STUDY OF ASSOCIATION RULE MINING AND DIFFERENT HIDING TECHNIQUES

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Computer Science and Engineering

By

BIKRAMJIT SAIKIA

DEBKUMAR BHOWMIK

Under the Guidance of

Prof. S.K. JENA

Department of Computer Science Engineering

National Institute of Technology

Rourkela

2009

2009

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the thesis entitled, "Study of various data mining techniques" submitted by Bikramjit Saikia and Debkumar Bhowmik in partial fulfillments for the requirements for the award of Bachelor of Technology Degree in Computer Science Engineering at National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Date:

Prof. S. K. Jena
Dept. of Computer Science Engineering
National Institute of Technology
Rourkela - 769008

ACKNOWLEDGEMENT

We would like to articulate our deep gratitude to our project guide Prof. Sanjay Kumar Jena who has always been our motivation for carrying out the project. We also give our sincere acknowledgement to Prof. K.S Babu, who helped us throughout out endeavor. An assemblage of this nature could never have been attempted without reference to and inspiration from the works of others whose details are mentioned in reference section. We acknowledge out indebtedness to all of them. Last but not the least our sincere thanks to all of the faculty members of out department who have patiently extended all sorts of help for accomplishing this undertaking.

BIKRAMJIT SAIKIA
ROLL NO. : 10506024

DEBKUMAR BHOWMIK
ROLL NO. : 10506026

CONTENTS

List of Tables:

PAGE NO.

List of Figures:

PAGE NO.

ABSTRACT

Data mining is the process of extracting hidden patterns from data. As more data is gathered, with the amount of data doubling every three years, data mining is becoming an increasingly important tool to transform this data into information. In this paper, we first focused on APRIORI algorithm, a popular data mining technique and compared the performances of a linked list based implementation as a basis and a tries-based implementation on it for mining frequent item sequences in a transactional database. We examined the data structure, implementation and algorithmic features mainly focusing on those that also arise in frequent item set mining. This algorithm has given us new capabilities to identify associations in large data sets. But a key problem, and still not sufficiently investigated, is the need to balance the confidentiality of the disclosed data with the legitimate needs of the data users. One rule is characterized as sensitive if its disclosure risk is above a certain privacy threshold. Sometimes, sensitive rules should not be disclosed to the public, since among other things, they may be used for inferring sensitive data, or they may provide business competitors with an advantage. So, next we worked with some association rule hiding algorithms and examined their performances in order to analyze their time complexity and the impact that they have in the original database. We worked on two different side effects – one was the number of new rules generated during the hiding process and the other one was the number of non-sensitive rules lost during the process.

CHAPTER 1

INTRODUCTION

1.1 What is data mining?

Data mining is a technique that helps to extract important data from a large database. It is the process of sorting through large amounts of data and picking out relevant information through the use of certain sophisticated algorithms. As more data is gathered, with the amount of data doubling every three years, data mining is becoming an increasingly important tool to transform this data into information.

Data mining techniques are the result of a long process of research and product development. This evolution began when business data was first stored on computers, continued with improvements in data access, and more recently, generated technologies that allow users to navigate through their data in real time. Data mining takes this evolutionary process beyond retrospective data access and navigation to prospective and proactive information delivery. Data mining is ready for application in the business community because it is supported by three technologies that are now sufficiently mature:

- Massive data collection
- Powerful multiprocessor computers
- Data mining algorithms

1.2 Scope of data mining

Data mining derives its name from the similarities between searching for valuable business information in a large database — for example, finding linked products in gigabytes of store scanner data — and mining a mountain for a vein of valuable ore. Both processes require either shifting through an immense amount of material, or intelligently probing it to find exactly where the value resides. Given databases of sufficient size and quality, data mining technology can generate new business opportunities by providing these capabilities:

- *Automated prediction of trends and behaviors.* Data mining automates the process of finding predictive information in large databases. Questions that traditionally required extensive hands-on analysis can now be answered directly from the data — quickly. A typical example of a predictive problem is targeted marketing. Data mining uses data on past promotional mailings to identify the targets most likely to maximize return on investment in future mailings. Other predictive problems include forecasting bankruptcy and other forms of default, and identifying segments of a population likely to respond similarly to given events.

- *Automated discovery of previously unknown patterns*. Data mining tools sweep through databases and identify previously hidden patterns in one step. An example of pattern discovery is the analysis of retail sales data to identify seemingly unrelated products that are often purchased together. Other pattern discovery problems include detecting fraudulent credit card transactions and identifying anomalous data that could represent data entry keying errors.

Data mining techniques can yield the benefits of automation on existing software and hardware platforms, and can be implemented on new systems as existing platforms are upgraded and new products developed. When data mining tools are implemented on high performance parallel processing systems, they can analyze massive databases in minutes. Faster processing means that users can automatically experiment with more models to understand complex data. High speed makes it practical for users to analyze huge quantities of data. Larger databases, in turn, yield improved predictions.

1.3 Technologies in data mining

According to a recent Gartner HPC Research Note, "With the rapid advance in data capture, transmission and storage, large-systems users will increasingly need to implement new and innovative ways to mine the after-market value of their vast stores of detail data, employing MPP [massively parallel processing] systems to create new sources of business advantage."

The most commonly used techniques in data mining are:

- *Artificial neural networks*: Non-linear predictive models that learn through training and resemble biological neural networks in structure.
- *Decision trees:* Tree-shaped structures that represent sets of decisions. These decisions generate rules for the classification of a dataset. Specific decision tree methods include Classification and Regression Trees (CART) and Chi Square Automatic Interaction Detection (CHAID).
- *Genetic algorithms:* Optimization techniques that use process such as genetic combination, mutation, and natural selection in a design based on the concepts of evolution.

- *Nearest neighbor method*: A technique that classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset. Sometimes called the k-nearest neighbor technique.
- *Rule induction*: The extraction of useful if-then rules from data based on statistical significance.

Apriori is a classic algorithm used in data mining 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 (Winepi and Minepi), or having no timestamps (DNA sequencing).

1.4 Privacy issues in data mining

Providing security to sensitive data against unauthorized access has been a long term goal for the database security research community and for the government statistical agencies. Recent advances in data mining technologies have increased the disclosure risks of sensitive data. Hence, the security issue has become, recently, a much more important area of research. Therefore, in recent years, privacy-preserving data mining has been studied extensively. A number of algorithmic techniques have been designed for privacy-preserving data mining. Most methods use some form of transformation on the data in order to perform the privacy preservation. Typically, such methods reduce the granularity of representation in order to reduce the privacy. This reduction in granularity results in some loss of effectiveness of data management or mining algorithms. This is the natural trade-off between information loss and privacy. Some examples of such techniques are as follows:

- *The randomization method***:** The randomization method is a technique for privacy-preserving data mining in which noise is added to the data in order to mask the attribute values of records. The noise added is sufficiently large so that individual record values cannot be recovered.
- *The k-anonymity model and l-diversity:* The *k*-anonymity model was developed because of the possibility of indirect identification of records from public databases. In the *k*-anonymity method, the granularity of data representation is reduced with the use of techniques such as generalization and suppression.

In the *l*-diversity model, the concept of intra-group diversity of sensitive values is promoted within the anonymization scheme.

