association rules is one of the most popular problems. There is a large body of research on Frequent Itemset Mining (FIM) but very little work addresses FIM in uncertain databases. Most studies on frequent itemset mining focus on mining precise data. However, there are situations in which the data are uncertain. This leads to the mining of uncertain data. There are also situations in which users are only interested in frequent itemsets that satisfy user-specified aggregate constraints. This leads to constrained mining of uncertain data. Moreover, floods of uncertain data can be produced in many other situations. This leads to stream mining of uncertain data. In this paper, we propose algorithms to deal with all these situations. We first design a tree-based mining algorithm to find all frequent itemsets from databases of uncertain data. We then extend it to mine databases of uncertain data for only those frequent itemsets that satisfy user-specified aggregate constraints and to mine streams of uncertain data for all frequent itemsets. Our experimental results show the more effectiveness than existing methods.

Keywords: Data Mining, Frequent Itemset Mining, Apriori Algorithm.

I. INTRODUCTION

Frequent itemset mining is an essential role in data mining, which looks for implicit, previously unknown, and potentially useful information from data. Introduced the research problem of finding frequent itemsets from traditional databases consisting of precise data, it has been the subject of numerous studies. Previous studies can be broadly divided into two categories: those that focus mainly on performance and those that focus mainly on functionality. The studies in the first category aim to explore how to compute the frequent itemsets as efficiently as possible. The studies in the second category target the questions such as which kinds of patterns to compute and where to mine frequent itemsets. Regarding the studies that focus on performance, a well-known frequent itemset mining algorithm, called Apriori algorithm was proposed in 1994. It depends on a generate-and-test approach. To find frequent itemsets from transaction database, Apriori and its extensions (i.e., Apriori-based algorithms) first generate candidates and then check the occurrences (supports) of these candidates against the transaction database. The candidates with their supports over the user specified minimum support threshold are counted as valid frequent itemsets. To avoid memory intensive candidate generation, Han et al. proposed a tree-based frequent itemset mining algorithm called FP-growth (Frequent Pattern growth). The algorithm first scans the transaction database once to find all frequent 1-itemsets and sorts them according to some criteria (e.g., in frequency descending order). Then, it scans the transaction database again to construct an extended prefix tree, called FP-tree (Frequent Pattern tree), which captures all frequent items in the transaction database. Based on the FP-tree, the FP-growth algorithm mines frequent itemsets by a test-only approach, called frequent pattern growth, which only tests the support for each itemset. As a result, all frequent itemsets are found without candidate generation. As a preview, all the mining algorithms we are going to propose in this thesis use variants of the FP-tree. Our algorithms also avoid generating candidates and only test for support. Regarding the studies that focus mainly on functionality, they aim to find interesting patterns other than frequent itemsets. Frequent itemset mining has played an important role in the mining of these patterns. However, most of the studies on frequent itemset mining rely on a computational model, in which data mining algorithm does almost everything and human users are not engaged in the mining process. In other words, these data mining algorithms provide little or no support for user focus. In many real-life applications, the user may be only interested in a tiny portion of mined data.

We have discussed so far are related to the mining of frequent itemsets from traditional static databases. Over the past decade, the automation of measurements and data collection has produced tremendously huge amounts of data in many real-life application areas. The recent development and increasing use of a large number of sensors has added to this situation. Consequently, these advances in technology have led to a flood of data. Algorithms for mining these dynamic streams are in demand. This calls for stream mining. When comparing with mining from traditional static databases, mining from data streams is more challenging due to the following two properties of data streams:

Manuscript published on 30 June 2012.
***** Correspondence Author (s)
Sanjaydeep Singh Lodhi, Department of Computer Application (Software Systems), S.A.T.I (Govt. Autonomous collage) , Vidisha, (M.P), India
Ghanshyam Rathore, Department of C.S.E. IIST, Indore (M.P), India.
Premnarayan Arya, Asst. Prof. Dept. of CA (Software Systems), S.A.T.I (Govt. Autonomous collage) , Vidisha, (M.P), India.

1. Data streams are continuous and unbounded. To find frequent itemsets from data streams, we no longer have the luxury of performing multiple data scans. Once the streams flow through, we lose them. Hence, we need some techniques to capture the important contents of the streams (e.g., to capture recent data because users are usually more interested in recent data than older ones) and ensure that the captured data can fit into memory.
2. Data in the streams are not necessarily uniformly distributed and their distributions are usually changing with time. A currently infrequent itemset may become frequent in the future, and vice versa. So, we have to be careful not to prune infrequent itemsets too early. Otherwise, we may not be able to get complete information such as frequencies of some itemsets (as it is impossible to recall those pruned itemsets).

