

PRIVACY PRESERVING ASSOCIATION RULE MINING ON VERTICALLY PARTITIONED DATABASE

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

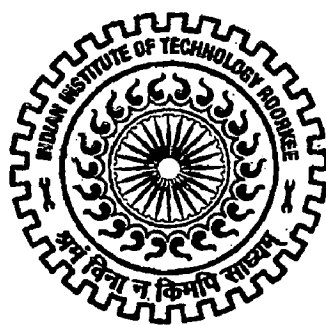
MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

SUSHEELA



DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE -247 667 (INDIA)

JUNE, 2008

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "PRIVACY PRESERVING ASSOCIATION RULE MINING ON VERTICALLY PARTITIONED DATABASE" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology in Computer Science and Engineering** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2007 to June 2008, under the guidance of **Dr. Durga Toshniwal, Assistant Professor, Department of Electronics and Computer Engineering, IIT Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

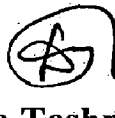
Date: 24/06/08

Place: Roorkee

(SUSHEELA) *Susheela*

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

 24/6/08

Dr. Durga Toshniwal

Assistant Professor

E & CE DEPT.

IIT Roorkee – 247 667

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to my guide **Dr. Durga Toshniwal**, Assistant Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for her able guidance, regular source of encouragement and assistance throughout this dissertation work. It is her vision and insight that inspired me to carry out my dissertation in the upcoming field of 'Privacy Preserving Data Mining'. I would state that the dissertation work would not have been in the present shape without her umpteen guidance and I consider myself fortunate to have done my dissertation under her.

I also extend my sincere thanks to **Dr. D. K. Mehra**, Professor and Head of the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for providing facilities for the work.

I also wish to thank all my friends for their valuable suggestions and timely help.

Finally, I would like to say that I am indebted to my husband for everything that he has given to me. I thank him for the sacrifices he made so that I could grow up in a learning environment. He has always stood by me in everything I have done, providing constant support, encouragement and love.

Susheela

SUSHEELA

ABSTRACT

Association rule mining is a data mining technique used to find interesting associations among a large set of data items. For finding association rules from market-basket databases customer-buying habits between the different items (that customers place in their shopping basket) are analyzed. The discovery of such associations can help retailers develop market strategies by gaining insight into which items are frequently purchased together by customers. Sometimes these association rule mining results disclose some new implicit information about individuals which is against privacy policies.

In vertically distributed databases, the data is vertically partitioned among various sites. These sites wish to work together to find globally valid association rules without revealing individual transaction data. So some privacy-preserving method must be used, which protect the privacy of the distributed databases and at the same time gives accurate association rules.

In this thesis, we propose an algorithm for finding association rules from vertically distributed Boolean databases which maintains a balance between the accuracy of the mining results and the privacy of the databases. For preserving the privacy, database is distorted by XORing the boolean data with a boolean random variable, and then adding some fake transactions in the distorted database. All frequent itemsets are generated for Master's partition. Then intersection of the TIDs of frequent itemsets of Master and real TIDs of other partitions is done. If the intersection value is greater than or equal to some minimum support value (provided by Master Partition) only then the algorithm proceeds. Then the partitions are combined only for the TIDs of Master's partition. Then association rule mining is done by on the combined database and a set of the relative TIDs are made for each candidate itemset. Then again the intersection is performed by third party for each set of TIDs of frequent itemsets to check whether the itemset is frequent in the real TIDs or not. If the third party sends 'OK' then association rules are generated from the frequent itemsets.

CONTENTS

CANDIDATE'S DECLARATION	i	
ACKNOWLEDGEMENTS.....	ii	
ABSTRACT.....	iii	
TABLE OF CONTENTS.....	iv	
Chapter 1	Introduction	1
	1.1 Introduction	1
	1.2 Motivation for work	2
	1.3 Problem Statement	3
	1.4 Organization of the Dissertation	4
Chapter 2	Literature Review	5
	2.1 Association Rule Mining	5
	2.2 Data Modification Methods	8
	2.3 Apriori Algorithm	9
	2.4 Data Layout Alternatives	14
Chapter 3	Proposed Algorithm for Privacy Preserving Association Rule Mining on Vertically Partitioned Database.	18
	3.1 Assumptions for Proposed Algorithm	18
	3.2 Distortion Procedure	20
	3.3 Execution Procedure of Proposed Algorithm	20
	3.4 Phases of the Proposed Algorithm	22

Chapter 4	Implementation Details.	28
	4.1 Database Used	28
	4.2 Distortion Module	28
	4.3 Intersection Module	28
	4.4 Union Module	28
	4.5 Apriori	29
Chapter 5	Result and Discussion	36
	5.1 Results and Discussion	36
	5.2 Analysis	39
Chapter 6	Conclusions and Future Work	41
	6.1 Conclusions	41
	6.2 Suggestions for future work	41
	REFERENCES.	42
	LIST OF PUBLICATIONS	45
	APPENDIX A: SOURCE CODE LISTING.	

CHAPTER 1

INTRODUCTION

1.1 Introduction

Generally, data mining (sometimes called knowledge discovery in data) is the process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases.

Data mining consists of five major elements:

- Extract, transform, and load transaction data onto the data warehouse system.
- Store and manage the data in a multidimensional database system.
- Provide data access to business analysts and IT professionals.
- Analyze the data by application software.
- Present the data in a useful format, such as a graph or table.

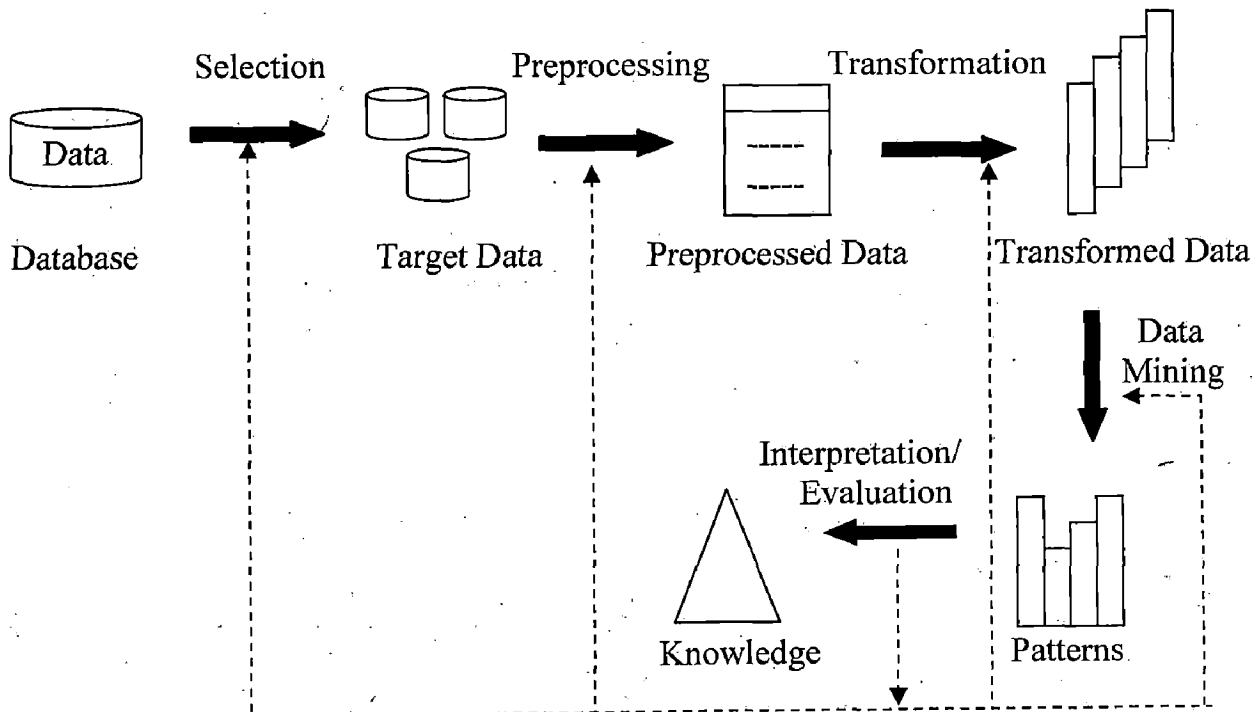


Fig 1.1 Data Mining is the core of Knowledge Discovery Process

1.2 Motivation

Security and privacy are important issues for any data collection because the data is shared and is intended to be used for making some decisions. Also, when we need data for customer profiling, user behavior understanding, etc., large amounts of sensitive and private data about individuals has to be gathered and stored. This makes it difficult to maintain the confidentiality of the data and prevent its illegal access. Also, sometimes data mining results disclose some new implicit information about individuals which is against privacy policies. As the information stored in databases is usually quite valuable, databases with all sorts of contents are regularly sold. Moreover, sometimes data can be withheld for the competitive advantage that can be attained by discovering the implicit knowledge. If data mining results in discovering the implicit knowledge then this information can be widely distributed and used without control [1].

For these reasons, privacy preserving data mining is essentially an emerging area of research in data mining, where data mining algorithms are developed for modifying the original data in some way, so that private data and private knowledge remains private even after the data mining process. The main consideration in privacy preserving data mining is the preservation of sensitive raw data and sensitive knowledge that can be mined from the database. For preserving the privacy, sensitive raw data like identifiers, names, addresses etc. must be modified or removed from the original database, so that the data recipient may not be able to get any personal details of the data provider. Also the sensitive knowledge that can be mined from the database must be omitted; as such information can equally compromise the data privacy [2].

In data mining of vertically partitioned databases, a site want to perform association rule mining from the data partitioned among various sites. But the sites may not want to disclose to each other their individual database for the purpose of preserving the confidentiality of their database. As each site holds some attributes of each

transaction, and the sites wish to work together to find globally valid association rules without revealing individual transaction data. So some privacy-preserving method must be used, which protect the privacy of the partitioned databases and at the same time gives accurate association rules.

A typical example in data mining of partitioned databases where privacy can be of great importance is in the field of medical research. Consider the case where a number of different hospitals wish to jointly mine their patient data, for the purpose of medical research. Privacy policy and law do not allow these hospitals from even pooling their data or revealing it to each other due to the confidentiality of patient records. Although hospitals are allowed to release data as long as the identifiers, such as name, address, and etc., are removed, it is not safe enough because the re-identification attack can link different public databases to relocate the original subjects. In order to pursue mutual gains and relieve the public from the privacy concerns, we need privacy-preserving distributed data mining protocols, which allow distributed data mining to take place while protecting privacy of underlying distributed data.

Another example is multiple competing supermarkets, each having an extra large set of data records of its customers' buying behaviors; want to conduct data mining on their joint data set for mutual benefit. Since these companies are competitors in the market, they do not want to disclose too much about their customers' information to each other, but they know the results obtained from this collaboration could bring them an advantage over other competitors.

1.3 Problem Statement

The aim of proposed research work is to design a technique for preserving the privacy of vertically partitioned database which is used for association rule mining. The following aspects are considered in the designing of the algorithm:

1. The proposed algorithm is designed for boolean data (specific example taken is market-basket data).
2. The partitions are disjoint with respect to each other except TIDs which are common to all.