- *Distributed privacy preservation:* In many cases, individual entities may wish to derive *aggregate results* from data sets which are partitioned across these entities. Such partitioning may be horizontal (when the records are distributed across multiple entities) or vertical (when the attributes are distributed across multiple entities). While the individual entities may not desire to share their entire data sets, they may consent to limited information sharing with the use of a variety of protocols. The overall effect of such methods is to maintain privacy for each individual entity, while deriving aggregate results over the entire data.
- *Downgrading Application Effectiveness:* In many cases, even though the data may not be available, the output of applications such as association rule mining, classification or query processing may result in violations of privacy. This has lead to research in downgrading the effectiveness of applications by either data or application modifications. Some examples of such techniques include association rule hiding, classifier downgrading, and query auditing.

In our work, we concentrated on *'Association Rule Mining'* technique for mining information from a transactional database and *'Association Rule Hiding'* method for privacy preservation.

We implemented *Apriori* algorithm to generate association rules from a given database and then used two different approaches to hide some of the rules that were considered as sensitive.

CHAPTER 2

ASSOCIATION RULE MINING

2.1 Brief introduction to association rules:

In a database of transactions *D* with a set of n binary attributes (items) *I*, a *rule* is defined as an implication of the form

$$X \Longrightarrow Y \text{ where } X,\ Y \subseteq I \text{ and } X \cap Y = \phi.$$

The sets of items (for short *item sets*) *X* and *Y* are called *antecedent* (left-hand-side or LHS) and *consequent* (right-hand-side or RHS) of the rule respectively. The *support*, *supp(X)*, of an item set *X* is defined as the proportion of transactions in the data set which contain the item set. The *confidence* of a rule is defined

$$conf(X \Longrightarrow Y) = supp(X \cup Y) / supp(X).$$

Following the original definition given by *Agrawal* et al [5]**,** *association rules (ARs)* are implication rules that inform the user about items most likely to occur in some transactions of a database. They are advantageous to use because they are simple, intuitive and do not make assumptions of any models. Their mining requires satisfying a user-specified *minimum support* and a user-specified *minimum confidence* from a given database at the same time. To achieve this, association rule generation is a two-step process.
First, minimum support is applied to find all frequent item-sets in a database.
In a second step, these frequent item-sets and the minimum confidence constraint are used to form rules. While the second step is straight forward, the first step needs more attention.

2.2 Steps in finding the association rules

Suppose one of the large item-sets is L_k, $L_k = \{I_1, I_2, \dots, I_k\}$, association rules with this item-set are generated in the following way: the first rule is $\{I_1, I_2, \dots, I_{k-1}\} \Rightarrow \{I_k\}$, by checking the confidence this rule can be determined as interesting or not. Then other rule are generated by deleting the last items in the antecedent and inserting it to the consequent, further the confidences of the new rules are checked to determine their usefulness. Those processes iterated until the antecedent becomes empty. Since the second sub-problem is quite straight forward, most of the researches focus on the first sub-problem.
In the contest for *Frequent Item-set Mining* Implementation, *Apriori* has proved to be one of the most versatile and successful algorithms ranking next only to more sophisticated algorithms like

eclat, nonordfp and lcm. It has been proved by scholars that *Apriori* outperforms some of these algorithms in areas like space and database content.

2.3 Defining the problem:

PROBLEM: A transactional database consists of sequence of transaction: $T=<t_1,t_2,\ldots.., t_n>$. t_A transaction is a set of items (t_i I). A set of items is often called item-set. The (absolute support or the occurrence of X is the number of transactions that are supersets of X (i.e. that contain X). The relative support is the absolute support divided by the number of transactions (i.e. n). An item-set is frequent if its support is greater or equal than a threshold value.

In the frequent item-set mining problem a transaction database and a relative support threshold is given and we have to find all frequent item-sets.

And eventually in the second step, we find the association rules with the given minimum confidence.

2.4 Apriori Algorithm:

Apriori[6] was proposed by Agrawal and Srikant in 1994. The algorithm finds the frequent set L in the database D. It makes use of the downward closure property. The algorithm is a bottom-up search, moving upward level-wise in the lattice. However, before reading the database at every level, it prunes many of the sets which are unlikely to be frequent sets, thus saving any extra efforts.

Candidate Generation: Given the set of all frequent (k-1) item-sets, we want to generate a superset of the set of all frequent k-item-sets. The intuition behind the apriori candidate generation procedure is that if an item-set X has minimum support, so do all subsets of X. after all the (l+1)- candidate sequences have been generated, a new scan of the transactions is started (they are read one-by-one) and the support of these new candidates is determined.

Pruning: The pruning step eliminates the extensions of (k-1) item-sets which are not found to be frequent, from being considered for counting support. For each transaction t, the algorithm checks which candidates are contained in t and after the last transaction are processed; those with support less than the minimum support are discarded.

```
Pass 1
    1. Generate the candidate item-sets in $C_1$
    2. Save the frequent item-sets in $L_1$
Pass k
        1. Generate the candidate item-sets in $C_k$ from the frequent
           item-sets in $L_{k-1}$
           a. Join $L_{k-1}$ p with $L_{k-1}$q, as follows:
              insert into $C_k$
              select p.item1, $q.item_1, \ldots, p.item_{k-1}, q.item_{k-1}$
              from $L_{k-1}$ p, $L_{k-1}$q
              where $p.item_1 = q.item_1, \ldots p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$
           b. Generate all (k-1) subsets from the candidate item-sets in $C_k$
           c. Prune all candidate item-sets from $C_k$ where some (k-1) subset of the candidate
              item-set is not in the frequent item-set $L_{k-1}$
    2. Scan the transaction database to determine the support for each candidate item-set in $C_k$
    3. Save the frequent item-sets in $L_k$
```

2.5 Implementing Apriori algorithm

The choice of the data-structure to store the candidates is the determining factor in calculating the efficiency of the algorithm. We present an analysis of our implementations using simple *Linked-lists* and *Tries* separately.

2.5.1 Linked List representation*:*

Here we use two kinds of structures—node and an item-set.

```
struct node{
  int index ;
  struct item_set *item,*temp1;
  struct node *next;
};
struct item_set{
  int *data;
  struct item_set *next1;  };
```

There are two separate linked lists----one composed of nodes, where each node specifies the index, i.e., k. Each node will again have a second linked list composed of item-sets associated with it. This connected second linked list will contain the actual items from the transactional database.

The choice of the data-structure to store the candidates is the determining factor in calculating the efficiency of the algorithm. We present an analysis of our implementations using simple linked-lists and tries separately.

2.5.2 The Tries Implementation of Apriori:

The *Tries* [7] data structure used here is similar to the one proposed by *Ferenc Bodon*[8].