To find frequent itemsets from data streams, several stream mining algorithms have been proposed.

In this paper, we first propose a tree-based algorithm for further enhancing the performance of U-Apriori algorithm. The key contributions of this part of work include the following: (a) the proposal of an effective tree structure called UF-tree for capturing the contents of transactions consisting of uncertain data, and (b) the development of an efficient algorithm called UF-growth—for mining frequent itemsets from the proposed UF-tree.

The UF-growth algorithm can find all frequent itemsets from uncertain data. However, as in the case for frequent itemset mining from traditional precise data, there are many real-life situations in which the user is interested in only some subsets (and may be some tiny subsets) of all the frequent itemsets. This phenomenon also widely exists in the uncertain cases.

II. BACKGROUND TECHNIQUES

Data Mining:

Data mining is the automated process of extracting useful patterns from typically large quantities of data. Different types of patterns capture different types of structures in the data: A pattern may be in the form of a rule, cluster, set, sequence, graph, tree, etc. and each of these pattern types is able to express different structures and relationships present in the data. For example, a rule may tell a marketer about strong relationships between purchased goods or services, predict customer 'churn' or be used as the basis of recommender systems. A set can indicate products that customers are purchasing together and a cluster might tell him or her about groups of customers that have similar purchasing patterns. These may be used as the basis for a marketing scheme that aims to maximize response rates and sales for a given investment. A graph may tell a biologist about strong and previously unknown gene or protein interactions present in their experiments, or a security agent about suspicious communication structures between potential criminals. A tree or a set of rules may describe the decision structure that can be used to accurately predict medical conditions, perhaps based on patient records or medical imaging data. As suggested by these examples, patterns can be descriptive or predictive (or both). That is, they can be used to describe, model and help better understand a process or phenomena or to predict future events. In this work, many new types of patterns are introduced. Some are purely descriptive and some are explicitly used for prediction purposes.

Frequent Itemset Mining:

The world around us is full of information and contemporary computer systems are allowing us to gather and store that information at an astounding rate. However, our ability to process that information lags far beyond our abilities as gatherers. Most of the truly amazing problems of our time are rooted in extracting the meaningful patterns from datasets so large as to have previously been unfathomable. The human genome has been sequenced but it is still uncertain which parts of it cause us to express particular phenotypes. Hundreds of weather stations around the world have been collecting information about local climate for decades, but it is still difficult to predict the effects of human activities. The universe beyond our stratosphere and the Internet within it are mysterious places, although we have terabytes of data collected from each.

Apriori Algorithm Concept:

In computer science and data mining, Apriori is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions, or having no timestamps (DNA sequencing). As is common in association rule mining, given a set of *itemsets* (for instance, sets of retail transactions, each listing individual items purchased), the algorithm attempts to find subsets which are common to at least a minimum number C of the itemsets. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as *candidate generation*), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found. Apriori uses breadth-first search and a tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k - 1$. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates. Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2^{|S|} - 1$ of its proper subsets.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

FP-Growth Algorithm:

In recent, Han et al. introduce a quite novel algorithm to solve the frequent itemset mining problem. They adapt the idea of a trie to the set of transactions rather than candidates. In so doing, they effectively compress the dataset D with the hope that it will fit entirely in main memory. Each transaction is inserted into the trie in its most-frequent-first order and at each node of the trie is stored a support counter. When a new transaction t is inserted, a path of size |t| is traced; the count at each node along this path is incremented. Thus, inserting the transaction involves updating |t| support counts. Additionally, a linked list is maintained between all nodes sharing the same label. In this way, one can quickly find all paths that involve the same item. Next, the trie is mined recursively to extract the frequent itemsets. By following the linked-list of nodes labeled by the least-frequent item, one retrieves all paths involving that item. Then a new conditional prefix tree can be built by copying and then modifying the original tree. All paths whose leaves are not labeled with the least-frequent item are removed, this least-frequent item is itself removed, duplicate paths are merged, and the trie is resorted based on the new conditional frequencies. This creates a trie with the same structure as the original tree, but conditioned on the presence of the least frequent item. So, the procedure can be repeatedly recursively from here until the trie consists of nothing but a root node denoting the empty set. This yields all frequent itemsets involving the least-frequent item. The procedure is then repeated for the second-least-frequent item, third-least frequent item, and so on to extract from the trie all frequent itemsets.

