User-Centric Mining of Association Rules

S K Gupta1, Vasudha Bhatnagar2, and S K Wasan3

1 Department of Computer Science and Engineering, Indian Institute of Technology, Hauz Khas, New Delhi 110 016, India, skg@cse.iitd.ernet.in
2 Department of Computer Science, Motilal Nehru College, University of Delhi, Delhi, India, vasudha@cse.iitd.ernet.in
3 Department of Mathematics, Jamia Millia Islamia, New Delhi, India, sngwasan@yahoo.com

Abstract. Association rule mining is an important mining function because of its completeness. It has been even used to generate classifiers and to segment database. However the current model of association rule mining poses problems regarding flexibility while handling evolving databases. The expensive process of massive rule discovery discourages frequent application of mining algorithm for exploratory mining. We present a methodology to reuse computation and provide an environment that encourages fair and flexible exploration of database through windows. The environment allows multiple data miners to simultaneously explore the database without causing interference either to the database operation or among each other.

Keywords: Evolving databases, Accumulation, Knowledge Concentrate, Operators, Application Development

1 Introduction

Association rule mining has been most widely studied of all data mining functions. Association rule mining though easy to use, but in its current form lacks flexibility for exploration. Minimal interaction with the user during the mining process, slow response, voluminous output etc. prevents end-user to exploit full potential of association rule mining.

In this paper we propose a user-centric approach to association rule mining. Windows of aggregated (concentrated) knowledge in the incremental database are made available to the user for mining. The knowledge aggregated in windows can be reused and shared by multiple users simultaneously using a set of operators, providing flexibility for association rule mining. Functionality to fulfill special needs can be embedded in the applications built over these operators. Simple routine queries can be answered using simple SQL-like interface.

1.1 Motivation

One of the strong motivating factors for working on association rule mining is the centrality of association rule mining to different data mining functions including classification and clustering [2,8–10,15].
Flexibility is the casualty in the current model of association rule mining because of high turnaround time and rigid specification of support and confidence parameters. Repeated mining with different values of parameters or constraints is expensive. It is not economical to provide an environment that permits multiple users to mine independently the desired subsets of database (Fig. 1). The current model of mining association rules also does not take into account the time dimension in a natural way, in the evolving databases. When the entire database is mined, the changes in trends get averaged out and hence are subdued. This is a considerable loss of knowledge as shown in the example in (Fig. 2).

![Fig. 1. User-Centric Mining through Application Development (LITS - Large item-sets)](image1)

![Fig. 2. Importance of Time Dimension in Association Rule Mining](image2)

![Fig. 3. Mining at two levels: User 1, 2 and 3 represent different categories of users.](image3)
1.2 Related Works

We give a brief description of some of the works, which have particularly influenced the work presented in this paper.

User centric mining environment, which forms an important theme of this paper has been inspired by [7]. The concept has been formally developed and implemented in [14]. However the approach pursued in this thesis falls short of the vision in [7]. It limits the exploratory power of the idea, since user can query only on the already generated rule-set. To vary the mining perspective, the user has to undertake the expensive process of massive rule discovery from scratch. At this level, there is no attempt to reuse computation. Further, there is no provision for taking care of the time dimension.

DEMON framework highlights the importance of mining datasets as they evolve with time [3]. The framework can easily accommodate different data mining models (large item-sets, decision trees, clustering etc.). Presently we confine our discussion to large item-sets. DEMON framework has been developed over a data-warehousing environment. This may be a serious limitation for small or medium database users, who may need the DEMON functionality directly from the database, without incurring extra expense for a data warehouse. Secondly, the thrust is on model maintenance only, missing an important requirement of dynamically generating and querying models on user demand.

An important factor that influenced our work is the emergence of new data structures and algorithms for association rule discovery. Traditional algorithms [1] etc., make multiple passes over the database generating candidate sets (for subsequent pruning) in each pass. Lattice based algorithms, make only one scan of database but the structure is expensive to maintain and store [16]. Tree based approaches proposed in [5, 6, 12] support storing counts of all item-sets existing in the database, in a tree structure. We chose Seq algorithm proposed in [12], for implementation of our approach because of reason of simplicity (compared to [5]), single scan (compared to [6]), and fair counting i.e. unbiased by user parameter (compared to [6]). The algorithm is naturally amenable to incremental implementation. We briefly describe this algorithm below.