A tries is a rooted and labeled tree. Though tries are generally used to store words, they are also useful in storing and retrieving any finite ordered sets. We utilize this property in building the tries for Apriori. The tries contains an item set if there exists a path where the nodes are labeled by the elements of the set, in increasing order. A candidate k-item set C= $\{i_1<i_2<\ldots <i_n\}$ is represented by the nodes i_1, $i_2\ldots\ldots i_n$ in order.

```
struct node{
  short int item, depth, sup;
  struct child *dp;
};
struct child{
   struct node *next;
   struct child *nextc;
};
```

Support counting is done by reading transactions one-by-one and determining which candidates are contained in the actual transaction. We maintain two indices to count support, one for the items in the transaction and the other for the nodes, each being initialized to the first element. After that, we check if the elements pointed by the two indices are equal. If true, we call the process recursively, otherwise we increase the index that points to the smaller item. These steps are repeated until the end of the transaction or the last edge is reached. The support counters of the leaves in the path are increased if the nodes in one path are similar to the particular transaction.

The time of finding supported candidates in a transaction can be reduced significantly by storing some extra information at the nodes. While counting support, we often have to make superfluous moves in tries search in the sense that there are no candidates in the direction we are about to explore. To avoid this superfluous traveling, at every node, we store the length of the longest directed path that starts from there. When searching for k-item set candidates at depth d, we move downward only if the maximal path length at this node is at least k-d.

The most important function that we use is probably the intersection pruning method suggested by *Bodon* [8] with a slight modification of our own. We denote by u the parent of the node that has to be extended (say **u′**). Suppose that the node u is at a depth l, to determine the children of **u′**, we first consider the subsequences at depth *l*, compare with **u′** and one-by-one denote these nodes at depth *l* by *v*. The intersection of the children of *u* and each and every *v* is appended to **u′**. Thus the pruned candidate is obtained in a simple go, saving a lot of superfluous traversals and hence computation time.

This technique of pruning could however not be applied to 1-itemset and 2-itemset candidate generation and we had to write separate functions for them. Some other issues had to be taken special care of such as the adjustment of the depth of the nodes after deletion is performed. Our implementation differed from that of *Bodon* only slightly in that *Bodon* had computed the children (**v′**) of *v* having the same label as that of the node that had to be extended while we did not consider any **v′**, reason being, we did not label our edges as the values or the items of the item sets. We used nodes only for that purpose.

CHAPTER 3

ASSOCIATION RULE HIDING

3.1 Problem Formulation

Providing solutions to database security problems require combining several techniques and mechanisms. In an environment where data have different sensitivity levels, this data may be classified at different levels, and made available only to those subjects with an appropriate clearance. It is however; well known that simply by restricting access to sensitive data does not ensure complete sensitive data protection. For example, sensitive or “high” data items may be inferred from non-sensitive, or “low” data through some inference process based on some knowledge of the semantics of the application the user has. Such a problem is known as ‘Inference Problem’. Association rules can be included in this category. The proposed solutions address the problem of how to prevent disclosure of sensitive data through the combination of known inference rules with non-sensitive data. Below we provide a notational view to the problem.

Let $I = \{i_{1,\ldots}, i_n\}$ be a set of literals, called items. Let *D* be a set of transactions which is the database that is going to be disclosed. Each transaction $t \in D$ is an item set such that *t* is a proper subset of *I*. A unique identifier, which we call it TID, is associated with each transaction. We say that a transaction *t* supports *X*, a set of items in *I* , if *X* is a proper subset of *t*. We assume that the items in a transaction or an item set are sorted in lexicographic order. An item set *X* has support *s* if *s*% of the transactions support *X*. Support of *X* is denoted as *Supp*(*X*).

An association rule is an implication of the form

$$X \Rightarrow Y, \text{ where } X \text{ and } Y \text{ are subsets of } I \text{ and } X \cap Y = \emptyset.$$

We say that the rule X =>Y holds in the database *D* with *confidence C* if

$$\frac{|XUY| * 100}{|X|} \geq C$$

(where **| A |** is the number of occurrences of the set of items *A* in the set of transactions *D*, and *A* occurs in a transaction *t*, if and only if *A* is a proper subset of *t*.).

We also say that the rule *X* => *Y* has *support S* if

$$\frac{|XUY| * 100}{N} \geq S$$

Where, *N* is the number of transactions in *D*.

Here, the *support* is a measure of the frequency of a rule, whereas, the *confidence* is a measure of the strength of the relation between sets of items.

Association rule mining algorithms scan the database of transactions and calculate the support and confidence of the candidate rules to determine if they are significant or not. A rule is significant if its support and confidence is higher than the user specified minimum support and minimum confidence threshold. In this way, algorithms do not retrieve all the association rules that may be derivable from a database, but only a very small subset that satisfies the minimum support and minimum confidence requirements set by the users. We aimed at preventing some of these rules that we referred to as "sensitive rules", from being disclosed. The problem can be stated as follows:

> *"Given a database D, a set R of relevant rules that were mined from D and a subset R_H of R, we had to transform D into a database D' in such a way that the rules in R could still be mined, except for the rules in $R_{H.}$"*

Thus, we were looking for a transformation of *D* (the source database) in *D'* (the released database) that would maximize the number of rules in R **-** R_H that could still be mined.

3.2 Different Approaches To Solve The Problem

There are two main approaches that could be adopted when we tried to hide a set R_H of rules (i.e., prevent them from being discovered by association rule mining algorithms):

(a) We could either prevent the rules in R_H from being generated, by hiding the frequent sets from which they are derived,

(b) We could reduce the confidence of the sensitive rules, by bringing it below a user-specified threshold **(***min_conf*).

We focused our work on the second approach. In order to achieve our goal, transactions were modified by removing some items, or inserting new items depending on the hiding strategy. The constraint on the algorithms was that the changes in the database introduced by the hiding process should be limited, in such a way that the information loss incurred by the process was minimal. Selection of the items in a rule to be hidden and the selection of the transactions that would be modified was a crucial factor for achieving the minimal information loss constraint. Before presenting the strategies and the algorithms, we introduce some notation below.

3.2.1 Notation and Preliminary Definitions

We used a bitmap notation with a few extensions to represent a database of transactions. Bitmap notation is commonly used in the association rule mining context. In this representation, each transaction *t* in the database *D* is a triple:

$$t =< TID; \text{values of items}; \text{size} >,$$

where, *TID* is the identifier of the transaction *t* and *values of items* is a list of values with one value for each item in the list of items *I* and *size* is the size of the transactions, that is, the number of items in the transaction t. An item is represented by one of the initial capital letters of the English alphabet. An item is supported by a transaction *t* if its value in the *values of items* is 1 and it is not supported by *t* if its value in *values of items* is 0. *Size* is the number of 1 values which appear in the *values of items* (e.g., the number of items supported by transaction *t*).

Given a set *P*, the conventional representation | *P* | indicates the number of elements of the set *P*. According to this notation, the number of transactions stored in a database D is indicated as | *D* |, while | *I* | represents the number of the different items appearing in D. The set of rules that can be mined from the database is indicated by *R* and the subset of these rules, that we're interested in hiding, is referred to as R_H. For each rule *r* in R_H we use the compact notation l_r to indicate the item set which appears in the left side of a rule *r* (also referred to as rule antecedent) and r_r to indicate the item set which appears in the right side of a rule (also referred to as rule consequent).

Next we present some useful definitions that will help to better understand the algorithms or strategies.