The data structure is quite cute and appears to eliminate the construction of candidates entirely. Indeed, experimental results have demonstrated consistently that it significantly outperforms A Priori. However, the story changes when the dataset is quite large because it suffers the same consequences as did the trie of candidates.

Even building the trie becomes extremely costly, to the point that it is remarked that the dominant percentage of execution time is that of constructing the trie. Consequently, on truly large datasets, the FPGrowth algorithm fails even to initialize.

Uncertain Data Model:

The uncertain data model applied in this paper is based on the possible worlds semantic with existential uncertain items.

Definition 1:

An uncertain item is an item $x \in I$ whose presence in a transaction $t \in T$ is defined by an existential probability $P(x \in t) \in (0; 1)$. A certain item is an item where $P(x \in t) \in f0; 1g$. I is the set of all possible items.

Definition 2:

An uncertain transaction t is a transaction that contains uncertain items. A transaction database T containing uncertain transactions are called an uncertain transaction database. An uncertain transaction t is represented in an uncertain transaction database by the items $x \in I$ associated with an existential probability value $P(x \in t)$ 2 (0; 1].

Example uncertain transaction databases are depicted. To interpret an uncertain transaction database we apply the possible world model. An uncertain transaction database generates possible worlds, where each world is defined by a fixed set of (certain) transactions. A possible world is instantiated by generating each transaction $ti \in T$ according to the occurrence probabilities $P(x \in ti)$. Consequently, each probability $0 < P(x \in ti) < 1$ derives two possible worlds per transaction: One possible world in which x exists in ti, and one possible world where x does not exist in ti. Thus, the number of possible worlds of a database increases exponentially in both the number of transactions and the number of uncertain items contained in it [9].

Uncertain Graphs:

Recently, there has been a growing interest in using uncertain graphs as a data model in applications that need to deal with uncertainty. Thus, various problems for mining uncertain graphs have emerged. The problem of finding reliable sub-graphs in uncertain graphs is studied. Given a graph that is subject to random edge failures, the goal is to find and remove a number of edges so that the probability of connecting a set of selected nodes in the remaining sub-graph is maximized. Three novel types of probabilistic path queries have been defined for uncertain graphs representing road networks, where edge probabilities capture the uncertainty in traffic conditions. Also, both exact and approximation algorithms are introduced to answer such queries. A generalization of *k*-Nearest Neighbor queries in uncertain graphs is presented, where a framework is proposed considering alternative ways to define the distance between nodes taking edge probabilities into account. All these works clearly show the increasing need and interest in mining uncertain graphs. However, to the best of our knowledge, up to now only one work has dealt with the problem of frequent subgraph pattern mining in uncertain graphs. The proposed method is an approximation algorithm, called MUSE, which allows for a tradeoff between accuracy and efficiency when computing the expected support of candidate sub-graph patterns. In particular, given a support threshold *minSup* and a relative error tolerance $\varepsilon \in [0, 1]$, the algorithm returns all subgraph patterns with expected support at least *minSup*, allowing also for some false positives with expected support in the range $[(1 - \varepsilon)\ minSup,\ minSup]$. Similar to corresponding methods for exact graphs, the solution addresses two main subtasks: (a) a method for enumerating candidate patterns, and (b) a method to compute the expected support of a pattern. Regarding the first task, the method proposed in g Span is adopted to construct a search tree of subgraph patterns. For the second task, two algorithms are proposed, an exact one for small instances of the problem (e.g., graphs with up to 30 edges) and an approximate one for larger instances. The main idea in both algorithms is to transform the problem to an instance of the DNF counting problem. Although this algorithm makes it possible to approximate the expected support of a candidate pattern for an uncertain graph with a large number of edges, the computational cost is still quite high, and therefore the method does not scale well, even for moderate size databases with up to a few hundreds of uncertain graphs.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

This limitation, by constructing an index of the uncertain graph database, which significantly prunes the search space and enables for additional optimizations based on early termination and efficient scheduling to avoid the expensive sub-graph isomorphism tests [2] and [11].