1.4 Organization of the Thesis

The report is divided into seven chapters including this introductory chapter. The rest of this thesis report is organized as follows:

Chapter 2 provides a brief description of literature review on association rules mining from market basket data, on Apriori algorithm is used widely for discovering large frequent itemset from market-basket data and then the various data modification methods and finally various possible data layout alternatives of market basket data.

Chapter 3 provides a detailed description of proposed algorithms for finding frequent itemsets and finding association rule between the frequent itemsets.

Chapter 4 provides a brief description of the data structure and the implementation details of the proposed algorithm.

Chapter 5 describes the results and discussion on the results. It also provides an analysis on the correctness of the proposed algorithm.

Chapter 6 concludes the dissertation and gives some suggestions for future work.

CHAPTER 2

Literature Review

In this chapter, we discuss about Association rule mining, literature review on privacy preserving association rule mining and various data layout alternatives for market-basket databases.

2.1 Association Rule Mining

Association rule mining finds association or correlation relationships among a large set of data items. With massive amount of data continuously being collected and stored, many industries are becoming interested in mining association rules from their databases. The discovery of interesting association relationship among huge amount of business transaction records can help in many business decision making processes such as catalog design, cross marketing, and loss leader analysis [4].

A typical example of association rule mining is market basket analysis. This process analyzes customer-buying habits by finding association between the different items that customers place in their shopping basket. The discovery of such associations can help retailers develop market strategies by gaining insight into which items are frequently purchased together by customers.

For instances, if customers are buying milk, how likely are they, to also buy bread (what kind of bread) on the same trip to the supermarket? Such information can lead to increase sales by helping retailers do selective marketing and plan their shelf space. For example, placing milk and bread with close proximity may further encourage the sales of items together within single visits to the store.

If we think of the universe as the set of items available at the store, then each item has a boolean variable representing the presence or absence of that item. Each basket can

then be represented by a boolean vector of values assigned to these variables. The Boolean vectors can then be analyzed by buying patterns that reflect items that are frequently associated or purchased together. These patterns are represented in the form of association rules. For example, the information that customers who purchase computers also need to buy financial management software at the same time is represented in association rules as follows:

Computer \Rightarrow *financial _ management _ software* [*support* = 2%, *confidence* = 60%]

Rule support and confidence are two measures of rule interestingness that were described as follows:

Each discovered pattern should have a measure of certainty associated with it that assesses the validity or 'trustworthiness' of the pattern. A certainty measure for association rules of the form ' $A \Rightarrow B$ ', where A and B are sets of items, is confidence. Given a set of task relevant data tuples (or transactions in a transaction database) the confidence of ' $A \Rightarrow B$ ' is defined as follows:

$$\text{Confidence}(A \Rightarrow B) = \frac{\# \text{ tuples containing both } A \text{ and } B}{\# \text{ tuples containing } A}$$

The potential usefulness of a pattern is a factor defining its interestingness. It can be estimated by a utility function, such as support. The support of an association pattern refers to the percentage of task-relevant data tuples for which the pattern is true. For association rules of the form ' $A \Rightarrow B$ ', where A and B are sets of items, the support is defined as follows:

$$\text{Support}(A \Rightarrow B) = \frac{\# \text{ tuples containing both } A \text{ and } B}{\text{Total number of tuples}}$$

A support of 2% for association rule means that 2% of all the transactions under analysis show that computer and financial management software are purchased together. A confidence of 60% means that 60% of the customers who purchased a

computer also bought the software. Typically association rules are considered interesting if they satisfy both a **minimum support threshold** and a **minimum confidence threshold**. Such thresholds can be set by users or domain experts.

Let $\tau = \{i_1, i_2, \dots, i_m\}$ be set of items. Let D be a set of database transactions where each transaction T is a set of items such that $T \subseteq \tau$. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$ where $A \subset \tau$, $B \subset \tau$, and $A \cap B = \phi$. The rule $A \Rightarrow B$ holds in the transaction set D with **support s**, where s is the percentage of transactions of D that contains $A \cup B$ (i.e. both A and B). This is taken to be the probability $P(A \cup B)$. The rule $A \Rightarrow B$ has confidence c in the transaction set D, if c is the percentage of transactions in D containing A that also contain B. This is taken to be conditional probability, $P(B/A)$ that is,

$$\text{Support}(A \Rightarrow B) = P(A \cup B)$$

$$\text{Confidence}(A \Rightarrow B) = P(B/A)$$

Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (min_conf) are called strong. By convention, we write support and confidence values so as to occur between 0% to 100%, rather than 0 to 1.0.

A set of items is referred to as an itemset. An itemset that contains k items is a k-itemset. The set {computer, financial_management_software} is a 2-itemset. The occurrence frequency of an itemset is the number of transactions that contains that itemset. This is also known simply as the **frequency**, **support count** or **count** of the itemset. An itemset satisfies minimum support if the occurrence of frequency of the itemset is greater than or equal to the product of min_sup and the total number of transactions in D. The number of transactions required for the itemset to satisfy minimum support is therefore equal to the **minimum support count**. If an itemset

satisfies minimum support, then it is a frequent itemset. The set of frequent k-itemsets is commonly denoted by L_K .

Association rule mining is a two step process. The two steps are:

- 1. Find all frequent itemsets:** In this step, all those itemsets which occur at least as frequently as a pre-defined minimum support count (considered as frequent itemsets) are calculated.
- 2. Generate strong association rules from the frequent itemsets:** Those rules which satisfy minimum support and minimum confidence (considered as strong association rules) are generated.

The overall performance of mining association rules is determined by the first step.

2.2 Data Modification Methods:

To preserve the privacy of the data, the real data is modified by using different methods of modification discussed as follows:

- (i) Perturbation or Distortion:* In perturbation, privacy is preserved by replacing the original value by a new value or altered by adding some noise in it.
- (ii) Merging:* In merging, privacy is preserved by combining several values into a common category.
- (iii) Swapping:* In swapping, privacy is preserved by interchanging the values of the records each other.

Another way of categorizing data modification techniques is based on heuristics, cryptography and reconstruction.

A. Heuristic-Based Techniques: In heuristic-based techniques only some of the selected values of the data are modified instead of all available values. For modifying the selected values, we can use various methods of data modification like perturbation, merging, swapping, etc.

B. Cryptography-Based Techniques: In cryptography-based techniques, first the data entered by the people is encrypted (changed) by using different cryptography algorithms such that at the end of the computation, no one knows anything except his own input and results. Depending on the type of application, many cryptography algorithms like RSA algorithm, ElGamal Encryption scheme, Triple DES, etc. are used for encrypting the data. For preservation of privacy of the data in cryptography-based techniques, the encrypted data is either kept by a server and the miner queries the server for mining on the data or shared by several miners, who can only jointly mine it. The aim is to protect the private data of the people as much as possible.

C. Reconstruction-Based Techniques: In reconstruction-based techniques, the values in individual records are randomized and then the randomized values are reconstructed and then the new reconstructed values are disclosed for data mining.

In the proposed algorithm, we used the distortion method for modifying the real values of the vertically partitioned databases discussed by S.J. Rizvi and J.R. Haritsa in [9].

2.3 Apriori Algorithm

The common algorithm used to compute large itemset is the Apriori algorithm. The Apriori algorithm has become a data mining classic and most data mining algorithms are based upon it. The first pass of the algorithm simply counts item occurrences to determine the large 1-items. A subsequent pass, say pass k , consists of two phases. First, the large itemset L_{k-1} found in the $(k-1)^{\text{th}}$ pass are used to generate the candidate itemsets C_k , using the Apriorigen function. Then, the database is scanned and the support of the candidates in C_k is counted. For fast counting, we need to efficiently determine the candidates in C_k that are contained in a given transaction. The apriori algorithm works as follows [5]:

1.	$L_1 = \{\text{large 1-itemsets}\};$
2.	for ($k = 2; L_{k-1} \neq \square; k++$) do begin
3.	$C_k = \text{apriori-gen}(L_{k-1})$ //New candidates
4.	forall transactions t in database do begin
5.	$C_t = \text{subset}(C_k, t)$ //Candidates contained in t
6.	forall candidates $c \in C_t$ do begin
7.	$c.\text{count}++;$
8.	End
9.	$L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$
10.	End
11.	Answer = $\bigcup_k L_k$

Apriori-gen function:

The most important step of apriori algorithm is step 3 in the prune step in apriori-gen function, which makes sure that all subsets of a candidate itemset are frequent. The basic idea is that any subset of a large itemset must be large. Therefore, the candidate itemsets having k items can be generated by joining large itemsets having $k-1$ items, and deleting those that contain any subset that is not large. The apriori-gen function takes as argument L_{k-1} , the set of all large $(k-1)$ -itemsets. It returns a superset of the set of all large k -itemsets. The function works as follows. First, the join step joins L_{k-1} with L_{k-1} [5]:

Join Step:

1.	insert into C_k
2.	select $p.\text{item}_1, p.\text{item}_2, \dots, p.\text{item}_{k-1}, q.\text{item}_{k-1}$
3.	from $L_{k-1}p, L_{k-1}q$
4.	where $p.\text{item}_1 = q.\text{item}_1, \dots, p.\text{item}_{k-2} = q.\text{item}_{k-2}, p.\text{item}_{k-1} < q.\text{item}_{k-1};$

Prune Step: The prune step deletes all itemsets $c \in C_k$ such that some $(k-1)$ -subset of c is not in L_{k-1} :

1.	forall itemsets $c \in C_k$ do
2.	forall $(k-1)$ -subsets s of c do
3.	if $(s \notin L_{k-1})$ then
4.	delete c from C_k
5.	end
6.	end
7.	end

Subset function:

Candidate itemsets C_k are stored in a hash tree. A node of the hash tree either contains a list of itemsets (a leaf node) or a hash table (an interior node). In an interior node, each bucket of the hash table points to another node. The root of the hash tree is defined to be at depth 1. An interior node at depth d points to nodes at depth $d+1$. Itemsets are stored in the leaves. When we add an itemset c , we start from the root and go down the tree until we reach a leaf. At an interior node at depth d , decide which branch to follow by applying a hash function to the d^{th} item of the itemset. All nodes are initially created as leaf nodes. When the number of itemsets in a leaf node exceeds a specified threshold, the leaf node is converted to an interior node.

Starting from the root node, the subset function finds all the candidates contained in a transaction t as follows: If we are at a leaf, find which of the itemsets in the leaf are contained in t and add references to them to the answer set. If we are at an interior node and we have reached it by hashing the item i , we hash on each item that comes after i in t and recursively applies this procedure to the node in the corresponding bucket. For the root node, we hash on every item in t .

To see why the subset function returns the desired set of references, consider what happens at the root node. For any itemset c contained in the transaction t , the first

item of c must be in t . At the root, by hashing on every item in t , we assume that we only ignore itemsets that starts with an item not in t . Similar arguments are applied at lower depths. The only additional factor is that, since the items in any itemset are ordered, if we reach the current node by hashing the item i , we only need to consider the items in t that occur after i .

Agrawal and Srikant [6] have proposed the apriori algorithm for discovering all significant association rules between items in a large (not distributed) database of transactions. However, this work does not address privacy concerns.

Later in [7], the authors propose a procedure in which some or all the numerical attributes are perturbed by a randomized value distortion so that both the original values and their distributions are changed. The proposed procedure then performs a reconstruction of the original distribution. This reconstruction does not reveal the original values of the data, and yet allows the learning of decision trees. Another paper [8] shows a reconstruction method, which does not entail information loss with respect to the original distribution.