Given a transaction *t* and an item set *S*, we say that *t fully supports S* if the values of the items of *S* in *t.values of items* are all 1; *t* is said to *partially support S* if the values of the items of *S* in *t.values of items* are not all 1's. For example, if S = {A, B, C, D} = [11110] and $p =< T1;$ [10100]; 2 >, $q =< T2;$ [11110]; 4 > then we would say that *q* fully supports *S* while *p* partially supports *S*.

A rule *r* corresponds to an item set. This item set is the union of the items in the left hand side and the right hand side of the rule. We denote the item set that corresponds to rule *r* as I_r, and we refer to it as the *generating item set* of *r*. Two different rules may have the same generating item set.

We use the notation *Tr* to denote the set of transactions that fully support the generating item set of a rule *r*. We also denote by Tl_r the set of transactions that fully support the left hand side or the antecedent of the rule *r*, while by Tr_r we denote the set of transactions that fully support the right hand side of the rule *r*. We slightly change the notations to represent a set of transactions that partially supports an item set. In the previous notations we add the prime symbol in all occurrences of *T*, so the set of transactions that partially support the antecedent of the rule becomes Tl_r' and the set of transactions that partially support the consequent of the rule becomes Tr_r'.

Further, we assume that each rule is assigned to a *sensitivity level*. The sensitivity level is determined based on the impact that a certain rule has in the environment that the rule is a part of. For example, in a retail environment, a rule that can be used to boost the sale of a set of items could be a sensitive rule. The impact of a rule in the retail environment is the degree at which the rule increases the sales and consequently the profit. Since only frequent and strong rules could be extracted by the data mining algorithms we assume that the sensitivity level of only the frequent and strong rules are of interest to us. If a strong and frequent rule is above certain sensitivity level, the hiding process should be applied in such a way that either the frequency or the strength of the rule will be reduced to bring the support and the confidence of the rule below the *min_supp* and the *min_conf* correspondingly.

3.2.2 Hiding Strategies

The hiding strategies heavily depend on finding transactions that fully or partially support the generating item sets of a rule. Because if we want to hide a rule, we need to change the support of some part of the rule, that is, we have to decrease the support of the generating item set. Again, as mentioned in the previous section, the changes in the database introduced by the hiding process should be limited, in such a way that the information loss incurred by the process is minimal. So, we try to apply minimal changes in the database at every step of the hiding algorithms that we propose.

The decrease in the support of an item set *S* can be done by selecting a transaction *t*, that supports *S* and by setting to 0 at least one of the non-zero values of *t.values of items* that represent items in *S*.

The increase in the support of an item set *S* can be accomplished by selecting a transaction *t* that partially supports it and setting to 1 the values of all the items of *S* in *t.values of items*.

If we analyze the formulas for determining support and confidence values (mentioned in previous section), we can find that there can be two ways to reduce the support and confidence of a rule. Both the confidence and the support are expressed as ratios of supports of item sets that support the two parts of a rule or its generating item set. In this way, if we want to lower the value of a ratio, we can adopt either one of the following options:

(a) we can decrease the numerator, while keeping the denominator fixed, or

(b) we can increase the denominator while keeping the numerator fixed.

Considering the case of decreasing support value, we know that support S of a rule X => Y is given by

$$\frac{|XUY| * 100}{N} \geq S$$

Since N is constant (as it is the no. of transactions in the given database), the only option left for us is to change the numerator value (option **(a)**). That means, we can decrease the support of any rule by decreasing the support of the generating item set of the rule.

Considering the case of decreasing confidence value, we know that confidence C of a rule X => Y is given by

$$\frac{|XUY| * 100}{|X|} \geq C$$

Now, we analyze each of the options separately to check which of them (or both) works in the current context.

Option **(a)** implies that we need to decrease the numerator (which is the support) of the generating item set of the rule, while the support of the item set in the left hand side of the rule remains fixed. In order to do that, we can decrease the support of the generating item set of the entire rule by modifying the transactions that support this item set, making sure that we hide items from the consequent or the right hand side of the rule. This will decrease the support of the generating item set of the rule, while it will leave unchanged the support of the left hand side or else the denominator.

Option **(b)** implies that we need to increase the denominator (which is the support of the item set in the antecedent) of the rule, while the support of the generating item set of the rule remains fixed. This option is also applicable, since we can increase the support of the rule antecedent while keeping the support of the generating item set fixed by modifying the transactions that partially support the item set in the antecedent of the rule but do not fully support the item set in the consequent.

So, briefly we can state the strategies as follows:

Given a rule X => Y on a database *D,* the support of the rule in *D* expresses the probability to find transactions containing all the items in X U Y. The confidence of X => Y is, instead, the probability to find transactions containing all the items in X U Y once we know that they contain *X*.

1. We decrease the confidence of the rule:

    (a) By increasing the support of the rule antecedent *X* through transactions that partially support it.

    (b) By decreasing the support of the rule consequent *Y* in transactions that support both *X* and *Y* .

2. We decrease the support of the rule:

    By decreasing the support of either the rule antecedent *X*, or the rule consequent *Y*, through transactions that fully support the rule.

3.2.3 Assumptions

We make the following assumptions in the development of the algorithms:

1. We hide only rules that are supported by disjoint large item sets.
2. We hide association rules by decreasing either their support or their confidence.
3. We select to decrease either the support or the confidence based on the side effects on the information that is not sensitive.
4. We hide one rule at a time.
5. We decrease either the support or the confidence, one unit at a time.

If we try to hide overlapping rules, then hiding a rule may have side effects on the other rules to be hidden. This may increase the time complexity of the algorithms since hiding a rule may cause an already hidden rule to haunt back. That is why, the first assumption is there.

According to the second assumption, we can choose to hide a rule by changing either its confidence or its support, but not both. Association rules with confidence above the confidence threshold are the significant ones. When we are hiding a rule by decreasing its support below the threshold, we do not need to further reduce its confidence. Also after decreasing the confidence of a rule, we do not need to decrease is confidence since it is no longer significant. So reducing the support or the confidence is enough for hiding a rule. Therefore we assume that either the support or the confidence reduction algorithms are used.

The third assumption is actually the main constraint of the algorithms that aims to maximize the data quality in terms of non-sensitive information. If we relax this assumption, we can just randomly choose a transaction and an item to hide from the database without the need for any kind of heuristic approach. But in that case, a large number of new rules can be generated, or many hidden rules may no longer be hidden or some non-sensitive rules may be hidden, causing loss of information.

The fourth assumption states that hiding one rule must be considered as an atomic operation. But the first assumption, in fact, fulfills the requirement specified by the fourth one. Because, if rules considered for hiding are disjoint, then items appearing in different rules will also be different and hence there will be no problem even if we consider multiple rules at a time. But relaxing the first one will increase the necessity of the fourth one.

The method works step by step, considering an item and a transaction at each step which is stated by the fifth assumption. If we relax this assumption, we can assume that we remove a set of transactions and remove items from the whole set which is an entirely different approach.

3.2.4 Algorithms

We have implemented one algorithm for each of the proposed strategies. Below we mention these two algorithms.