Seq Algorithm: The algorithm finds large item-sets, from a typical transaction database in a single scan. The algorithm works in two phases. The first phase reads the database transactions (assumed to be lexicographically ordered), and stores all item-sets as sequences. The item-sets are stored in the form of a tree for each item. In the second phase these trees are manipulated in such a manner that for a given support value, they yield all the large item sets of all sizes for the given minimum support. It’s incremental version has been developed in [13].

2 Proposed Methodology

We present methodology for user-centric mining of association rules in evolving databases, by providing DBMS like environment. The users can selectively mine
on aggregated knowledge available in form of windows. The knowledge aggregated in windows is free from any kind of user bias (e.g., minimum support value) and can be used by multiple users simultaneously. Hence it supports flexible experimentation and exploration in mining/monitoring process.

2.1 Accumulation Operation

We introduce a novel operation viz. Accumulation to process the evolving database incrementally. The operation may be performed either periodically (automatically) or on user demand (manually) depending on the rate of growth of the database. During Accumulation, pre-determined (user-directed) functions for selection, cleaning and transformation are applied on the incremental database. Subsequently data aggregation is performed on the pre-mined database. In this way Accumulation operation tightly binds the pre-mining process steps and preliminary data aggregation. ACCUMULATE operator defined in (Sec. 4.1) effectuates Accumulation.

The output of Accumulation operation is a Knowledge Concentrate (described in detail with example in (Sec. 3)). For each increment of the database the Knowledge Concentrate provides a ”window” of aggregated knowledge, which can be mined for association rules. Knowledge Concentrates are stored on the secondary storage and can be viewed as sequenced elements (non-overlapping windows) of database, preserving the time dimension. With in a window, however the time dimension is collapsed. Deciding the granularity of a window is an important issue in Accumulation. The windows can be resized dynamically at the time of mining. From now onwards, we will be using the terms ”window” and Knowledge Concentrate interchangeably.

Knowledge Concentrates can be mined using the set of operators defined in (Sec. 4). The extra space required to store Knowledge Concentrates is an overhead. However, the benefits of this extra cost are tremendous. Knowledge Concentrates for different increments of the database provide the capability to explore and monitor the changing trends in the database. The Knowledge Concentrates extracted after Accumulation can be mined independently with user specific perspective, simultaneously by different independent users (Fig. 1). The Knowledge Concentrates can be stored on the server and the users can mine independently on their respective clients. Thus effective reuse of knowledge aggregated during Accumulation is made for all subsequent mining requirements of the users at different levels (Fig. 3).

2.2 Advantages of Accumulation Operation

Other advantages of Accumulation which help in achieving the objective of User-Centric mining are listed below.

1. Flexible Exploration: Flexibility is provided in three dimensions. It gives choice to the user for i) support and confidence parameters ii) focusing on subsets of database iii) other constraints, in a dynamic way. Repeated mining
of Knowledge Concentrates, varying parameters and constraints becomes cheaper with reduced turn around time.

2. Two Level Exploration: Accumulation of aggregated knowledge allows exploration at two levels - either directly from the database (via Knowledge Concentrate) or through exploration of the knowledge discovered earlier (Fig. 3). Exploring the already discovered knowledge makes near total reuse of computation.

3. Application Development: Functionality for Application Development is provided by defining a set of operators at two levels - primary level and secondary level (Sec. 4). These operators can be combined to develop applications for specific needs (Sec. 5).

Combination of all these ideas contribute in the development of KDDMS (Knowledge Discovery and Data Mining System) as envisioned in [7].

2.3 Other Issues in Accumulation

Depending on the nature of database, the user may choose to Accumulate after a fixed time period $t$, or after a fixed number of transactions $n$. Rate of growth of database, availability of computing resources, objective of knowledge discovery exercise are some of the factors that might influence the periodicity of accumulation. Accumulation may be scheduled at OS level to carry it out automatically. There may not be any interaction with the user during Accumulation in this case. However, it is also possible to keep manual control of the operation if the user desires. The operation may be scheduled at a lean time i.e. when the database activity is minimum.