Other randomization techniques were proposed in order to provide association rules mining without revealing sensitive information about individuals [9, 2]. These techniques are based on probabilistic distortion of user data in the way that can provide a high degree of privacy and retain a high level of accuracy of the result. For example, in [9], the value of the attribute is retained with probability p and flipped with probability $1 - p$. The presented experimental results showed that distortion probability of $p = 0.1$ is ideally suited to provide both privacy and good mining results. . But this work is also for central database of transactions.

In [10] the existing data mining algorithms (for a centralized database) are used for mining association rules from the database which is partitioned among several sites. The algorithm is applied for each site independently and combines the results, but this

method will often fail to achieve a globally valid result. Because this can cause a disparity between local and global results ([11]) include:

- Values for a single entity may be split across sources. Data mining at individual sites will be unable to detect cross-site correlations.
- The same item may be duplicated at different sites, and will be over-weighted in the results.
- Data at a single site is likely to be from a homogeneous population, hiding geographic or demographic distinctions between that population and others.

To overcome the above problems, algorithms were proposed for partitioning data between sites. The algorithms that were proposed for horizontally partitioned data (i.e., each site contains basically the same schema), include Cheung et al. [12], Kantarcioglu and Clifton [13] and Kantarcioglu and Vaidya [14]. Some of them use cryptographic techniques to minimize the amount of disclosed information [13] or a special architecture [14]. This architecture contains sites that sequentially add noise to the original data, compute the answer with noise and remove the noise from the answer. All these methods work with the assumption that no collusion occurs between the sites and the sites follow the protocol precisely.

There has been much work addressing Secure Multiparty Computation. It was first investigated by Yao [15], and later, after Goldreich proved existence of a secure computation for any feasible function [16], some algorithms based on his Circuit Evaluation Protocol have been proposed. But the general method, which is based on Boolean circuits, is inefficient for large inputs. Du and Atallah [17] proposed a more efficient technique for some cases of the multi-party computation problem. One of them is the Two-Party Scalar Product Protocol. In [18], an algorithm is presented for association rule mining which requires the intensive use of secure computation in order to preserve privacy.

The main work on mining association rules from vertically partitioned data across several databases, where the columns in the table are at different sites is done by

Vaidya and Clifton in [11] and by and by Boris Rozenberg and Ehud Gudes in [19]. In [11], Vaidya and Clifton presented some successful solutions for the database vertically partitioned among two sites. But these algorithms have the potential for inferring private information based on the results in certain cases. Then in [19], Boris Rozenberg and Ehud Gudes presented another solution for preserving privacy. But their algorithm also has the potential for inferring private information if the miner has some external knowledge about the customers. This external knowledge problem was one of the main motivations for our algorithms.

2.4 Data Layout Alternatives

Conceptually, a market-basket database is a two-dimensional matrix where the rows represent individual customer purchase transactions and the columns represent the items on sale. This matrix can be implemented in the following four different ways [4], which are pictorially shown in fig. 2.1

Horizontal Item-vector (HIV): The database is organized as a set of rows with each row storing a transaction identifier (TID) and a bit-vector of 1's and 0's to represent for each of the items on sale, its presence or absence, respectively in the transaction (Figure a).

Horizontal Item-list (HIL): This similar to HIV, except that each row stores an ordered list of item-identifiers (IID), representing only the items actually purchased in the transaction (Figure b).

Vertically Tid-vector (VTV): The database is organized as a set of columns with each column storing an item-identifier (IID) and a bit vector of 1's and 0's to represent the presence or absence, respectively, of the item in the set of customer transactions (Figure c). Note that a VTV database occupies exactly the same space as an HIV representation.

Vertical Tid-list (VTL): This is similar to VTV, except that each column stores an ordered list of only the TIDs of the transactions in which the item was purchased (Figure d). Note that a VTL database occupies exactly the same space as an HIL representation.

TID	Item IDs				
	1	2	3	4	5....
1	1	0	1	1	0....
2	0	1	1	0	0....
3	1	0	0	1	1....
4	0	1	1	1	0....

(a) HIV

TID	Item IDs				
	1	1	3	4	7
2	2	3	8	10	
3	1	4	5		
4	2	3	4	9	13

(b) HIL

TID	Item IDs			
	1	2	3	4.....
1	1	0	1	1
2	0	1	1	0
3	1	0	0	1
4	0	1	1	1

(c) VTV

TIDs	Item IDs			
	1	2	3	4.....
1	1	2	1	1
3	3	4	2	3
			4	4

(d) VTL

Fig. 2.1 Data Layout Alternatives of Market-Basket Data

In our research work, we worked on VTV data layout.

Merits of vertical mining:

The vertical layout of market-basket database appears to be a natural choice for achieving association rule mining's objective of discovering correlated items. More specifically, it has the following major advantages over the horizontal layout:

Firstly, computing the supports of the itemsets is simpler and faster with the vertical layout since it involves only the intersections of TID-lists or TID-vectors, operations that are well supported by current database systems. In contrast, complex hash-tree data structures and functions are required to perform the same function for horizontal layouts (e.g. [3]).

Secondly, with the vertical layout, there is an automatic reduction of the database before each scan in that only those itemsets that are relevant to the following scan of the mining process are accessed from disk. In the horizontal layout, however, extraneous information that happens to be part of a row in which useful information is present is also transferred from disk to memory. This is because database reductions are comparatively hard to implement in the horizontal layout. Further, even if reductions were possible, the extraneous information can be removed only in the scan following the one in which its irrelevance is discovered. Therefore, there is always a reduction lag of at least one scan in the horizontal layout.

Thirdly, bit-vector formats, due to their sequences of 0's and 1's, offer scope for compression. From this perspective also, the vertical layout is preferred since a VTV format results in higher compression ratios than the equivalent HIV format. This is because compression techniques typically perform better with large datasets since there is greater opportunity for identifying repeating patterns – in a VTV, the length of the dataset is proportional to the number of customer transactions, whereas for HIV, it is limited to the number of items in the database, usually a fixed quantity that is small relative to the number of tuples in the database.

Finally, the vertical layout permits asynchronous computation of the frequent itemsets. For example, given a database with items A, B, C, once the supports of the items A and B are known, counting the support of their combination AB can commence even if item C has not yet been fully counted. This is in marked contrast to the horizontal approach where the counting of all itemsets has to proceed synchronously with the scan of the database. A careful algorithmic design is required to ensure that the above mentioned inherent advantages of the vertical layout are translated into tangible performance benefits.

CHAPTER 3

Proposed Work for Privacy Preserving Association Rule Mining on Vertically Partitioned Database

In this chapter we will discuss the proposed algorithm for preserving the privacy of the vertically partitioned databases when they are used for discovering frequent itemsets. Our work is an extension of the work done by B. Rozenberg et al. in [19].

3.1 Assumptions for Proposed Work

In our proposed algorithm, we assume partition i of vertically partitioned database as Master partition, who wants to find out the frequent itemsets. Other partitions of the database only provide their partition data for mining but will not do any global computations & Third Party performs intersection of the TIDs. This third party is not trusted with the database, but it is trusted with computations).

We assume that in all partitions of the database, the domain of the TIDs is the same and its size is equal to n - some number that depends on the area of the business. The number of transactions in each partition is up to n and the TIDs range from 1 to n . When fake transactions are introduced, they use "unoccupied" TIDs. When information between the parties is shared, then only information in which some attributes are real (in one of the databases) is of use. That is, a fake transaction whose corresponding TID in the other database is empty, is not considered at all. Also when each partition computes large itemsets it does not know whether the attributes corresponding to his real transaction, are real or not.

The flow chart for the proposed approach for privacy preserving association rule mining is drawn in Figure 3.1. . The flowchart is showing all 4 phases of the proposed algorithm which are described just after the figure.

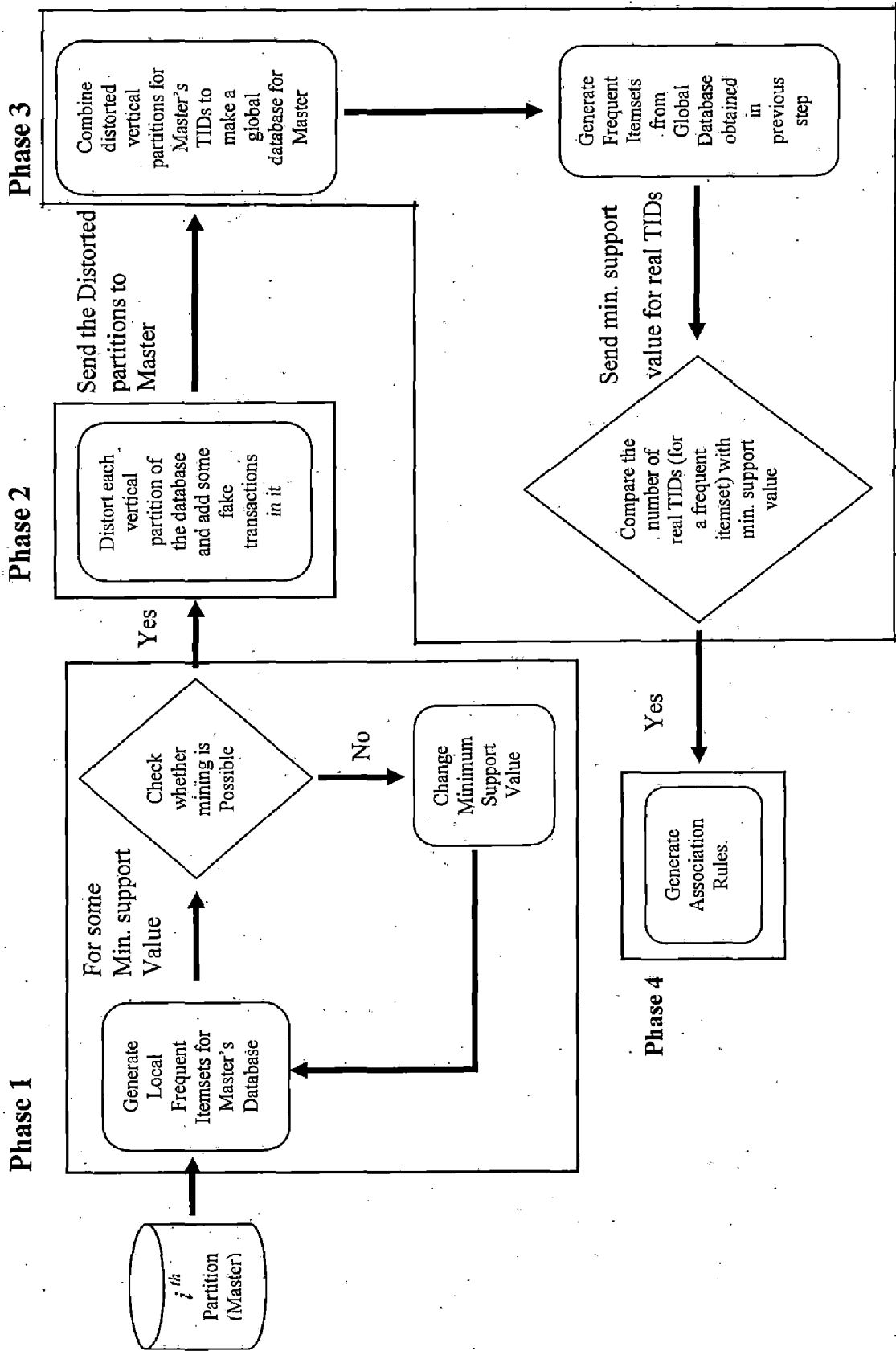


Fig. 3.1 Flowchart of the Proposed Work for Privacy Preserving Association Rule Mining

3.2 Distortion Procedure:

There are many data modification methods for distorting the data as discussed in chapter 2. We are using the following data distortion method for modifying or distorting the values of the vertically partitioned database described by S.J. Rizvi et al. in [9]:

A customer tuple can be considered to be a random vector $X = \{X_i\}$, such that $X_i = 0$ or 1 . We generate the distorted vector from this customer tuple by computing $Y = \text{distort}(X)$ where $Y_i = X_i \text{ XOR } r_i$, and r_i is $(100 - r_i)$, r_i is a random variable with density function $f(r) = \text{bernoulli}(p)$ ($0 \leq p \leq 1$). That is, r_i takes a value 1 with probability p and 0 with probability $1 - p$.