Sampling the Uncertain Dataset:

The first method proposed the Concatenating the Samples, takes the uncertain dataset and samples according to the given existential probabilities. For every transaction t and every item i in transaction t we generate an independent random number 0 <= r <= 1 (coin flip) and we compare it with the probability p associated with the item i. If p >= r then item i will appear in the currently sampled transaction. For every transaction in the uncertain dataset we repeat the step above n times, for a given n. The result is a dataset which can be mined with any traditional frequent itemset mining algorithm. To obtain the estimated support of an itemset in the uncertain dataset, its support in the sampled dataset still needs to be divided by n.

The difficulty of this method resides in the fact that we physically instantiate and store the sampled "certain" dataset which can be up to n times larger than the original uncertain dataset. Fortunately, for most efficient itemset mining algorithms, we do not actually have to materialize this samples database. After all, most efficient techniques read the database from disk only once, after which their advanced data structures contain the database in the main memory. Therefore, the sample can be generated immediately in memory when the database is being read from disk for the first time. This method is called Inline Sampling. To this end, we made minor modifications of the frequent itemset mining algorithms [11] and [13].

The U-Eclat and UFP-growth, the modified versions of the ECLAT and FP-growth algorithms. U-Eclat is an adaptation of the ECLAT algorithm with an improvement based on diff sets as described. In only one scan of the dataset the relevant items are stored into memory together with the list of transactions where the items appear, called tid-list. The candidates are then generated using a depth first search strategy and their support is computed by intersecting the tid-list of the subsets. The only adaptation for U-Eclat consists in reading the uncertain transactions and instantiating them as described above. More specifically, given the number of iterations n, for every transaction t and every item i in transaction t and generate n independent random numbers r1, . . . , rn between 0 and 1 and we compare them with the probability p associated with the item i. If p >= rj , for 1 <= j <= n, then n · t + j will appear in the tid-list of item i. From there on, the standard Eclat algorithm is being executed. The algorithm then extracts the frequent itemsets the same way as the FP-growth algorithm [12] and [13].

We propose and develop two tree-based algorithms called UF streaming for mining frequent itemsets from streams of uncertain data. The key contributions of this part of work in my thesis include: (a) the proposal of effective tree structures to capture the important contents of transactions in streams of uncertain data and (b) the development of two efficient algorithms to mine frequent itemsets from the transactions captured by these proposed tree structures.

III. PROPOSED TECHNIQUES

Frequent itemset mining from uncertain data:

U-Apriori is an Apriori-based algorithm for mining frequent itemsets from uncertain data. It is well-known that when handling precise data, the FP-based algorithms are superior in performance to their corresponding Apriori-based counterparts. Hence, in this chapter, we investigate how tree-based frequent itemset mining can be applied to uncertain data. Specifically, we propose a tree-based algorithm, called UF-growth, for mining frequent itemsets from uncertain data. The algorithm consists of two main operations: (a) the construction of a novel tree structure called UF-tree and (b) the mining of frequent itemsets from UF-trees. We also investigate the performance and functionality of our UF-growth algorithm when compared with the U-Apriori algorithm.

The Construction of UF-trees:

The key for many tree-based mining algorithms is how to represent and store data. In our case, how to represent uncertain data in each tree node and how to order these tree nodes are two main challenges. Note that precise data and uncertain data are different in many aspects. For precise data, each item in a database transaction is implicitly associated with a definite certainty of its presence in the transaction. In contrast, for uncertain data, each item is explicitly associated with an existential probability ranging from 0 (indicating that the item is not present in the database transaction) to a value of 1 (indicating that the item is definitely present). When this existential probability is close to 0, the chance of the item presented in database transaction is quite small. On the other hand, the chance of the item presented in database transaction is very high if this existential probability is close to 1. Moreover, the existential probability of an item can vary from one transaction to another. To perform frequent itemset mining from uncertain data, the key modification made by the U-Apriori algorithm to the Apriori algorithm is incrementing the support count of candidate itemsets by the expected support instead of the actual support.