3 Knowledge Concentrate for Association Rule Mining

A Knowledge Concentrate is a data structure derived from a scan of the incremental database, which can be used for mining without further recourse to database scan. We map the data structure used by the Seq algorithm without any modification [12], to the Knowledge Concentrate. The items in a transaction are assumed to be lexicographically ordered. A transaction $t = \langle i_1, i_2, \ldots, i_l \rangle$ of length $l$ is considered to be sequence of the same length and is represented as a branch of a tree, whose root is the first item of the transaction ($i_1$). The leaf of the branch stores the support count of the sequence. $l-1$ subsequences of lengths ($l-1$, $l-2$, \ldots, 1) are derived from the same transaction, and each subsequence is added as a branch of the respective tree as described above. Thus a single transaction of length $l$ leads to addition of $l$ branches in $l$ different trees (Fig. 4). During processing of a transaction, derivation of sequences and subsequences gives rise to a forest of trees. To give clearer picture of the data structure of the Knowledge Concentrate we use the database shown in (Fig. 2) and show windows for $\delta d_1, \delta d_2, \delta d_3$ in (Fig. 5).
3.1 Properties of Knowledge Concentrate

In general, Knowledge Concentrates (KC) must satisfy certain properties in order to ensure validity, integrity, soundness and completeness of the knowledge discovered from the database. Equivalence property and Additive property of KC have been stated and proved in [4]. The Equivalence property ensures that the knowledge discovered from a database D at any time using windows, should be same as that discovered by a traditional mining algorithm applied to D. The additive property of KC is useful for merging two or more consecutive windows to create a temporary window of desired size. MERGE operator, explained in (Sec. 4.1) implements additive property.

3.2 Space Occupied by Knowledge Concentrate

In [12,13] it has been shown that the ratio of the space occupied by the forest (KC) to that of the database, declines with the growing volume of database. Duplication of transactions, in the database leads to the space saving in the forest where an item-set is stored only once. In our strategy, however this advantage is somewhat reduced. The size of the incremental database is much smaller compared to the size of entire database, and duplicity in storage of sequences (item-sets) that are uniformly distributed results in larger space requirements of KC. The KCs thus occupy more space than the database. This extra storage can be construed as the cost of other benefits that accrue to the Accumulation operation. It is not possible to estimate apriori space requirements for KC. We infer the lower and upper bounds of the space required by KC, by analyzing the best case (when all transactions are identical) and the worst case (when all transactions are distinct) in [4].
3.3 Data Structure and Files

To implement the functionality for window based mining, complex data structures need to be maintained. They are stored as meta-data in a number of different files, to be carried forward to satisfy mining requirements of the users. We briefly describe the important data structure and files maintaining meta-data, for our model of association rule mining.

**Data Structure:** Three important objects, that are of direct interest to the user relate to the large item-sets, rules and constraints of mining. These objects can be defined and used in the application programs.

1. Large Item-set List: The item-sets discovered to be large, for a given support value are stored in an array of linked lists. The item-sets are graded by their sizes.
2. Rules: The rules found to be interesting, can be processed further by the user, using this object. The object actually consists of a collection of rules (up to a maximum limit) and a count of the rules in the collection.
3. Constraint Block: A constraint block is used to convey user defined constraints during discovery process.

**Files:** Following files are maintained which store the KC and the corresponding meta-data.

1. History File: A History file is maintained to record the history of Accumulation performed over the database. Facts related to mining process, along with some performance related parameters are recorded in each history record.
2. For every instance of Accumulation, the KC is stored as collection of files; each time in a new directory identified by a time stamp.
3. Files containing meta-data for KCs are maintained which are used during mining.

4 Mining Operators

Availability of windows (KCs) for mutually exclusive subsets of database, permits simultaneous mining of different or overlapping subsets of database by single or multiple users. Simple mining queries can be answered using query interface and for complex queries applications may be developed. We propose a set of operators for providing this functionality. The operators have been segregated into two classes depending on the level at which they operate. The primary operators work at the level of system aggregated knowledge and discover base knowledge (large item-sets or rule-sets). This base knowledge can be stored/retrieved or possibly processed using secondary operators. Combining these operators meaningfully and embedding desired functionality could develop applications for specific needs. Logical view of the operators is given in (Fig. 6).
4.1 Primary Operators

Primary Operators operate at lower level, processing either system-aggregated knowledge (i.e. KC) or database. The knowledge discovered after processing the KC is available in the main memory. The user can either store this knowledge for exploration/monitoring at later time or use it directly in applications. We propose four primary operators, which are described below with their logical interfaces. The bold parameter in the parameter list is the output.