After distorting the database with the above described method, add some fake transactions (by simply putting 1 randomly in non existing transactions) in the distorted database to get the final distorted database.

3.3 Execution Procedure of the proposed algorithm:

The execution procedure of the proposed algorithm is as follows:

1. Firstly phase 1 (can be called as 'Mining Check phase') is executed, in which the Master (ith partition in vertically distributed database) finds out all frequent itemsets in its own partition for some minimum support value by using apriori algorithm discussed in chapter 2. After finding all frequent itemsets Master partition sends the TIDs of the frequent itemsets and the other partitions send their real TIDs to the Third Party. Then the Third Party find out the intersection of the TIDs sent to him by using the 'Third Party's execution process'. If the intersection is greater than or equal to the minimum support value, the Third Party sends "Ok" (which means mining is possible) to all partitions, otherwise sends "NotOk" (which means mining is not

possible) to all partitions. The algorithm proceeds only if all partitions get “Ok” from the Third Party.

2. If all partitions gets “Ok” from the Third Party then phase 2 is executed. In this phase, all partitions distort their own data by using the distortion method described above and by adding fake transactions in the distorted data and send it to Master partition.
3. Then phase 3 is executed in which Master partition makes the global database by joining the Item TID’s values of all partitions only for the real TIDs of its own partition. After making a global database, the Master partition finds all frequent itemsets from the global database. Then the Master sends the transaction IDs of the frequent itemset to the Third Party with some minimum support value to check that the number of real transactions present in a frequent itemset. The Third Party executes the ‘Third Party’s execution process’ to calculate the size of the intersection of the received set with the set of all real ID’s of all partitions of the database other than master partition, if this intersection value is greater than equal to the minimum support value, it sends an “OK” to the Master, which means that the itemset is also frequent in the all real partitions of the database.
4. Then phase 4 is executed, in which association rules are generated for all those frequent itemsets for which the third party sends ‘OK’.

In this algorithm there is no communication with partitions other than Master partition after initial submission of their partition data and since the Third Party just answers “Ok” or “Not Ok” to the Master partition, the master partition knows only that some minimum number of the transaction IDs (equal to the minimum threshold value) are common in both databases. Also, the Third Party does not have knowledge of the partition’s data, so its role and the trust required is very limited.

3.4 Phases of the proposed algorithm:

1. **Phase 1:** This phase is 'Mining check' phase. The following steps are included in this phase:

1.	Using apriori-gen function, generate all frequent itemsets from Master's partition.
2.	Send all TIDs of Master partition present in frequent itemsets and all the real TIDs of all partitions other than Master partition to the Third party.
3.	Receive the response from the Third party whether mining is possible or not (See third party execution phase). If the mining is possible then continue.

2. **Phase 2:** This phase is 'Data Preprocessing' phase. The following steps are included in this phase:

1.	Distort all vertical partitions of the database (by using the distortion procedure) and then add some fake transactions in it.
2.	Send all distorted partitions to the Master.

3. **Phase 3:** This phase is 'Master's Execution phase'. The following steps are included in this phase:

1.	Build the global database (GDB) with true TIDs from its own partition and attributes from its own and other partitions of the database.
2.	Using apriori-gen function, generate all frequent itemsets from GDB.
3.	For each frequent itemset: <ul style="list-style-type: none">i. Build the set of relative <i>TIDs</i>.ii. Check with the third party whether the itemset is frequent.

Third party execution phase: The 'Third party execution phase' used in the above 3 phases for calculating the intersection of TIDs includes the following the steps:

1.	Receive <i>TIDs</i> from the two parties (<i>MTID</i> , <i>STID</i>)
2.	If ($ MTID \cap STID \geq minsup$) send "OK" while (Master not finished) do receive set of <i>TID</i> 's from Master if ($ MTID \cap STID \geq minsup$) then send "OK" to the Master else send "NotOK" to the Master end while else // mining is not possible! Send "NotOK"

Phase 4: This phase is 'Association rule generation phase'. In this phase firstly, the value of c (the minimal confidence value) is taken for by the Master partition. Then the following Master's execution process and Third party execution process is executed for calculating the confidence:

a) Master's Execution process:

1.	Send c to the third party.
2.	For each frequent itemset Z , generate all possible rules $X \rightarrow Y$ such that $Z = XY$ according to the Master's real transactions.
3.	For each rule from step 2 do: Generate two sets of ids: TID_x (<i>IDs</i> of all real transaction that contain X) and TID_{xy} (<i>IDs</i> of all real transaction that contain XY). Send TID_x and TID_{xy} to the 3 rd party. Receive from the 3 rd party "OK" or "NOT".
4.	Send to 3 rd party "FINISH".

b) Third party execution process::

1.	Receive c from the Master.
2.	While not "FINISH" do: Receive two sets from the Master (TID_x and TID_{xy}). Calculate answer = $\left(\left \frac{TID_{xy} \cap STID}{TID_x \cap STID} \right \geq c \right)$, where $STID$ is the set of all real TID s of the other partitions. Send answer to the Master.

Now we will explain the proposed algorithm with the help of sample database (vertically partitioned in two sites). Let the 1st (Master's) and 2nd two vertical partitions of the databases are:

Table 3.1 1st (Master's) Partition's Real Data

Transaction IDs	Item IDs			
	A	B	C	D
1	1	1	0	0
2	1	1	0	0
6	1	1	1	0
7	1	1	0	1
9	1	1	0	1

Table 3.2 2nd Partition's Real Data

Transaction IDs	Item IDs					
	E	F	G	H	I	J
1	1	1	1	0	0	0
2	1	1	1	1	1	0
4	0	1	0	1	0	1
6	1	1	1	0	1	1
7	1	1	1	0	0	0
10	0	1	0	1	0	1

Let the value of minimum support is 4. Master partition will find the frequent itemsets in its partition. {A, B} is frequent itemset in Master's partition. The TIDs in which {A, B} is present are {1, 2, 6, 7, 9}. So, both partitions sends TIDs {1, 2, 6, 7, 9} and {1, 2, 4, 6, 7, 10} to Third Party. Now Third Party calculates the size of the intersection of the two sets ($|\{1, 2, 6, 7, 9\} \cap \{1, 2, 4, 6, 7, 10\}| = 4$) and sends 'OK' to each partition which means that mining is possible (because the size of intersection \geq minimal support). After this step, each side distorts the database by using the distortion method described above and then add some fake transactions in their partitions and then 2nd partition sends the resulting distorted database to the 1st (Master partition) partition.

Table 3.3 1st Partition's (Master's) Distorted Database (with $r = 80\%$) without fake transactions

Transaction IDs	Item IDs			
	A	B	C	D
1	1	1	0	0
2	1	1	0	1
6	1	1	1	0
7	1	1	1	1
9	1	1	0	1

Table 3.4 2nd Partition's Distorted Database (with $r = 80\%$) without fake transactions

Transaction IDs	Item IDs					
	E	F	G	H	I	J
1	1	1	1	0	0	1
2	1	1	1	1	0	0
4	0	1	0	1	0	1
6	1	1	0	1	1	1
7	1	1	1	0	0	0
10	0	1	0	1	0	1

Table 3.5 1st Partition's (Master's) Distorted Database with fake transactions

Transaction IDs	Item IDs			
	A	B	C	D
1	1	1	0	0
2	1	1	0	1
3	0	1	1	1
4	0	1	1	0
5	1	0	0	1
6	1	1	1	0
7	1	1	1	1
8	1	0	0	1
9	1	1	0	1
10	1	0	1	0

Table 3.6 2nd Partition's Distorted Database with fake transactions

Transaction IDs	Item IDs					
	E	F	G	H	I	J
1	1	1	1	0	0	1
2	1	1	1	1	0	0
3	1	1	1	1	0	1
4	0	1	0	1	0	1
5	1	0	1	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	0	0
8	0	0	0	1	1	0
9	1	1	1	0	0	0
10	0	1	0	1	0	1

Table 3.7 1st Partition's (Master's) Global Database

Transaction IDs	Item IDs									
	A	B	C	D	E	F	G	H	I	J
1	1	1	0	0	1	1	1	0	0	1
2	1	1	0	0	1	1	1	1	0	0
6	1	1	1	0	1	1	0	1	1	1
7	1	1	0	1	1	1	1	0	0	0
9	1	1	0	1	1	1	1	0	0	0

Table 3.8 2nd Partition's Global Database

Transaction IDs	Item IDs									
	A	B	C	D	E	F	G	H	I	J
1	1	1	0	0	1	1	1	0	0	0
2	1	1	0	1	1	1	1	1	1	0
4	0	1	1	0	0	1	0	1	0	1
6	1	1	1	0	1	1	1	0	1	1
7	1	1	1	1	1	1	1	0	0	0
10	1	0	1	0	0	1	0	1	0	1

Now Master generates the frequent itemsets from its global database by using apriori-generation function and he found that itemset $I = \{A, B, E, F, G\}$ is frequent in the global database. Then the Master wants to know if the itemset $I = \{A, B, E, F, G\}$ that is frequent in the above global Master's database is frequent in the 2nd Partition. The Master sends the transaction IDs that contains the frequent itemset to the Third Party ($\{1, 2, 6, 7, 9\}$). The Third Party calculates the size of the intersection of the received set with the set of all real TID's of 2nd partition, and since the size of the result (1, 2, 6, 7) is greater than or equal to 4, it sends an "Ok" to the Master.

CHAPTER 4

IMPLEMENTATION DETAILS

The implementation of the proposed Algorithm is done in C++ and Java. The implementation details of the proposed algorithm are listed as follows:

4.1 DATABASE

The vertical partition database used in the thesis is the database used in [22]. The transaction database is taken as an $m \times n$ matrix. Transaction 1 appears in row one. Columns are separated by a space and represent items. A 1 indicates that item is present in the transaction and a 0 indicates it is not.

4.2 DISTORT.C

This program is used to distort the Boolean Market-basket database. The program takes input from a file which contains the boolean database. A boolean random number is generated by using `rand ()` function. The amount of occurrence of 1 and 0 for this random number can be controlled by the user. Then this random number is XORed with the values in the database for distorting the database values. Then the distorted database is stored in other file.