Instead of storing and incrementing the actual supports of items in transactions at tree nodes (as in FP-growth); can we store and increment the expected supports for uncertain data? To answer this question, let us examine this approach. On the surface, this approach appears to work. For instance, it finds frequent itemsets of size 1 successfully. However, a close examination reveals that such an approach does not completely find all frequent itemsets. This means the expected support of a singleton itemset can be computed by using the sum of expected support. So, the approach works when mining frequent itemsets of size 1 because each tree node keeps the sum of expected support of an item. However, the approach does not work when mining frequent itemsets of size greater than 1 because the expected support of any itemset is computed as the sum of products of the expected support of items in such an itemset. If we only store and increment the expected supports, we miss the information for the product.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Therefore, to effectively represent uncertain data and find all frequent itemsets, we propose a variant of the FP-tree (i.e., a variant of an extended prefix-tree structure). We call it an UF-tree. Each node in the UF-tree stores (a) an item, (b) its expected support and (c) the number of occurrence of such expected support for such an item. Now, let us turn our focus to the ordering of tree nodes in the UF-tree. We know that the expected support of any itemset can be computed as the sum of products of the expected supports of items in the itemset. Note that the order of any portion in a product does not influence the result of the product. From this point of view, no matter how we order the tree nodes in the UF-tree, we can find the expected support of an itemset with the same efficiency. However, we observed from many other tree-based mining algorithms (i.e., FP-growth) that the ordering of tree nodes affects the tree size. Smaller trees consume less memory space than bigger ones. Bigger trees require more computation to travel the tree nodes than the smaller ones. We also know that when we handle precise data, ordering the items with descending frequency order can increase the chance of tree path sharing, which leads to reduction in the size of the tree. Applying a similar approach to our UF-tree, we order the tree nodes in descending order of accumulated expected supports to increase the chance of path sharing and reduce the size of the UF-tree. With the above uncertain data representation and the ordering of the tree nodes, let us take a close look at how our proposed UF-growth algorithm constructs the UF-tree. The algorithm first scans the database once and accumulates the expected support of each item. Hence, it finds all frequent items (i.e., items having expected support minsup). It then sorts these frequent items in descending order of accumulated expected supports. After that, the algorithm scans the database the second time and inserts each transaction into the UF-tree in a similar fashion as in the construction of an FP-tree. Specifically, for each transaction, frequent items are sorted according to the descending global expected support order and infrequent items are removed. New transactions are added at the root level. At each level, nodes of the new transaction are compared with children (or descendant) nodes. If the same item and same expected support exist in both the new transaction and the children (or descendant) nodes, the transaction is merged with the node at the highest support level. The occurrence count of the merged node is then incremented by 1. The remainder of the transaction (items with their expected support) is then added to the merged node, and this process is repeated recursively until all common items and expected supports are found. Any remaining items and expected supports of the transaction are added as a new branch to the last merged node. It is important to note that only the same items with same expected support can be merged in the UF-tree. The same items with different expected supports have to be in the different branches.

Our UF-tree processes the following nice property:

• The occurrence count of a node is greater than or equal to the sum of occurrence counts of all its children nodes. On the surface, the UF-tree may appear to require a large amount of memory. However, it is important to note that the UF-tree like the FP-tree is an extended prefix-tree structure that captures the contents of transactions. In the worst case, the number of nodes in a UF-tree is the same as the sum of items in all transactions in the original database of uncertain data. Moreover, thanks to advances in modern technology, we are able to make the same realistic assumption as in many studies that we have enough main memory space in the sense that the trees can fit into the memory. In the situations where the trees cannot fit into memory, recursive projections and partitioning are required to break the trees into smaller pieces.

Once the UF-tree is constructed, our proposed UF-growth algorithm recursively mines frequent itemsets from this tree in a similar fashion as in the FP-growth algorithm except for the following:

• When forming a UF-tree for the projected database for an itemset X, we need to keep track of the existential probability of X. The existential probability of X can be calculated as the multiplying results of the existential probabilities of each item in X.

• When computing the expected support of an “extension” of an itemset X (say, X [{y}), we need to first multiply the existential probability of X by the existential probability of y, and then multiply this results by their occurrence in each tree path. Finally, we sum all the multiplying results. Note that, if an itemset Z is an “extension” of an itemset X, then X is a prefix of Z. In other words, the “extension” Z can be formed by appending items to X (e.g., if X = {a, c}, then {a, c, d}, {a, c, e} and {a, c, d, e} are some “extensions” of X but {a, b, c} is not). To elaborate, the algorithm first finds the frequent itemsets of size 1 (singleton items) from the UF-tree, calculates the expected supports and keeps tracking the existential probability for each of them. For each frequent item x in the UF-tree, the UF-growth algorithm forms its projected database (i.e., a collection of transactions having {x} as its prefix) and builds a {x}-projected tree for this projected database. For each frequent item y in the {x}-projected tree, the algorithm forms the itemset {x, y} and calculates the expected support for this itemset. Here, the expected support for itemset {x, y} can be calculated as follows. In each path of the {x}-projected tree, the algorithm multiplies the existential probability of item x by the existential probability of item y. Then, the algorithm multiplies this result by the occurrence of itemset {x, y}. Finally, the algorithm sums the multiplying results from all paths to obtain the expected support of itemset {x, y}. If this expected support >= minsup, itemset {x, y} is frequent. In contrast, if this expected support < minsup, itemset {x, y} is infrequent.