1. ACCUMULATE: This operator performs the sensitive task of creating windows (KCs) for the incremental database. It carries out Accumulation operation, pre-mining and processing transactions to extract KCs. This operator can be invoked either manually or automatically. The history record in the history file is created each time this operator is invoked. The output i.e. KC and other meta-data are stored in a directory which is time stamped. Accumulate operator is the only primary operator that is not available to general users. Only an authorized personnel can invoke this operator (Chap. 3 of [4]).
   **Interface:** void Accumulate(history-file-name, database-name);

2. MERGE: This operator implements additivity of KC. To perform powerful exploratory data analysis interactively, the window sizes need to be defined dynamically at the time of invoking mining query. MERGE operator provides the capability to dynamically merge windows according to the user requirement. The operator is defined in conjunction with the design of KC. This is the only primary operator, which satisfies closure property.
   **Interface:** void Merge(history-file-name, window-list, output-window);

3. GETLITS: This operator discovers all large item-sets from the given window for a given support value and constraints. The output of the operator, the large item-sets are available in the memory. The discovered knowledge can be either stored as file or further processed by a secondary operator in an

---

**Fig. 6.** Logical View of the Operators for Mining Association Rules
application. The algorithm to discover the large item-sets is basically phase 2 of Seq algorithm [12]. It has been modified to handle user constraints.

**Interface**: void Getlits(input-window, support, constraint-block, large-itemset-list);

4. **GETRULES**: This operator generates all association rules from the (discovered) large item list for the user given metric and constraints. Because of the sheer enormity of numbers, it is not possible to store the rules in memory at any point of time. Therefore by default, an ASCII file of rules is prepared, showing the rules with respective metric value and support count. Alternatively the user can store rules as binary file to be used later for exploration.

**Interface**: void Getrules(large-itemset-list, metric-name, metric-value, constraint-block, file-name);

### 4.2 Secondary Operators

The secondary operators support storage and retrieval of knowledge discovered earlier. Some commonly required processing capabilities are also provided. These operators facilitate reuse and sharing of the discovered knowledge among multiple users. These operators are simple in design, and provide facility to create an environment for application development by the user. We describe these operators and give their user interfaces below.

1. **STORLITS**: Stores the memory resident large item-sets into file(s) graded by their respective sizes.

   **Interface**: void Storlits(large-itemset-list, file-name)

2. **RTRLITS**: Retrieves the large item-sets from secondary storage, and makes them memory resident. The user can selectively retrieve the large item-sets of desired size and explore only a subset of the large item-set.

   **Interface**: void Rtrlits(file-name, max-item-set, large-itemset-list)

3. **STORRULE**: Stores the rules available in memory (as an object) in a binary file for future exploration at a later time.

   **Interface**: void Storrule(rule-object, file-name)

4. **RTRVRULE**: This operator makes a set of rules stored in a binary rule file, available in the memory. All the rules may not be accommodated in the memory and the user can see a block of rules at a time.

   **Interface**: void Rtrvrule(file-name, rule-object)

5. **ASCILITS**: This operator writes the large item-sets in a ASCII file, instead of binary file (in Storlits). An editor or a word processor can view this file.

   **Interface**: void Asciliits(large-itemset-list, file-name)

6. **ASCIRULE**: This operator writes the block of rules as an ASCII file. An editor or a word processor can view this file. This facility may help in preparation of reports.

   **Interface**: void Ascirule(rule-object, file-name)

7. **SRCHLITS**: This operator searches for subsets or supersets of given large item set or for a given item set. The search is made in the memory resident large item-sets.