4.3 INTERSECT.C

This program calculates the number of common transaction IDs from the TIDs entered in it. The some value for minimum support is also provided. If the number of common transactions is greater than or equal to the minimum support value, the program returns "OK" to the database owner, else return "NotOK".

4.4 UNION.C

This program is used to merge different partitions of the database on the basis of the TIDs provided. First, the transactions IDs (TIDs) of the Master's partitions are matched with the TIDs of the other partitions of the database. If a TID is common in

all partitions of the database then the values are combined. If the TID is present only in Master's database then the item values for that TID are taken as they are and '0' is entered for items of the other partitions.

4.5 APRIORI.JAVA

This program creates a user interface, which contains three buttons – Open File, Add Support, and Run. 'Open File' button is used to open the database file. 'Add Support' button is used to add the minimum support for finding association rules. 'Run' button is used to run the apriori algorithm.

public Apriori (String s)

This constructor is used to create user interface.

public void actionPerformed(ActionEvent event)

This function is used to perform the –open file, read file, add minimum support and run actions.

protected void createTtreeTopLevel ()

This function generates top level (i.e. 1st level) of the T-tree.

protected void createTtreeLevelN()

This function performs the process of determining the remaining levels in the T-tree (other than the top level), level by level in an "Apriori" manner by adding support, then performing pruning and generate loop until there are no more levels to generate.

protected void addSupportToTtreeLevelN(int level)

This function performs the process of adding support to a given level in the T-tree (other than the top level).

private void addSupportToTtreeFindLevel (TTreeNode[] linkRef, int level, int endIndex, short[] itemSet)

This function operates in a recursive manner to first find the appropriate level in the T-tree before processing the required level (when found). linkRef is the reference to the current sub-branch of T-tree (start at top of tree), level is the level marker, set to the required level at the start and then decremented by 1 on each recursion. endIndex is the length of current level in a sub-branch of the T-tree. itemSet is the current itemset under consideration.

protected void pruneLevelN(TreeNode [] linkRef, int level)

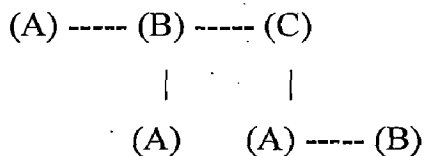
This function performs the pruning of the given level in the T-tree. Pruning carried out according to value of minSupport field. linkRef is the reference to the current sub-branch of T-tree (start at top of tree), level is the level marker, set to the required level at the start and then decremented by 1 on each recursion.

protected void generateLevel2()

This function generates level 2 of the T-tree. The general 'generateLevelN' method assumes we have to first find the right level in the T-tree, that is not necessary in this case of level 2.

protected void generateLevelN (TreeNode[] linkRef, int level, int requiredLevel, short[] itemSet)

This function performs the process of generating remaining levels in the T-tree (other than top and 2nd levels) by proceeding in a recursive manner level by level until the required level is reached. Example, if we have a T-tree of the form:

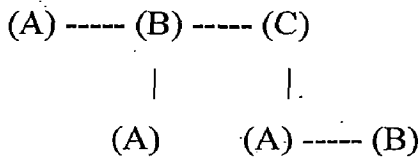


Where all nodes are supported and we wish to add the third level we would walk the tree and attempt to add new nodes to every level 2 node found. Having found the correct level we step through starting from B (we cannot add a node to A), so in this case there is only one node from which a level 3 node may be attached. linkRef is the reference to the current sub-branch of T-tree (start at top of tree). level is the level

marker, set to 1 at the start of the recursion and incremented by 1 on each repetition. requiredLevel is the required level. itemSet is the current itemset under consideration.

protected void generateNextLevel (TTreeNode[] parentRef, int endIndex, short[] itemSet)

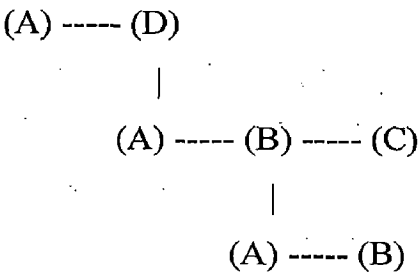
This function generates a new level in the T-tree from a given "parent" node. Example 1, given the following:



where we wish to add a level 3 node to node (B), i.e. the node {A}, we would proceed as follows:

Generate a new level in the T-tree attached to node (B) of length one less than the numeric equivalent of B i.e. 2-1=1. Loop through parent level from (A) to node immediately before (B). For each supported parent node create an itemset label by combing the index of the parent node (e.g. A) with the complete itemset label for B -- {C, B} (note reverse order), thus for parent node (B) we would get a new level in the T-tree with one node in it --- {C, B, A} represented as A. For this node to be a candidate large item set its size-1 subsets must be supported, there are three of these in this example {C,A}, {C,B} and {B,A}. We know that the first two are supported because they are in the current branch, but {B, A} is in another branch. So we must generate this set and test it. More generally we must test all cardinality-1 subsets which do not include the first element. This is done using the method testCombinations.

Example 2, given:



where we wish to add a level 4 node (A) to (B) this would represent the complete label {D, C, B, A}, the N-1 subsets will then be {{D, C, B}, {D, C, A}, {D, B, A} and {C, B, A}}. We know the first two are supported because they are contained in the current sub-branch of the T-tree, {D, B, A} and {C, B, A} are not. parentRef is the reference to the level in the sub-branch of the T-tree under consideration. endIndex is the index of the current node under consideration. itemSet is the complete label represented by the current node (required to generate further itemsets to be X-checked).

protected boolean testCombinations (short[] currentItemSet)

This function performs the process of testing whether the N-1 sized sub-sets of a newly created T-tree node are supported elsewhere in the Ttree --- (a process referred to as "X-Checking"). Thus given a candidate large itemsets whose size-1 subsets are contained (supported) in the current branch of the T-tree, tests whether size-1 subsets contained in other branches are supported. Proceed as follows:

Using current item set split this into two subsets: itemSet1 = first two items in current item set, itemSet2 = remainder of items in current item set, Calculate size-1 combinations in itemSet2, For each combination from (2) append to itemSet1.

Example 1:

currentItemSet = {A,B,C},

itemSet1 = {B, A} (change of ordering),

size = {A, B, C}-2 = 1

itemSet2 = {C} (currentItemSet with first two elements removed).

Now calculate combinations between {B, A} and {C}.

Example 2:

currentItemSet = {A, B, C, D}

itemSet1 = {B, A} (change of ordering)

itemSet2 = {C, D} (currentItemSet with first two elements removed)

calculate combinations between {B, A} and {C, D}

currentItemSet the given itemset.

private boolean combinations (short[] sofarSet, int startIndex, int endIndex, short[] itemSet1, short[] itemSet2)

This function determines the cardinality N combinations of a given itemset and then checks whether those combinations are supported in the T-tree. Operates in a recursive manner.

Example 1: Given --- sofarSet = null, startIndex = 0, endIndex = 2, itemSet1 = {B, A} and itemSet2 = {C},

itemSet2.length = 1, endIndex = 2 greater than itemSet2.length
if condition succeeds

testSet = null + {B, A} = {B, A}

return true if {B, A} supported and

return null otherwise

Example 2: Given --- sofarSet = null, startIndex = 0, endIndex = 2, itemSet1 = {B, A} and itemSet2 = {C, D}

endindex not greater than length {C, D}

go into loop

tempSet = {} + {C} = {C}

combinations with --- sofarSet={C}, startIndex=1,

endIndex=3, itemSet1 = {B, A} and itemSet2 = {C}

endIndex greater than length {C, D}

testSet = {C} + {B, A} = {C, B, A}

tempSet = {} + {D} = {D}

combinations with --- sofarSet={D}, startIndex=1,

endIndex=3, itemSet1 = {B, A} and itemSet2 = {C}

endIndex greater than length {C, D}

testSet = {D} + {B, A} = {D, B, A}

sofarSet is the combination itemset generated so far (set to null at start), startIndex is the current index in the given itemSet2 (set to 0 at start). endIndex is the current index of the given itemset (set to 2 at start) and incremented on each recursion until it is

greater than the length of itemset2. itemSet1 is the first two elements (reversed) of the total label for the current item set. itemSet2 is the remainder of the current item set.

private boolean findItemSetInTtree (short[] itemSet)

This function commences the process of determining if an itemset exists in a T-tree. Used to X-check existence of Ttree nodes when generating new levels of the Tree. Note that T-tree node labels are stored in "reverse", e.g. {3, 2, 1}. itemset is the given itemset (in reverse order). It returns true if itemset found and false otherwise.

private boolean findItemSetInTtree2 (short[] itemSet, int index, int lastIndex, TTreeNode[] linkRef)

This function returns true if the given itemset is found in the T-tree and false otherwise. It operates recursively. itemSet is the given itemset. index is the current index in the given T-tree level (set to 1 at start). lastIndex is the end index of the current T-tree level. linkRef is the reference to the current T-tree level.

private void addSupport()

This function add the minimum support value and enable run button if have data and a minimum support value.

private void getFileName()

This function displays an open file dialog box so that the user can select file to open. If "OK" button is clicked then the file is opened and if cancel button is clicked then return. Obtain selected file and read the file if the file is readable (i.e. not a directory etc.). Enable run button if have data and a minimum support value. Output the no. of rows & columns to text area.

private boolean checkFileName()

This function check the whether the selected file is a text file or not. It returns false if selected file is a directory or is not a file name or access is denied.

private void checkLine (int counter, String str)

This function check whether input file is of appropriate boolean input.

public void inputDataSet() throws IOException

This function reads input data from file specified in command line argument.

protected void outputItemSet (short[] itemSet)

This function outputs a given item set. itemSet is the given item set.

public void outputFrequentSets()

This function commences the process of outputting the frequent sets contained in the T-tree.

private int outputFrequentSets (int number, short[] itemSetSofar, int size, TTreeNode[] linkRef)

This function outputs T-tree frequent sets. It operates in a recursive manner. number is the number of frequent sets so far. itemSetSofar is the label for a T-treenode as generated sofar. Size is the length/size of the current array level in the T-tree. linkRef is the reference to the current array level in the T-tree. It returns the incremented (possibly) number the number of frequent sets so far.

protected short[] realloc1 (short[] oldItemSet, short newElement)

This function resizes given item set so that its length is increased by one and append new element. oldItemSet is the original item set. newElement is the new element/attribute to be appended. It returns the combined item set.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Results and Discussion

Impact of Distortion on Privacy of data:

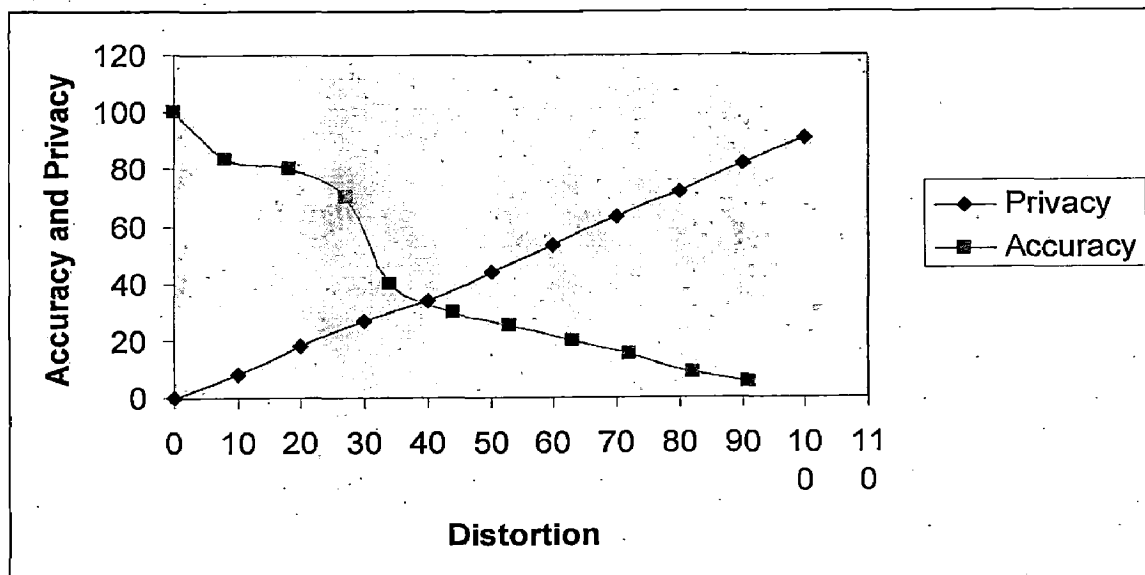


Figure 5.1 Distortion v/s Accuracy and Privacy of the database

The above graph is showing that as the distortion of the database increases, the accuracy of the frequent itemsets decreases, but the privacy of the database increases. So we can choose the amount of distortion done on the database where requirement of accuracy and privacy both are fulfilled.