Next, for each frequent item z in the {x, y}-projected tree, the algorithm calculates the expected support using the same approach we discussed above. This process is then applied recursively to each frequent item in the UF-tree for subsequent projected database. Similar to the FP-growth algorithm, the entire mining process of UF growth algorithm can be viewed as a divide-and-conquer approach of decomposing both the mining task and the transaction database according to the frequent itemsets obtained so far. This leads to a focused search of smaller data sets. For better understanding the mining approach of UF-growth algorithm.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Aggregate Constraint UF-growth (ACUF- grouth) Algorithm-

We propose a tree-based algorithm called ACUF-growth (indicates Aggregate Constraint UF-growth) to efficiently mine from uncertain data those frequent itemsets which satisfy the user-specified aggregate constraints. Our algorithm first scans the database with uncertain data once to accumulate the expected support of each of the domain items and finds those frequent ones. Then, the algorithm arranges these frequent items according to some order R. After that, the ACUF-growth algorithm scans the database the second time to build a modified UF-tree. Recall that the UF-tree is a tree structure to mine frequent itemsets from uncertain data. In the UF-tree, each node stores (a) an item, (b) its expected support and (c) the number of occurrences of such expected support for such an item. The order of the tree nodes in the UF-tree is based on the descending order of accumulated expected supports. In our modified UF-tree, each tree node contains exactly the same contents as the UF-tree. However, the order of the tree nodes is based on the order R, which is the only difference between a UF-tree and our modified UF-tree.

During the modified UF-tree construction process, our ACUF-growth algorithm only inserts those frequent items in each database transaction into the tree. As discussed above, the items in the tree are arranged according to some order R. Here, a new transaction is merged with a child (or descendant) node of the root of the modified UF-tree only if the same item and the same expected support exist in both the transaction and the child (or descendant) node. Note that the occurrence count of a node is at least the sum of occurrence counts of all its children nodes.

Once the modified UF-tree is constructed, our ACUF-growth algorithm recursively mines constrained frequent itemsets from this tree in a depth-first divide-and-conquer manner. Specifically, the algorithm first forms a modified UF-tree for the projected database of an item x (according to the order R) and keeps track of the expected support of x. From this tree, the algorithm recursively finds all constrained frequent itemsets containing x in a similar manner (i.e., by recursively forming a modified UF-tree for the projected database of {x, y} where y is an item in the {x}-projected database). This process is repeated for other items. By exploring nice properties of aggregate constraints, we avoid unnecessary constraint checking and formation of projected databases.

Algorithm for Mining Streams of Uncertain Data:

We proposed the UF-streaming algorithm for mining frequent itemsets from streams of uncertain data. There are some potential problems associated with such an approximate algorithm. First of all, the UF-streaming algorithm calls the UF-growth algorithm with preMinsup which is lower than the actual minsup. As a result, the algorithm finds "frequent" itemsets (i.e., itemsets with expected support expSup >= preMinsup). And we know that some of these itemsets are not truly frequent (e.g., some itemsets may have expected support expSup < minsup). Consequently, to find truly frequent itemsets, one needs to apply a post-processing step. Moreover, the success of UF-streaming strongly depends on the value of pre- Minsup. If it is too high (e.g., too close to minsup), we may lose frequency information of some itemsets. To another extreme, if it is too low, a lot of redundant itemsets (e.g., those itemsets with expected support higher than preMinsup but lower than minsup) may be generated and stored. So, similar to minsup, it is not easy to find an appropriate value for preMinsup. Second, the UF-streaming algorithm requires an extra data structure (the UF-stream structure) to store the mined itemsets. Third, the UF-streaming algorithm uses an "immediate" mode for mining, which leads to lots of computation wasting, especially when many batches flow in before the user requests for the mining results (frequent itemsets).