   **Interface**: void Srchlits(large-itemset-list, constraint-block, result-list)
8. **SRCHRULE**: This operator searches for rules in the given binary file, satisfying the given constraints. (Recall that all the rules will not be available in the memory).
   **Interface**: void Srchrules(file-name, constraint-block, out-rule)

9. **RANKLITS**: This operator sorts the large item list of a given size, according to the given order and returns the desired number of large item-sets to the user as output.
   **Interface**: void Ranklits(lar-item-list, size-of-list, order, number, out-list)

10. **RANKRULE**: This operator sorts the rules in the file, according to the metric value and given order and returns the desired number of rules to the user in an ASCII/binary file
   **Interface**: void Rankrule(rule-file, order, number, out-file)

Operators to display the memory resident rules and large item lists are also available (DISPRULE and DISPLITS respectively).

5  **Application Development**

In this section we show how the proposed operators can be used in queries and application. The grammar for query formulation (IMQL) has been given in detail in [4]. Here we give examples of the common queries along with specific-need applications. The queries are ordered by increasing complexity. Because of paucity of space, the pseudo-codes showing the use of operators have not been included.

1. **Filter the rules stored in file F1, with modified confidence measure c**
   This query can be used to focus with varied metric value. The reuse of discovered knowledge is most prominently display here.

2. **Compare the large item sets in periods Jan95 to Jun95 and Jan98 to Jun98 using support value s, and identify those large item sets which are either new or missing in later window.**
   This query is slightly complex. It may be useful for monitoring the item-sets which have either gained or lost support over two specific time periods.

3. **Mine for rules from Jan96 to Jan97, with support s, confidence c and give minimum, maximum, mean support for rules containing item-set X as consequent**
   This application shows statistical analyses that can be performed for discovered rules. Simple statistical parameters have been used here. However more complex analyses like finding median or mode support for a set of rules can also be found.

6  **Performance Study**

In this section, we briefly discuss the results of experiments performed using the defined operators. We studied two important issues: The extra space occupied
by the KCs and performance of the primary operators. 'Merge' and 'Getlits' are the two primary operators which are crucial to the speed of applications. Note that Accumulate operator, doesn’t ever figure in any user application (Item 1 in Sec. 4.1). Getrules operator has been consciously omitted as we use the standard algorithm for generating rules from large item-sets.

The graph in (Fig. 7) shows that irrespective of the size of the average transaction length, the fall in the measure of the extra space requirement of KC, with the size of the window is similar in nature. Smaller sized window though provide more flexibility in mining, they are more expensive in terms of storage. The tradeoff between the required flexibility and the storage available must be resolved before embarking on the KDD exercise [4]. (Fig. 8) shows that the time require to merge the windows scales linearly with the size of the target window. It is also a function of the number of merged windows.

![Fig. 7. Variation of ratio of storage requirement of KC wrt window size](image1)

![Fig. 8. Behavior of Merge operator wrt number of windows and target window size](image2)

Since "Getlits" is effectively the phase II of Seq algorithm proposed in [12], similar considerations apply. Because of the application of "Getlits" on KCs, some speedup in the mining process is achieved.

7 Conclusion and Future Work

We propose a strategy for mining association rules from an evolving database in a user-centric manner. Accumulation operation is introduced which incrementally constructs windows in database. These windows also called Knowledge Concentrates provide access to the intermediate form of discovered knowledge. Mining is performed by processing these Knowledge Concentrates using a set of operators. Availability of Knowledge Concentrates permits reuse of computation, by multiple users independently and simultaneously. Users can either use a query interface or develop their own applications for specific discovery needs. Due to these strong features it promotes user-centric mining of association rules.

The future work will focus on similar direction for classification and clustering functions of data mining.
Acknowledgements The authors wish to acknowledge the support received from the Ministry of Human Resource Development, New Delhi, INDIA. The research project on "Use of Domain Knowledge in Data Mining". The authors also acknowledge fruitful discussions held with Amuj Khare, Gagan Singla, Shalini Maheshwari, Dr. Naveen Kumar and DVN Somayajulu. Programming help rendered by H. GuruShyam and Sandeep Mehta is gratefully acknowledged. The authors also wish to thankfully acknowledge comments of the anonymous referee, which helped in improving the presentation.

References

6. J. Han, J. Pei, Y. Yin: Mining Frequent Patterns without Candidate Generation. In Proceedings of Int'l Conf SIGMOD 2000, (2000)
11. B. Liu, W. Hsu, Y. Ma: Pruning , Summarizing the Discovered Associations. In Proceedings of Int'l Conf on Knowledge Discovery and Data Mining (KDD-99), (Aug 1999)