Impact of Distortion on Accuracy of Association Rules:

The graph drawn below is showing that as the distortion (probability of occurrence of 1 as random variable) increases the accuracy of the frequent itemsets decreases. As the distortion increases, the frequent itemsets contains some false positives and false negatives. False positive means that the distorted database contains some frequent

itemsets which are not frequent in the real database. False negative means that some of the itemsets which are frequent in the real database are not frequent in the distorted database.

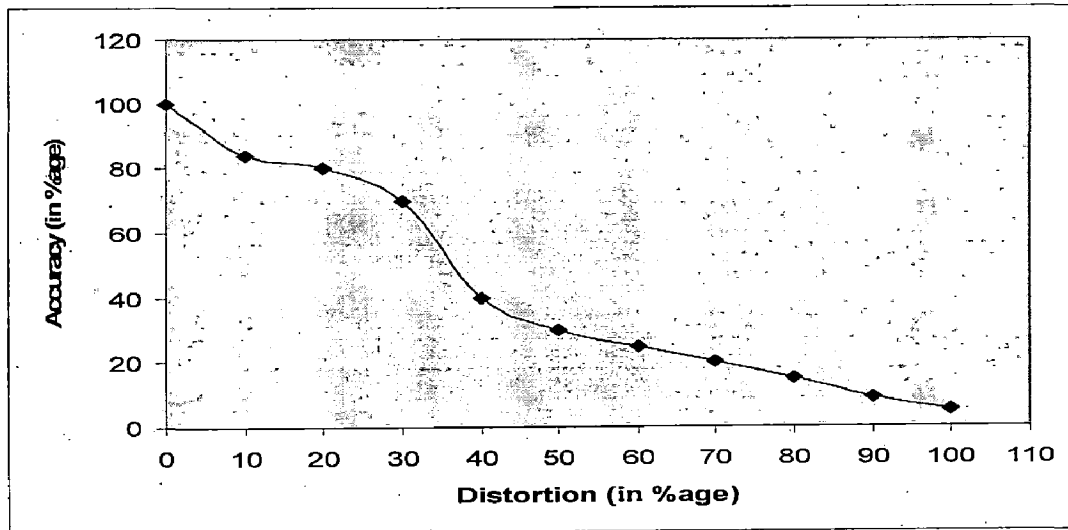


Fig.5.2 Probability of occurrence of 1 v/s accuracy of the frequent itemsets

Execution Time v/s Database size:

As the database size (number of transactions) increases, the execution time of finding frequent itemset also increases.

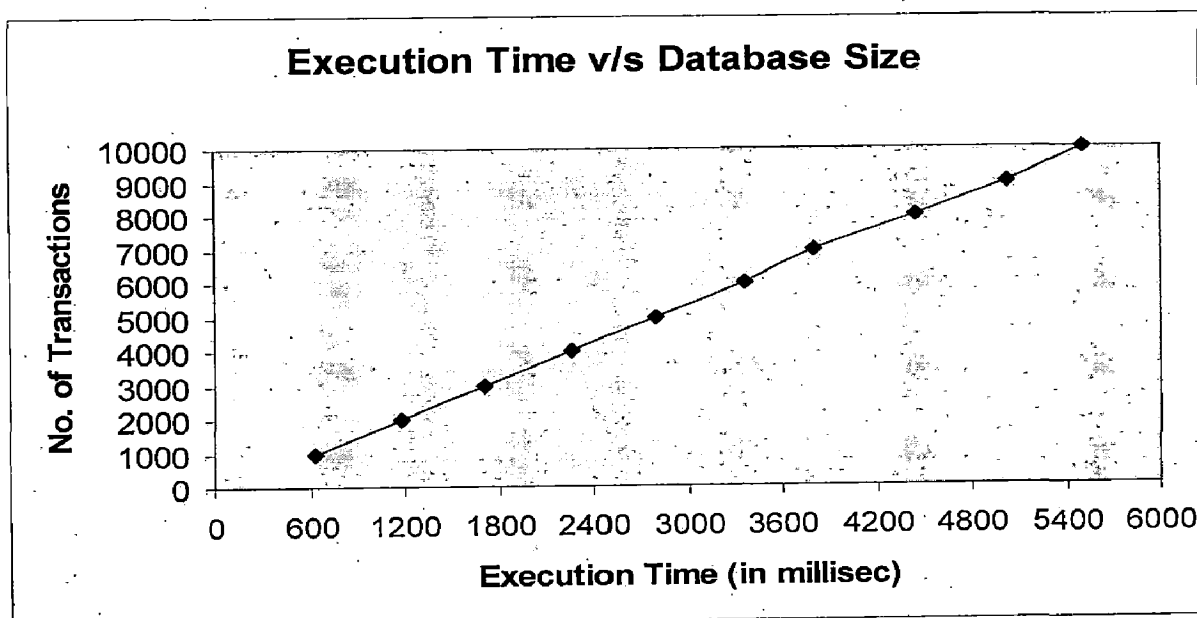


Fig 5.3 Database Size v/s Execution Time

In the above graph the execution time is for a database having 8 itemsets with 30% minimum support. As the database size (number of transactions) increases the execution time of apriori algorithm for generating frequent itemsets also increases.

Impact of Minimum Support on Execution time:

The value of minimum support directly affects the execution time of the algorithm. As we are assuming that in market-basket database, the number of 1's is less than the number of 0's. So the number of itemsets having high minimum support is less. If the value of minimum support is less then the number of frequent itemsets is more and if the value of minimum support is more then the number of frequent itemsets is less. The graph below is showing that as the minimum support for generating frequent itemset increases, the execution time decreases.

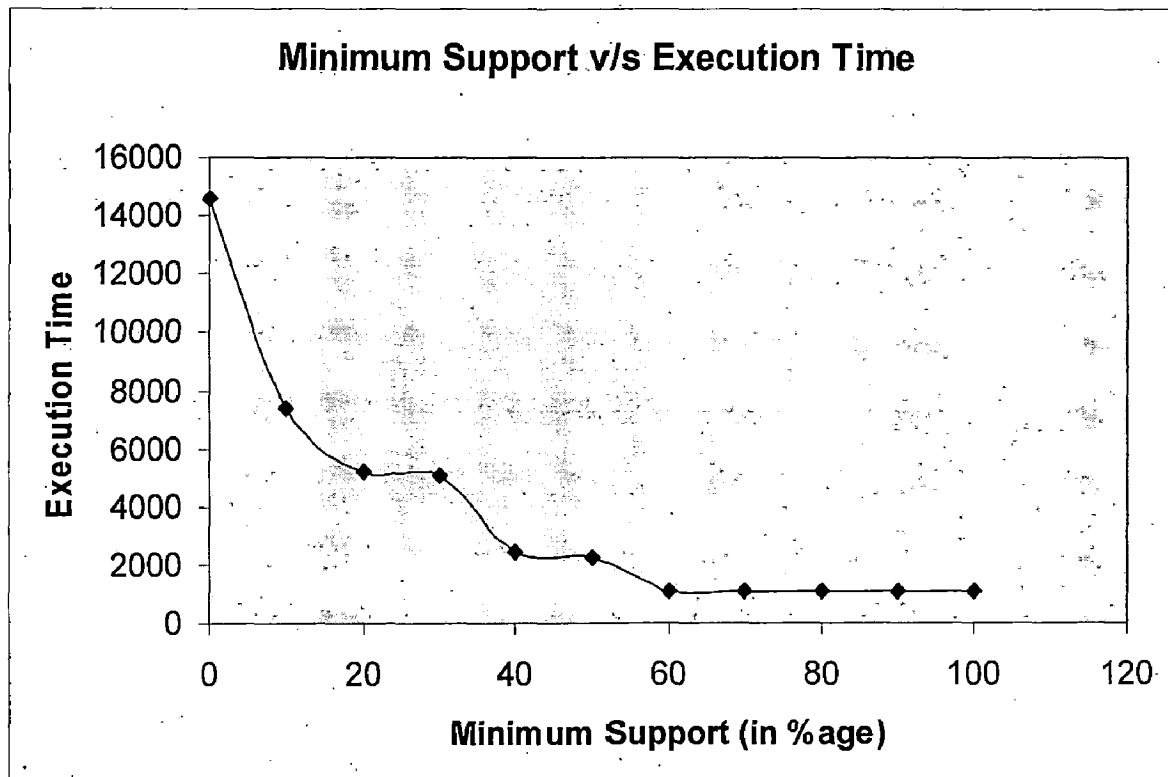


Fig 5.4 Minimum Support Value v/s Execution Time

5.2 Analysis

Correctness:

The algorithms for frequent itemsets mining use the apriori-gen function to generate all candidates for large itemsets. In each iteration k , a large k -itemset that was found by the Master according to its true transactions is checked by the other party. So, all existing large itemsets are found. In the algorithm for association rules mining, the Master generates all possible rules from frequent itemsets found earlier and checks with the 2nd Party the resulted confidence. So, all existing rules are found.

Information disclosed by our algorithm:

Here, we discuss the information disclosed before the algorithm starts and information disclosed during and after the algorithm operation. Our algorithm disclosure depends whether we assume the existence of external knowledge. If no external knowledge is assumed, then before starting the algorithm, the only information the Master has is that mining is possible. Since Master Party has no knowledge of the real or fake transactions and their values, the probability of any transaction of the 2nd Party to be true is minsup/r , where r is the number of real transactions on the local database.

If no external knowledge is assumed, still a corrupt Master is able to learn exactly which transactions in the other party's database are fake. In order to do so, it should operate in the following way: Assume that the minimal support threshold is 4. The Master sends to the trusted party sets of exactly four TIDs until it receives an "OK" answer, which means all four TIDs are not fake. Then it chooses three of these, and for every other TID j it sends to the trusted party a set containing these three TIDs together with j . The answer of the trusted party is "OK" if and only if j is not a fake TID. But, as the database is distorted, then also the only information the Master has is

that this individuals participate in the two databases, but he does not know whether that real transaction is really “real” or not.