IV. RESULTS ANALYSIS

To evaluate our proposed algorithms that performs uncertain frequent itemset mining (i.e., UF-growth algorithm), constrained frequent itemset mining from uncertain data (i.e., ACUF-growth algorithm) and frequent itemset mining from streams of uncertain data (i.e., UF-streaming algorithm and SUF-growth algorithm), we performed several experiments.

We focus on our proposed UF-growth algorithm and its Apriori-based counterpart, namely the U-Apriori algorithm. In the first experiment, we tested the effect of minsup. We fixed the database size as 10k records with an average transaction length of 10 items and a domain of 1,000 items and varied the minsup. We compared the runtimes of two algorithms: (1) the U-Apriori algorithm and (2) our UF-growth algorithm. We observed that the runtime for our UF-growth algorithm is lower than the runtime for the U-Apriori algorithm. The U-Apriori algorithm relies on the costly candidate generation process. It explains why the experimental result showed that our UF-growth algorithm required much less runtime. We also repeated the same experiment with a database which contains 100k records with an average transaction length of 10 items and a domain of 1,000 items.

In the second experiment, we tested scalability with the number of transactions for our UF-growth algorithm. We fixed the minsup and the distribution of existential probability. The variant in this experiment is the size of transaction database. Specifically, we varied the size of transaction database from 10k transactions to 100k transactions. The experimental result showed that when the number of transactions increased, the runtime for our UF-growth algorithms increased. Moreover, the mining with our proposed algorithm had linear scalability.

In the third experiment, we tested the effect of the distribution of item existential probabilities by varying the average value of existential probabilities. Here, we fixed the database size as 100k records with an average transaction length of 10 items and a domain of 1,000 items and the minsup. The experimental result showed that with the average value of existential probabilities increased, the runtime of our UF-growth algorithm increased.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

The fourth experiment also tested the effect of the distribution of item existential probabilities. This time, we varied the number of unique existential probability values. In this experiment, We fixed the database size as 100k records with an average transaction length of 10 items and a domain of 1,000 items and the minsup. We also fixed the average value of existential probabilities as 0.5. The experimental result showed that when the number of unique existential probability values increased, the runtime of our UF-growth algorithm increased.

In the fifth experiment, we evaluated the relation between the distribution of item existential probabilities and the number of nodes in the UF-tree. Here, we fixed minsup as 0.01 and the average value of existential probabilities as 0.5. We varied the number of unique existential probability values from 1 to 10. The experimental result showed that when the number of unique existential probability values increased, the number of nodes in global UF-tree increased.

In the sixth experiment, we measured the number of nodes in a UF-tree. The result showed that the number of nodes in the global UF-tree representing a database is no more than the number of items in all transactions of such a database. We observed that the number of tree nodes is less than the number of items for all transaction databases.

The experimental results also showed that when the size of transaction database, average value of existential probabilities or number of unique existential probability increased, the runtime for our UF-growth algorithm increased. However, when the value of minsup increased, the runtime for our algorithm decreased. Moreover, when the number of unique existential probability increased, the number of the tree nodes in global UF-tree increased. Nevertheless, the number of tree nodes in global UF-tree was no more than the number of items in the transaction database (which was used to build the global UF-tree) in all situations. For our ACUF-growth, UF-streaming and SUF-growth algorithms, we did the similar experiments and received the similar results such as when the size of transaction database increased, the runtime for all these algorithms increased. Since our ACUF-growth algorithm was a constraint based algorithm, we also evaluated it by selectivity. The result showed that when constraint selectivity increased, the runtime for our ACUF-growth algorithm increased.

V. CONCLUSION

There are many real-life situations in which the data in transaction databases are uncertain. This calls for uncertain frequent itemset mining. In uncertain frequent itemset mining, the user may be only interested in some small specific subsets of all frequent itemsets. This calls for constrained frequent itemset mining of uncertain data. Moreover, due to advances in technology, a flood of uncertain data can be produced in many situations. This leads to frequent itemset mining from streams of uncertain data. To deal with these situations, we proposed in this paper the following four algorithms:

• UF-growth algorithm, which efficiently finds frequent itemsets from uncertain data;

• ACUF-growth algorithm, which mines from uncertain data for those frequent itemsets which satisfy user-specified aggregate constraints;

• UF-streaming algorithm, which is an approximate algorithm designed to find frequent itemsets from streams of uncertain data;