3.2.4.1 Algorithm 1:

This algorithm hides the sensitive rules according to the 1st strategy. The basic idea behind it is to increase the denominator in the expression for confidence as mentioned in previous section, while keeping the numerator constant. To do this, first we need to find out all those transactions that partially support both the antecedent and the consequent of the sensitive rule. Then, for each transaction, increase the support of the antecedent so that the transaction now fully supports the antecedent, but still partially supports the rule. This process is repeated until the confidence of the rule goes below the threshold value, so that it cannot be mined any longer. Thus each of the sensitive rules is hidden.

Following is the pseudo code for the above algorithm:

INPUT: a set R_H of rules to hide, the source database D, the number |D| of transactions in D, the min _conf threshold and the min supp _threshold.

OUTPUT: the database D transformed so that rules in R_H cannot be mined.

```
Begin
    For each rule r in $R_H$ do
    {
        1. $T_{lr}$'= { t Є D | t partially supports $l_r$ and $r_r$ }
        // count how many items of $l_r$ are in each transaction of $T_{lr}$'.
        2. for each transaction t in $T_{lr}$' do
            {
                3. t.num items= | I | - Hamming dist( $l_r$, t.values of items)
            }
        // sort transactions of $T_{lr}$' in descending order of number of items of $l_r$
```

```
            // contained
            4. sort ($T_{lr}$')
            5. N _iterations = [| D | * (( supp(r) / min_conf) - supp($l_r$))]
            6. For i = 1 to N _iterations do
            {
                // pick the transaction of $T_{lr}$' with the highest number of items
                7. t = $T_{lr}$' [1]
                // set to one all the bits of t that represent items in $l_r$
                8. set _all_ ones (t.values _of_ items, $l_r$)
                9. supp ($l_r$) = supp ($l_r$) +1
                10. conf(r) = supp(r) / supp($l_r$)
                11. $T_{lr}$' = $T_{lr}$'- t
            }
            12. $R_H$ = $R_H$ - r
    }
End
```

3.2.4.2 Algorithm 2:

This algorithm hides sensitive rules by decreasing the frequency of the consequent until either the confidence or the support of the rule is below the threshold. It first finds out those transactions that support the sensitive rule fully. Then it decreases, for each transaction, the support of the consequent, while keeping the support of the antecedent constant. This process is repeated for all sensitive rules.

Following is the pseudo code for the above algorithm:

INPUT: a set R_H of rules to hide, the source database D, the number |D| of transactions in D, the min _conf threshold and the min supp _threshold.

OUTPUT: the database D transformed so that rules in R_H cannot be mined.

```
Begin
    For each rule r in $R_H$ do
        {
            1. $T_r$= {t Є D | t fully supports r}
            // count how many items are in each transaction of $T_r$.
            2. for each transaction t in $T_r$ do
            {
                3. t.num items= count (t)
            }
            // sort transactions of $T_r$ in ascending order of size of the transactions
            4. sort ($T_r$)
            5. N _iter_conf = [| D | * (( supp(r) / min_conf) - supp($l_r$))]
            6. N_iter_supp = [| D | * ( supp(r) / min_supp)]
            7. N_iterations = min (N_iter_conf, N_iter_supp)
            8. For i = 1 to N _iterations do
            {
                // pick the transaction of $T_r$ with the lowest number of items
                9. t = $T_r$ [1]
                // choose the item of $r_r$ with the minimum impact on the (|r| - 1)
                // item-sets
                10. j = choose_item( $r_r$ )
                 // set to zero the bit oft.calues_of_items that represents item j
                11. set_to_zero ( j, t.values_of_items)
                12. supp (r) = supp (r) - 1
                13. conf(r) = supp(r) / supp($l_r$)
                14. $T_r$ = $T_r$- t
            }
            15. $R_H = R_H - r$
        }
End
```

CHAPTER 4

IMPLEMENTATION RESULTS

4.1 Introduction

In this chapter, we present the results we obtained by implementing the algorithms, that have been mentioned in earlier chapters. First we implemented Apriori algorithm to mine significant association rules from a given database. This algorithm first finds out the frequent item sets that have support and confidence values above a pre-specified threshold value. Then it derives the association rules.

After derivation of association rules, we considered some of the rules, having higher confidence values, as sensitive and implemented the two hiding algorithms on those rules. Number of new rules generated and number of non-sensitive rules lost during the hiding process, were considered as the side-effects of these two algorithms and the algorithms were compared against their side-effects.

To compare the side-effects, each time we compared the new database with the source database rule by rule. If any non-sensitive rule were found to be missing from the new database, it was considered to be lost. Thus we got the number of lost rules. Then we calculated the number of new rules generated by computing the difference between the rules generated by the resultant database and that of the original database.

The algorithms were executed on a workstation with AMD Athlon^{TM} 64*2 Dual Core Processor, 2.81 GHz and 2.00 GB of RAM on Fedora OS.

4.2 Data Mining: Apriori algorithm

We have generated some binary transaction data to test our algorithm, i.e. in a transaction; an item is either present or not present. All implementations were tested on different numbers of transactions, data items and minimum support values. A complete account of the results would require too much space, thus only the most typical ones are shown below.

4.2.1 Linked-List based implementation:

We implemented Apriori algorithm first using linked list data structure. The results, we obtained, are shown in the table below:

Database Size	**Support**	**Memory used (in bytes)**	**Time (in sec.)**
18000	50	16496	0.05495
20000	50	16238	0.05495
50000	50	19082	0.10989
80000	50	22110	0.16484

Table 4.1: Memory and time requirement of linked list based implementation

Next we have plotted the graphs between memory used and size of the database, time and size of the database. Here are the graphs:

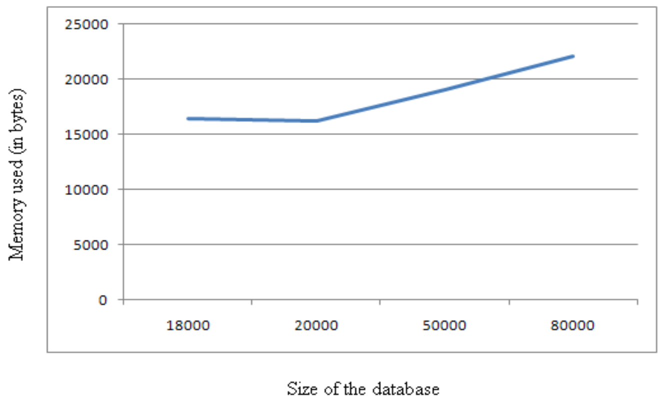

Figure 4.5 : Memory vs. Size of database

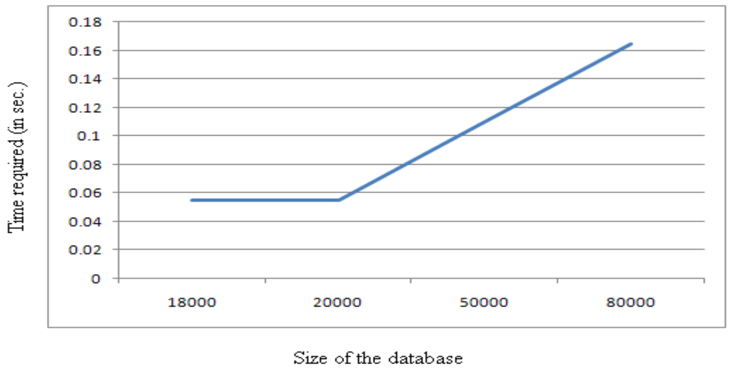

Figure 4.6: Time vs. Size of database

The following observations are made from the above table and graphs:

- This implementation is good only for a small database.
- As the database size increases, the implementation fails to give desired results.
- The above observations made us to switch to the Tries implementation pretty soon.