If there is any external knowledge, as the database is distorted, then also the only information the Master party has is that this individuals participate in the two databases, but he does not know whether their real transaction is really “real” or not. With distortion probability of $p = 0.1$, the resulted expected error is less than $(k/r * 10) \%$, where k is the number of suspected transactions, and r is the number of real transactions of some side and usually $k \ll r$.

At present there is no standard definition for the measure of privacy loss. In our algorithm, we use as the measure for privacy loss, the probability of learning whether a particular transaction value is real or fake.

In our algorithm, the parties do not know the exact support for each tested itemset. This decreases the probability that the Master will learn that a set of items on another site has a given property, and it occurs only when the global support value is above or equal to the threshold value, and is also equal to the Master’s support.

CHAPTER 6

CONCLUSIONS

6.1 Conclusions

We focus our attention on the problem of privacy preserving association rule mining in vertically partitioned databases. While other existing approaches try to overcome the problem of information disclosure, there still exist cases in which some information may be disclosed. In this work, we propose an algorithm for discovering all frequent itemsets and then generating association rules from them in vertically partitioned databases, without disclosing individual transaction values. The proposed algorithm preserve the privacy of vertically partitioned data by distorting all partitions data (by XORing the boolean data values with a boolean random variable) and then adding some fake transaction in the distorted data. The proposed algorithm does mining check (to check whether some frequent itemsets occur in the Master partition) in the starting phase of the algorithm to avoid the unnecessary computations. The proposed algorithm also put a limit on the number of fake transactions responsible for making an itemset as frequent itemset. This algorithm reduces the amount of disclosed information up to some extent. In our proposed algorithm, a balance between privacy and accuracy can be maintained by choosing the amount of distortion.

6.2 Suggestions for further work

In our proposed work, we had used Boolean vertically partitioned database. In future work, we will try to extend the proposed algorithm for numeric database.

REFERENCES

1. Osmar R. Zaïane, "*Chapter I: Introduction to Data Mining*", CMPUT690 Principles of Knowledge Discovery in Databases, 1999.
2. V.S. Verykios, E. Bertino, I.N. Fovino, L.P. Provenza, Y. Saygin, and Y. Theodoridis, "*State-of-the-Art in Privacy Preserving Data Mining*", ACM SIGMOD Record, vol. 3, no. 1, Mar. 2004, pp. 50-57.
3. R. Agarwal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami, "*An Interval Classifier for Database Mining Applications*", In. Proc. of the VLDB Conference, Vancouver, British Columbia, Canada, August 1992, pp. 560-573.
4. A. Savasare, E. Omiescinski, and S. Navathe, "*An Efficient Algorithm for Mining Association Rules in Large Databass*", In. Proc. of 21st Intl Conf on Very large Databases (VLDB), 1995, pp. 432-444.
5. R. Agrawal and R. Srikant, "*Fast Algorithms for Mining Association Rules in large Databases*", Research Report RJ 9839, IBM Almaden Research Center, San Jose, California, pp. 207-216, June 1994.
6. R. Agrawal, R. Srikant, "*Fast algorithms for mining association rules*", In: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, September 12-15, 1994, pp. 1-32.
7. R. Agrawal and R. Srikant, "*Privacy-preserving data mining*", In: Proceedings of ACM SIGMOD International Conference on Management of Data, May 2000, pp. 439-450.
8. D. Agrawal and C.C. Agrawal, "*On the design and quantification of privacy preserving data mining algorithms*", In: Proceedings 20th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, May 2001, pp. 247-255.
9. S.J. Rizvi and J.R. Haritsa, "*Maintaining data privacy in association rule mining*", In Proceedings 28th VLDB Conference, Hong Kong, China, 2002, pp. 1-12.
10. A. Prodomidis, P. Chan, S. Stofo, "*Meta-learning in Distributed Data Mining Systems: Issues and Approaches*", AAAI/MIT Press, 2000 (Chapter 3).
11. J. Vadya and C. Clifton, "*Privacy Preserving Association Rule Mining in Vertically Partitioned Data*", In: Proceedings of SIGKDD 2002, Edmonton, Alberta, Canada, 2002, pp. 639-645.

12. D.W.-L. Cheung, V. Ng, A.W.-C. Fu, and Y. Fu, "*Efficient mining of association rules in distributed databases*", IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996), pp. 911-922.
13. M. Kantarcioglu and C. Clifton. "*Privacy-preserving distributed mining of association rules in horizontally partitioned data*", In: Proceedings of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD, September 2004 (Vol. 16, No. 9), pp. 1026-1037.
14. M. Kantarcioglu and J. Vaidya, "*An architecture for privacy-preserving mining of client information*", IEEE ICDM Workshop on Privacy, Security and Data Mining, Volume 14, Maebashi, Japan, December 2002, pp. 37 - 42.
15. A.C. Yao, "*How to generate and exchange secrets*", In: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 162-167.
16. O. Goldreich, S. Micali, A. Wigderson, "*How to play any mental game—a completeness theorem for protocols with honest majority*", in: Proceedings, 19th ACM Symposium on the Theory of Computing, 1987, pp. 218-229.
17. Du, M.J. Atallah, "*Secure multi-party computational geometry*", In: Proceedings of the 7th International Workshop on Algorithms and Data Structures, Providence, Rhode Island, Jan 2001, pp. 165-179.
18. J. Vaidya, C. Clifton, "*Secure set intersection cardinality with application to association rule mining*", Journal of Computer Security, Volume 13, April 2005, pp. 593-622.
19. Boris Rozenberg and Ehud Gudes, "*Association rule mining in vertically partitioned databases*", Data & Knowledge Engineering, Volume 59, Issue 2, November 2006, pp. 378-396.
20. R. Agrawal, T. Imielinski, and A.M. Swami, "*Mining association rules between sets of items in large databases*", In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, 1993, pp. 207-216.
21. B. Rozenberg and E. Gudes, "*Collaborative privacy preserving frequent itemset mining in vertically partitioned databases*", In: Proceedings IFIP WG11.3 International Conference on Data and application security, Estes Park, Colorado, 2003, pp.91-104.

22. S. Brin, R. Motwani, J.D. Ullman, S. Tsur, "*Dynamic Itemset Counting and Implication Rules for Market Basket Data*", SIGMOD Record, Volume 6, Number 2: New York, June 1997, pp. 255 - 264.

LIST OF PUBLICATIONS

- [1] **Susheela Dahiya**, Durga Toshniwal, “Privacy Preserving Data Mining Using Reconstruction-Based Techniques”, Accepted in AICTE Sponsored National Seminar On “Enterprise Information Systems”.

APPENDIX A: SOURCE CODE LISTING

DISTORT.C

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>

void main()
{
    FILE *fp;
    FILE *fw;
    float temp;
    int i=0,j=0, pos = 1;
    int count=0, count1=0;
    static int a[100];
    char ch;
    int ch1;
    static int row=0, col=0;
    int total,total1;
    struct time t1,t2;
    void timediff( struct time t1, struct time t2);

    clrscr();

    pos=1;
    fp = fopen("test.txt","r");
    fw = fopen("result.txt","w");
    if(fp == NULL)
    {
        printf("cannot open the file\n");
        exit(1);
    }

    if(fw == NULL)
    {
        printf("cannot open the file\n");
        exit(1);
    }
    gettimeofday(&t1);
```

```
    srand(20);
```

```
    while(total>0)
```

```
    {
```

```
        temp=(int)rand()%10;
```

```
        printf("\t%d",temp);
```

```
        a[i++]=temp/9.0;
```

```
        total--;
```

```
    }
```

```
    for(i=0;i<(row*col);i++)
```

```
    {
```

```
        int k, temp1=1;
```

```
        if(a[i]==1)
```

```
        {
```

```
            int temp=0;
```

```
            count1++;
```

```
            if(count1>total1)
```

```
                a[i]=temp;
```

```
        }
```

```
        k=total1-count1;
```

```
        while(count1<total1)
```

```
        {
```

```
            a[k++]=temp1;
```

```
        }
```

```
    }
```

```
    while((ch = fgetc(fp)) != EOF)
```

```
    {
```

```
        count++;
```

```
        if(count>4 && ch !='\n')
```

```
        {
```

```
            ch1=a[j];
```

```
            printf(" %d\t",a[j]);
```

```
            j++;
```

```
            ch = ch^ch1;
```

```
            fprintf(fw,"%c", ch);
```

```
            pos++;
```

```
        }
```

```
    } else
```

```
    {
```

```
        ch = ch^0;
```

```
        fprintf(fw,"%c", ch);
```

```
        pos++;
```

```

    }
}

if(ch == '\n')
{
    fprintf(fw, "\n");
    pos = 1;
    count=0;
}
if(count<=3)
    fprintf(fw, "%c",ch);
}

fclose(fp);
fclose(fw);
gettime(&t2);
getch();
}
void timediff(struct time t1, struct time t2)
{
    struct time t3;
    t3.ti_hour= t2.ti_hour - t1.ti_hour;
    t3.ti_min= t2.ti_min - t1.ti_min;
    t3.ti_sec= t2.ti_sec - t1.ti_sec;
    t3.ti_hund= t2.ti_hund - t1.ti_hund;

    printf("\n %02d:%02d:%02d:%02d", t3.ti_hour,t3.ti_min,t3.ti_sec,t3.ti_hund);
}

```

INTERSECT.C

```

#include<stdio.h>
#include<conio.h>
#include<dos.h>

```

```

void main()
{
    FILE *f1;
    FILE *f2;
    int a1[10];
    int a2[10];
    int n1;

```

```
int n2;
char ch1,ch2;
int count=0;
int min_sup;
int i=0;
int j=0;
struct time t1,t2;
void timediff( struct time t1, struct time t2);
```

```
clrscr();
```

```
f1 = fopen("test.txt","r");
f2 = fopen("testt.txt","r");
```

```
if((f1==NULL) || (f2==NULL))
```

```
{
```

```
    printf("cannot open the file\n");
    exit(1);
```

```
}
```

```
gettime(&t1);
```

```
while((ch1=fgetc(f1))!=EOF)
```

```
{
```

```
    a1[i++] = atoi(&ch1);
    while((ch1=fgetc(f1))!='\n')
```

```
{
```

```
    if(ch1 == EOF)
```

```
        break;
```

```
}
```

```
    if(ch1 == EOF)
```

```
        break;
```

```
}
```

```
while((ch2=fgetc(f2))!=EOF)
```

```
{
```

```
    a2[j++] = atoi(&ch2);
    while((ch2=fgetc(f2))!='\n')
```

```
{
```

```
    if(ch2 == EOF)
```

```
        break;
```

```
}
```

```
    if(ch2==EOF)
```

```
        break;
```

```
}
```

```
n1=i;
```

```
n2=j;
```

```

for(i=0;i<n1;i++)
{
    for(j=0;j<n2;j++)
    {
        if(a1[i]==a2[j])
        {
            count++;
        }
    }
}
gettime(&t2);
printf("\n Enter the Minimum support : ");
scanf("%d",&min_sup);

if(min_sup<=count)
    printf("\n OK, Mining is possible");
else
    printf("\n NotOK, Mining is not possible");

printf("\n Intersection is : %d ", count);

getch();
}
void timediff(struct time t1, struct time t2)
{
    struct time t3;
    t3.ti_hour= t2.ti_hour - t1.ti_hour;
    t3.ti_min= t2.ti_min - t1.ti_min;
    t3.ti_sec= t2.ti_sec - t1.ti_sec;
    t3.ti_hund= t2.ti_hund - t1.ti_hund;

    printf("\n %02d:%02d:%02d:%02d", t3.ti_hour,t3.ti_min,t3.ti_sec,t3.ti_hund);
}

```

JOIN.C

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>