Among them, the UF-growth algorithm constructs a UF-tree to capture the important contents of uncertain data where each item is associated with an existential probability. Each node in the UF-tree not only contains an item and its expected support, but also the number of occurrence of such expected support for such an item. After the UF-tree is constructed, the algorithm recursively mines frequent itemsets from this UF-tree. The ACUF-growth algorithm builds a modified UF-tree in which the order of the tree nodes is based on a specific order R. Here, the order R is closely related to the forms of aggregate constraints handled by the algorithm to make sure the constraints possessing nice properties (e.g., anti-monotonicity property, monotonicity property, convertible anti-monotonicity property, and convertible monotonicity property). These constraint properties helps our ACUF-growth algorithm efficiently mine constrained frequent itemsets from uncertain data without checking every itemset to determine whether or not it satisfies the aggregate constraints. UF-streaming is an approximate algorithm, which applies the UF-growth algorithm with preMinsup (which is lower than actual minsup) to find "frequent" itemsets and uses a tree-structure called UF-stream to store and maintain mined "frequent" itemsets. Moreover, UF-streaming is an "immediate" mode based mining algorithm which starts mining "frequent" itemsets as soon as streams of uncertain data flow in. We evaluated the algorithms which perform uncertain frequent itemset mining (i.e., UF-growth algorithm), constrained frequent itemset mining from uncertain data (i.e., ACUF-growth algorithm) and frequent itemset mining from streams of uncertain data (i.e., UF-streaming and SUF-growth algorithms) by sets of experiments. The experiment results showed the behaviors of our proposed algorithms and illustrated the effectiveness of them.

References

[1] C.C. Aggarwal, Y. Li, J. Wang, and J. Wang, "Frequent pattern mining with uncertain data", In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pages 29–38, Paris, France, 2009.

[2] T. Bernecker, H.P. Kriegel, M. Renz, F. Verhein, and A. Zuefle, "Probabilistic frequent itemset mining in uncertain databases", In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pages 119–128, Paris, France, 2009.

[3] M. Chau, R. Cheng, B. Kao, and J. Ng, "Uncertain data mining: an example in clustering location data", In Proceedings of the 10th Pacific Asia Conference on Knowledge Discovery and Data mining (PAKDD), pages 199–204, Singapore, 2006.

[4] G. Cormode and M. Hadiieleftheriou, " Finding frequent items in data streams", In Proceedings of the 34th International Conference on Very Large Data Bases (VLDB), pages 1530–1541, Auckland, New Zealand, 2008.

[5] C.K. Chui and B. Kao, " A decremental approach for mining frequent itemsets from uncertain data", In Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data mining (PAKDD), pages 64–75, Osaka, Japan, 2008.

[6] D.Y. Chiu, Y.H. Wu, and A.L. Chen, "Efficient frequent sequence mining by a dynamic strategy switching algorithm", The International Journal on Very Large Data Bases (VLDB Journal), 18(1):303–327, 2009.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

[7] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu, " Mining frequent patterns in data streams at multiple time granularities", In Data Mining: Next Generation Challenges and Future Directions. AAAI/MIT Press, 2004.
[8] N. Jiang and L. Gruenwald, "Research issues in data stream association rule mining", ACM SIGMOD Record, 35(1):14–19, 2006.
[9] C.C. Aggarwal, "On density based transforms for uncertain data mining", In Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE), pages 866–875, Istanbul, Turkey, 2007.
[10] H.P. Kriegel and M. Pfeifle, "Density-based clustering of uncertain data", In Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pages 672–677, Chicago, IL, USA, 2005.
[11] C.K. Leung and D.A, "Brajczuk. Efficient algorithms for mining constrained frequent patterns from uncertain data", In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pages 9–18, Paris, France, 2009.
[12] Z. Li, Z. Chen, and Y. Zhou, "Mining blocks correlations to improve storage performance" ACM Transactions on Storage (TOS), 1(2):213–245, 2005.
[13] Mohamed Anis Bach Tobji, Boutheina Ben Yaghlane, and Khaled Mellouli, "A New Algorithm for Mining Frequent Itemsets from Evidential Databases", Proceedings of IPMU'08, pp. 1535{1542 Torremolinos (M_alaga), June 22{27, 2008.
[13] L. Manikonda, A. Mangalampalli1, V. Pudi, "UACI: Uncertain Associative Classi¯er for Object Class Identification in Images", 2010 IEEE.