4.2.2 Tries Based Implementation:

A better implementation then the Linked-List, it helped us to obtained more test results. We implemented Apriori algorithm using tries data structure. The results, we obtained, are shown in the table below:

Database Size	**Support**	**Memory used (in kilo bytes)**	**Time (in sec.)**
10000	7	42	0.01
12000	7	89	0.01
15000	7	221	0.02
20000	7	713	0.07
30000	7	3703	0.37
50000	7	29266	4
70000	7	113700	25
100000	7	477745	170

Table 4.2: Memory and time requirement of tries based implementation

Next we have plotted the graphs between memory used and size of the database, time and size of the database. Here are the graphs:

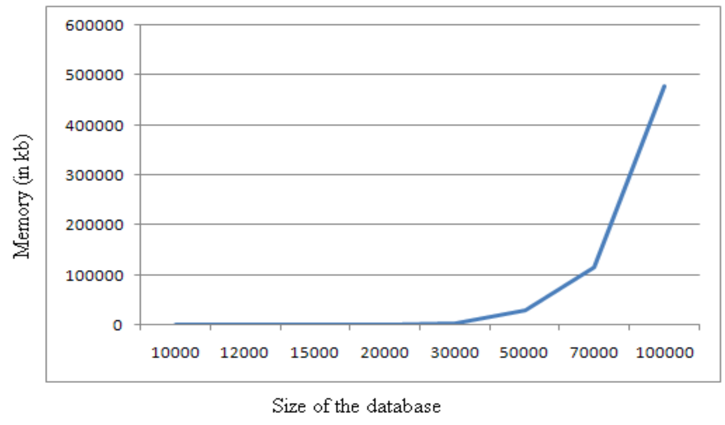

Figure 4.7: Memory vs. Size of the database

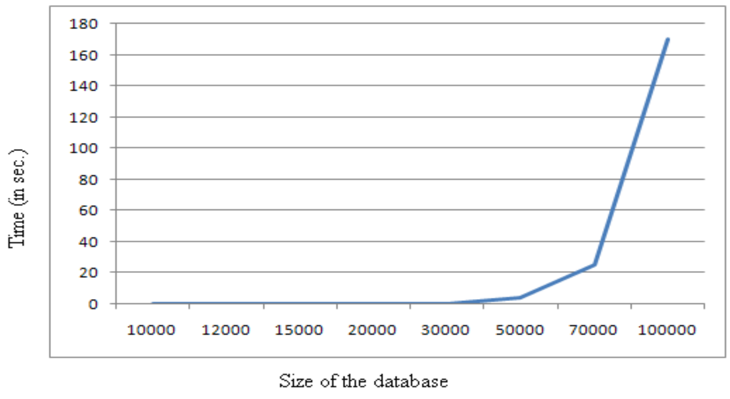

Figure 8.4: Time vs. Size of the database

The following observations are made from the above table and graphs:

- For small database the Linked-List implementation was better. The size and depth of the tries can be attributed to this observation.
- Memory uses is much better in the Tries implementation.
- The memory increases exponentially with increasing database because the depth of the Tries increases exponentially.
- The time required also follow the same trend as the memory as with increase with depth the search space is also increasing.

4.3 Performance Evaluation of Association Rule Hiding Algorithms

After getting the association rule hiding algorithms work correctly for a small sample database (discussed in previous section), we executed the algorithms for increasing number of transactions with varying number of rules (to be hidden) to check their performance and also to have an idea about their relative performances.

We compared the two algorithms on three different measures. Those are mentioned below:

(a) Time requirement

(b) Number of new rules generated

(c) Number of rules lost

4.3.1 Time Requirement

We executed both algorithms for 2 rules and 5 rules (to be hidden) and for each of those two cases; we increased the number of transactions starting from 10000 to 35000 with an increment of 5000 transactions at each step. These databases of different sizes were generated using **Apriori** Algorithm**.**

In the following tables, we have provided the results we obtained from our experiments. Then, we have plotted graphs using these statistics to get a comparative result among the two.

Here are the tables:

Table 4.3: Time requirement of Algorithm 3 (No. of rules to be hidden = 2)

No. of transactions	**Time required for execution (in sec.)**
10000	0.659341
15000	1.318681
20000	2.582418
25000	4.67033
30000	6.978022
35000	8.351648

Table 4.4: Time requirement of Algorithm 4 (No. of rules to be hidden = 5)

No. of transactions	**Time required for execution (in sec.)**
10000	0.879121
15000	1.703297
20000	3.681319
25000	6.868132
30000	10.16484
35000	12.03297

Table 4.5: Time requirement of Algorithm 2 (No. of rules to be hidden = 2)

No. of transactions	**Time required for execution (in sec.)**
10000	0.10989
15000	0.274725
20000	0.43956
25000	0.604396
30000	0.769231
35000	0.879121

Table 4.6: Time requirement of Algorithm 2 (No. of rules to be hidden = 5)

No. of transactions	**Time required for execution (in sec.)**
10000	0.274725
15000	0.494505
20000	0.714286
25000	0.989011
30000	1.373626
35000	1.593407

Next we have plotted the graphs for the time requirements of two algorithms against number of transactions in the database. The graphs are shown below:

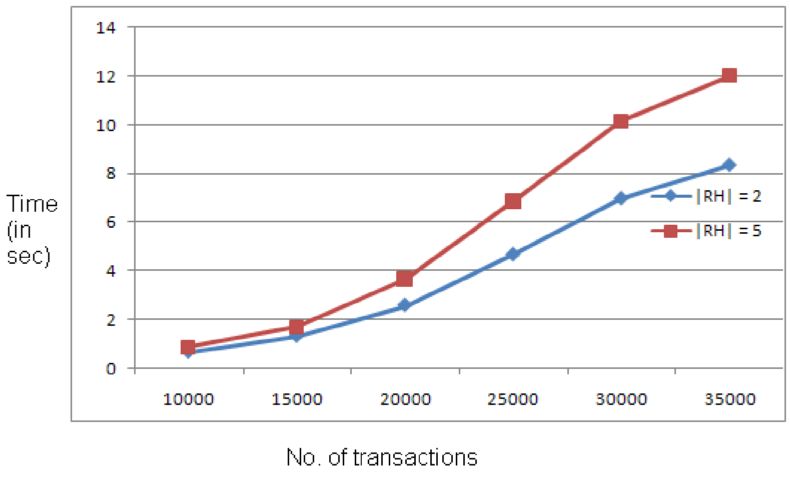

Figure 4.5: Time requirement for Algorithm 2

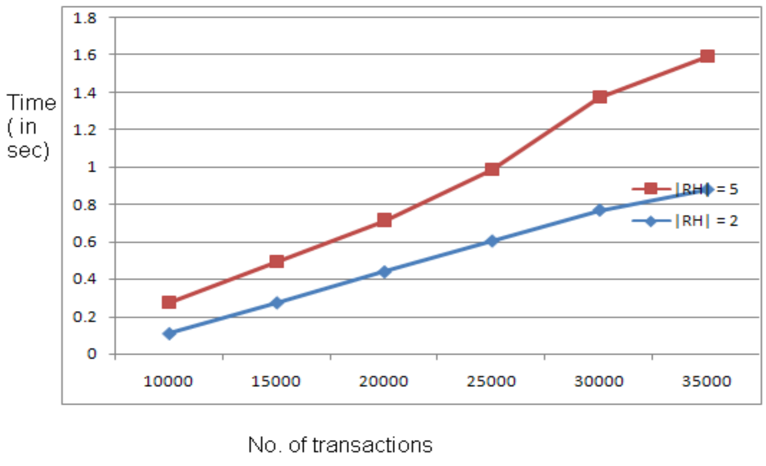

Figure 4.6: Time requirement for Algorithm 2

From the above tables and the graphs, we can infer that:

- Algorithm 2 works far better than Algorithm 1 in respect of time required for execution.

- In case of both the algorithms, as number of transactions increases, required time also increases.
- Similarly required time also increases with the increase of number of rules to be hidden (R_H).
- In case of Algorithm 1, at each step support of the antecedent gets increased, which causes the database to grow up in size gradually. On the other hand, in case of Algorithm 2, after each iteration, the support of the rule gets decreased. As a result, the database shrinks. Hence, we got the above result.

4.3.2 New Rules Generated

In the following tables, we have provided the results we obtained from our experiments.

Table 4.7: New rules generated for Algorithm 1

No. of transactions	**No. of new rules generated(R_H =2)**	**No. of new rules generated(R_H =5)**
5000	94	754
10000	78	641
15000	59	468
20000	48	278
25000	35	165
30000	35	128
35000	35	98

Table 4.8: New rules generated for Algorithm 2

No. of transactions	**No. of new rules generated(R_H =2)**	**No. of new rules generated(R_H =5)**
5000	3	6
10000	3	5
15000	3	4
20000	2	4
25000	2	4
30000	1	2
35000	1	1

Next we have plotted the graphs for the no. of new rules generated against number of transactions in the database for both the algorithms. The graphs are shown below:

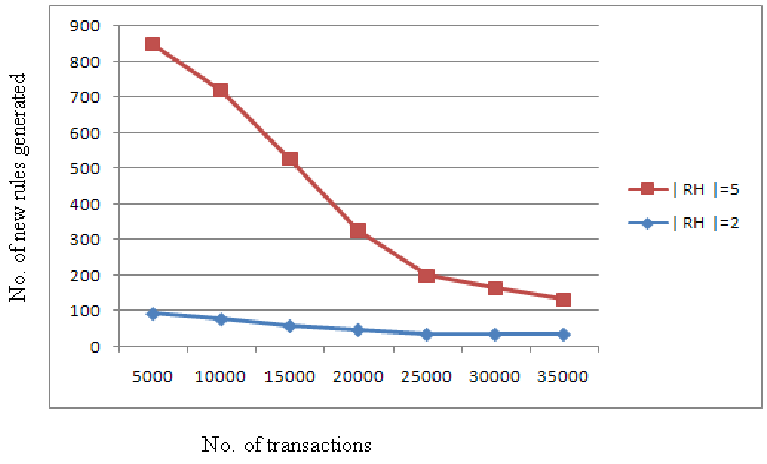

Figure 4.7:No. of new rules generated vs. No. of transactions for Algorithm 1

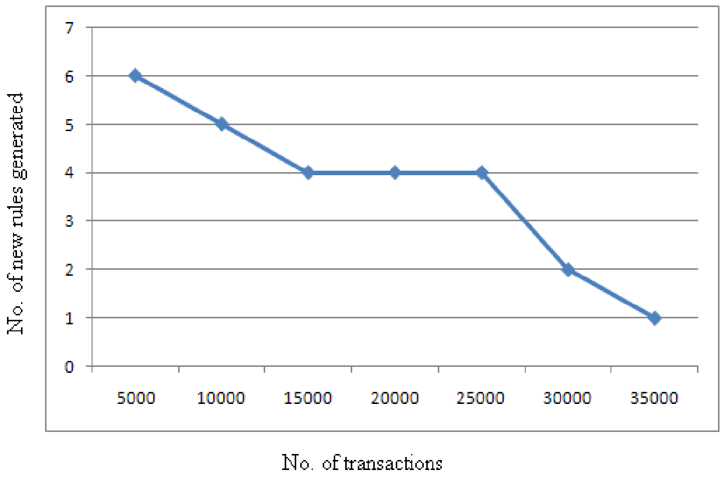

Figure 4.8: No. of new rules generated vs. No. of transactions for Algorithm 2 for 5 rules

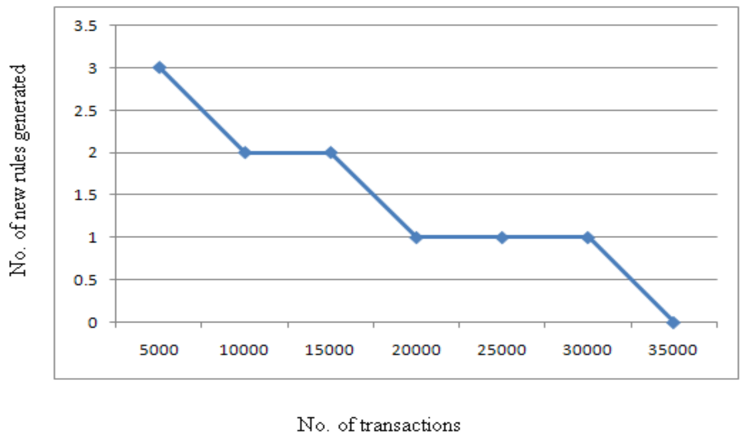

Figure 4.9:No. of new rules generated vs. No. of transactions for Algorithm 2 for 2 rules

From above graphs and tables we make the following conclusions:

4.3.2.1 Algorithm 1:

- For lesser no. of rules (e.g. 2) that are to be hidden, there is no significant change in the no. of rules generated with increase in no. of transaction.
- When these rules are more in number, the no. of new rules generated becomes more in number.
- In this algorithm, new items are added to the transactions, which lead to increase the support of the item sets. Hence, more frequent item sets are generated leading to the generation of new rules.
- But as the number of transactions increases, number of new rules generated becomes less.

4.3.2.2 Algorithm 2:

- New rules may get generated when the support of the rule is more the threshold support and in future the support of the antecedent decreases to make it an association rule.
- To hide a rule we need to make the confidence or support of the rule below the threshold. Once this is done no more items need to be removed from the transactions.

- Hence, as the no. of transactions increase, the chance that the support of the antecedent decreases holds good to a certain value and then decreases gradually.
- In addition, more the no. of rules to be hidden more is the chance of decreasing the support for an antecedent and hence more association rule generation.

4.3.3 Non-Sensitive Rules Lost

In the following tables, we have provided the results we obtained from our experiments.

Table 4.9: Rules lost for Algorithm 1

No. of transactions	**No. of rules lost (R_H =2)**	**No. of rules lost (R_H =5)**
5000	3	3
10000	3	3
15000	3	3
20000	3	3
25000	3	3
30000	3	3
35000	3	3

Table 4.10: Rules lost for Algorithm 2

No. of transactions	**No. of rules lost (R_H =2)**	**No. of rules lost (R_H =5)**
5000	1	3
10000	2	3
15000	2	4
20000	2	5
25000	3	7
30000	3	9
35000	4	10

Next we have plotted the graphs for the no. of rules lost against number of transactions in the database for both the algorithms. The graphs are shown below:

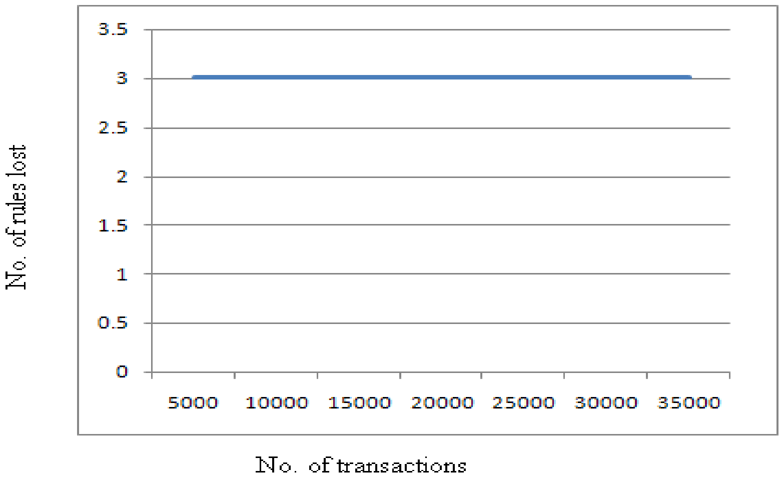

Figure 4.10:No. of rules lost vs. No. of transactions for Algorithm 1

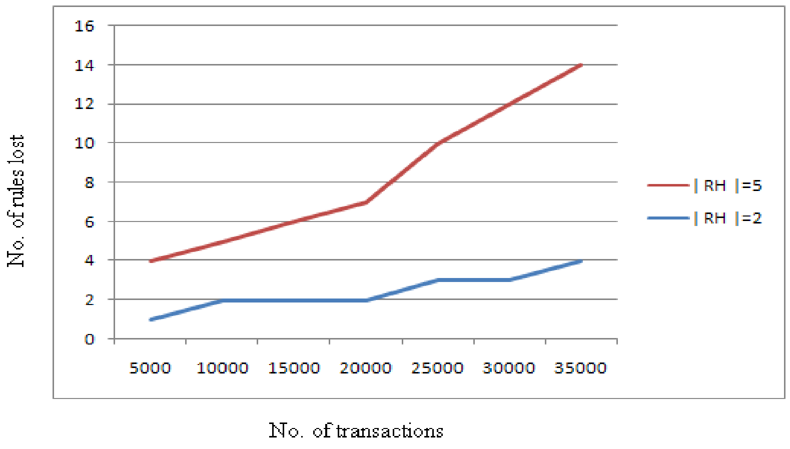

Figure 4.11: No. of rules lost vs. No. of transactions for Algorithm 2

From above graphs and tables we make the following conclusions:

4.3.3.1 Algorithm 1:

- Since in this algorithm hiding strategy depends on increasing the support of the antecedent, hence new items are added into an existing transaction.

- Therefore, the chance of losing a non sensitive rule is minor as it is neither dependent on the number of rules selected for hiding nor on the number of transactions. The values almost tend to zero.

4.3.3.2 Algorithm 2:

In this algorithm the downward slope of the graph is well understood by the following argument:

- To hide a rule we need to make the confidence or support of the rule below the threshold. Once this is done no more items need to be removed from the transactions.
- Hence, as the no. of transactions increase, the no. of rules lost increases.
- In addition, more the no. of rules to be hidden more is the chance of decreasing the support and hence more rules may be lost.

CHAPTER 5

CONCLUSION

In our thesis, we presented the implementation of Apriori algorithm for association rule mining using two different data structures. The first data structure is based the linked list and the second is based on tries data structure. We compared these two implementations on the time they required and the memory they use. The Tries based implementation was better in both the context, but the linked list based implementation required lesser time for small database.

Later, we presented two algorithms for association rule hiding. Both these algorithms are rule based. They decrease either the confidence or the support of a set of sensitive rules, until the rules are hidden. This can happen either because the large item sets that are associated with the rules are becoming small or because the rule confidence goes below the threshold.

We also measured the performance of the proposed algorithms according to two criteria:

a) The time that is required by the hiding process and

b) The side effects that are produced.

As side effects, we considered both the loss and the introduction of information in the database.

We loose information whenever some rules, originally mined from the database, cannot be retrieved after the hiding process. We add information whenever some rules that could not be retrieved before the hiding process can be mined from the released database.

We compared the proposed algorithms on the basis of the results of these experiments and we concluded that there is no best solution for all the metrics. The choice of the algorithm to adopt depends on which criteria one considers as the most relevant: the time required or the information loss or the information that is added.

References:

[1] Charu C. Aggarwal and Philip S. Yu,"Privacy-Preserving Data Mining: A Survey", IBM, T. J. Watson Research Center.

[2] http://www.thearling.com/text/dmwhite/dmwhite.htm

[3] Verykios V. S., Elmagarmid A., Bertino E., Saygin Y., Dasseni E., "Association Rule Hiding", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 4, April 2004.

[4] D. Agrawal and C. C. Aggarwal,"On the Design and Quantification of Privacy Preserving Data Mining Algorithms", *Proc. of ACM PODS Conference*, 2001.

[5] R. Agrawal, T. Imielinski, A. Swami,"*Mining Association Rules* Between Sets of Items in Large Databases", Proc. SIGMOD Conference, 1993.

[6] Rakesh Agrawal and Ramakrishnan Srikant, "Fast algorithms for mining association rules in large databases", In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, Proc. of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

[7] Edward Fredkin (1960), "Trie Memory", *Communications of the ACM*, Vol. **3**, No. 9, Page 490.

[8] Ferenc Bodon, "A Trie-based APRIORI Implementation for Mining Frequent Item sequences", ACM, New York, USA, August 2005.

[9] Arun K. Pujari, " Data Mining Techniques", 14th impression, 2008.

[10] Aaron M. Tenenbaum et.al, "Data Structures Using C and C++", 2nd edition.

[11] Chris Clifton, "Protecting against Data Mining through Samples", *Proc. of the 13th IFIP WG11.3 Conference on Database Security*, 1999.

[12] M. Atallah, E. Bertino, A. Elmagarmid, M. Ibrahim, and V. Verykios," Disclosure Limitation of Sensitive Rules", *Proc. of Knowledge and Data Exchange Workshop*, 1999.

[13] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, "*Advances in Knowledge Discovery and Data Mining*", AAAI Press/The MIT Press, 1996.

[14] Daniel E. O’Leary.” Knowledge Discovery as a Threat to Database Security”, *Proc. of the 1st International Conference on Knowledge Discovery and Databases*” pages 107–516, 1991.