```



```

void main()
{
FILE *f1;
FILE *f2;
FILE *f3;
float temp = 0;
int pos = 1;
struct time t1,t2;
void timediff( struct time t1, struct time t2);

```

```

char ch;

```

```

static char a[20][20];
static int a1[20];
static int b1[20];
static char b[20][20];
char ch1, ch2;
int count=0;
static int count1=0;
static int count2=0;
int ch3=0;
int ch4;
int x, y;
static int i=0, r=0, s=0, j=0, k=0, l=0;
static int m=0, n=0;
static int row1=0, row2=0;
clrscr();

```

```

f1=fopen("sample.txt","r");
f2=fopen("sample1.txt","r");
f3=fopen("union.txt","w");

```

```

if((f1==NULL) || (f2==NULL))
{
printf("cannot open the file.\n");
exit(1);
}
gettime(&t1);
while((ch1=fgetc(f1)) !=EOF)
{
if(ch1!='\n')
{

```



```

count1++;
a[i][j++]=ch1;
if(count1==1)
    a1[k++]=atoi(&ch1);
}
else
{
    x=count1;
    count1=0;
    i++;
    j=0;
    row1++;
}
}
while((ch2=fgetc(f2)) !=EOF)
{
    if(ch2!='\n')
    {
        count2++;
        b[r][s++]=ch2;
        if(count2==1)
        {
            b1[l++]=atoi(&ch2);
        }
    }
    else
    {
        y=count2;
        count2=0;
        r++;
        s=0;
        row2++;
    }
}
while(m<row1 && n<row2)
{
    if(a1[m]==b1[n])
    {
        fprintf(f3,"%c",a[m][0]);
        fprintf(f3," ");
        for(i=3;i<x;i++)
        {
            fprintf(f3,"%c",a[m][i]);
        }
    }
}

```

```

    for(j=3;j<y;j++)
        fprintf(f3,"%c",b[n][j]);
    fprintf(f3,"\n");
    m++;
    n++;
}

```

```

if((a1[m]<b1[n]) && a1[m] != NULL && b1[n] != NULL)
{
    fprintf(f3,"%c",a[m][0]);
    fprintf(f3," ");
    for(i=3;i<x;i++)
        fprintf(f3,"%c",a[m][i]);
    for(j=3;j<y;j++)
        fprintf(f3,"%d",ch3);
    fprintf(f3,"\n");
    m++;
}

```

```

if((a1[m]>b1[n]) && a1[m] != NULL && b1[n] != NULL)
{
    fprintf(f3,"%c",b[n][0]);
    fprintf(f3," ");
    for(i=3;i<x;i++)
        fprintf(f3,"%d",ch3);
    for(j=3;j<y;j++)
        fprintf(f3,"%c",b[n][j]);
    fprintf(f3,"\n");
    n++;
}
}

```

```

if(m>=row1 && n<=row2)
{
    fprintf(f3,"%c",b[n][0]);
    fprintf(f3," ");//printf("\n%d",a1[m]);

    for(i=3;i<x;i++)
        fprintf(f3,"%d",ch3);
    for(j=3;j<y;j++)
        fprintf(f3,"%c",b[n][j]);
    fprintf(f3,"\n");
    n++;
}

```

```

if(m<=row1 && n>=row2)
{
    fprintf(f3,"%c",a[m][0]);
    fprintf(f3," ");
    for(i=3;i<x;i++)
        fprintf(f3,"%c",a[m][i]);
    for(j=6;j<y;j++)
        fprintf(f3,"%d",ch3);
    fprintf(f3,"\n");
    m++;
}
fclose(f1);
fclose(f2);
fclose(f3);
gettime(&t2);
getch();
}
void timediff(struct time t1, struct time t2)
{
    struct time t3;
    t3.ti_hour= t2.ti_hour - t1.ti_hour;
    t3.ti_min= t2.ti_min - t1.ti_min;
    t3.ti_sec= t2.ti_sec - t1.ti_sec;
    t3.ti_hund= t2.ti_hund - t1.ti_hund;

    printf("\n %02d:%02d:%02d:%02d", t3.ti_hour,t3.ti_min,t3.ti_sec,t3.ti_hund);
}

```

Apriori

```

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Apriori extends JFrame implements ActionListener
{
    private BufferedReader fileInput;
    private JTextArea textArea;

```

```
private JButton openButton, minSupport, runButton;
```

```
protected class TTreeNode
```

```
{  
    protected int support = 0;  
    protected TTreeNode[] childRef = null;  
    protected TTreeNode() {}
```

```
    private TTreeNode(int sup)  
    {  
        support = sup;  
    }  
}
```

```
private TTreeNode[] startTtreeRef;  
private short[][] dataArray = null;
```

```
private static final double MIN_SUPPORT = 0.0;  
private static final double MAX_SUPPORT = 100.0;
```

```
private boolean inputFormatOkFlag = true;  
private boolean haveDataFlag = false;  
private boolean hasSupportFlag = false;  
private boolean nextLevelExists = true ;
```

```
private File fileName;  
private int numRows = 0;  
private int numCols = 0;  
private double support = 20.0;  
private double minSupportRows = 1.0;
```

```
public Apriori(String s)
```

```
{  
    super(s);
```

```
    Container container = getContentPane();  
    container.setBackground(Color.pink);  
    container.setLayout(new BorderLayout(5,5));
```

```
    runButton = new JButton("Run");  
    runButton.addActionListener(this);  
    runButton.setEnabled(false);
```

```
    openButton = new JButton("Open File");
```

```

openButton.addActionListener(this);

minSupport = new JButton("Add Min. Sup.");
minSupport.addActionListener(this);

JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(1,3));
buttonPanel.add(openButton);
buttonPanel.add(minSupport);
buttonPanel.add(runButton);
container.add(buttonPanel, BorderLayout.NORTH);

textArea = new JTextArea(40, 15);
textArea.setEditable(false);
container.add(new JScrollPane(textArea), BorderLayout.CENTER);

JPanel creditsPanel = new JPanel();
creditsPanel.setBackground(Color.pink);
creditsPanel.setLayout(new GridLayout(4,1));
Label creditLabel1 = new Label("IIT Roorkee " + "Privacy Preserving
Association Rule Mining");
Label creditLabel2 = new Label(" ");
Label creditLabel3 = new Label("Created by Susheela (May " + "2008)");

creditsPanel.add(creditLabel1);
creditsPanel.add(creditLabel2);
creditsPanel.add(creditLabel3);
container.add(creditsPanel, BorderLayout.SOUTH);
}

public void actionPerformed(ActionEvent event)
{
    if (event.getActionCommand().equals("Open File"))
        getFileName();
    if (event.getActionCommand().equals("Read File"))
        readFile();
    if (event.getActionCommand().equals("Add Min. Sup. "))
        addSupport();
    if (event.getActionCommand().equals("Run"))
        aprioriT();
}

```

```

/* ----- */
/*          APRIORI-T          */
/* ----- */

```

```
private void aprioriT()
```

```
{
    textArea.append("Apriori-T (Minimum support threshold = " + support +
"%)\n-----\n" + "Generating K=1 large
itemsets\n");
```

```

    minSupportRows = numRows*support/100.0;
    createTreeTopLevel();
    generateLevel2();
    createTreeLevelN();
    textArea.append("\n");
    outputFrequentSets();

```

```
}
```

```
protected void createTreeTopLevel()
```

```
{
    startTreeRef = new TTreeNode[numCols+1];
    for (int index=1;index<=numCols;index++)
        startTreeRef[index] = new TTreeNode();
```

```
    createTreeTopLevel2();
```

```
    pruneLevelN(startTreeRef,1);
```

```
}
```

```
protected void createTreeTopLevel2()
```

```
{
    for (int index1=0;index1<dataArray.length;index1++)
    {
        if (dataArray[index1] != null)
        {
            for (int index2=0;index2<dataArray[index1].length;index2++)
            {
                startTreeRef[dataArray[index1][index2]].support++;
            }
        }
    }
}
```

```

protected void createTtreeLevelN()
{
    int nextLevel=2;

    while (nextLevelExists)
    {
        textArea.append("Generating K=" + nextLevel + " large itemsets\n");
        addSupportToTtreeLevelN(nextLevel);
        pruneLevelN(startTtreeRef,nextLevel);
        nextLevelExists=false;
        generateLevelN(startTtreeRef,1,nextLevel,null);
        nextLevel++;
    }
}

```

```

protected void addSupportToTtreeLevelN(int level)
{
    for (int index=0;index<dataArray.length;index++)
    {
        if (dataArray[index] != null)
        {
            addSupportToTtreeFindLevel(startTtreeRef, level,
                dataArray[index].length, dataArray[index]);
        }
    }
}

```

```

private void addSupportToTtreeFindLevel(TreeNode[] linkRef, int level,
    int endIndex, short[] itemSet)
{
    if (level == 1)
    {
        for (int index1=0;index1 < endIndex;index1++)
        {
            if (linkRef[itemSet[index1]] != null)
            {
                linkRef[itemSet[index1]].support++;
            }
        }
    }
}

```



```

    }
    else
    {
        for (int index=0;index<endIndex;index++)
        {
            if (linkRef[itemSet[index]] != null)
            {
                if (linkRef[itemSet[index]].childRef != null)
                    addSupportToTtreeFindLevel(linkRef[itemSet[index]].
                    childRef, level-1,index,itemSet);
            }
        }
    }
}

```

```

/*-----*/
/*          PRUNING          */
/*-----*/

```

```

protected void pruneLevelN(TreeNode [] linkRef, int level)
{
    int size = linkRef.length;

    if (level == 1)
    {
        for (int index1=1;index1 < size;index1++)
        {
            if (linkRef[index1] != null)
            {
                if (linkRef[index1].support < minSupportRows)
                    linkRef[index1] = null;
            }
        }
    }
}

else
{
    for (int index1=1;index1 < size;index1++)
    {
        if (linkRef[index1] != null)
        {
            if (linkRef[index1].childRef != null)
                pruneLevelN(linkRef[index1].childRef,level-1);
        }
    }
}

```



```
protected void generateNextLevel(TreeNode[] parentRef, int endIndex,
    short[] itemSet)
```

```
{
    parentRef[endIndex].childRef = new TreeNode[endIndex];
    short[] newItemSet;
    TreeNode currentNode = parentRef[endIndex];

    for (int index=1;index<endIndex;index++)
    {
        if (parentRef[index] != null)
        {
            newItemSet = realloc2(itemSet,(short) index);
            if (testCombinations(newItemSet))
            {
                currentNode.childRef[index] = new TreeNode();
                nextLevelExists=true;
            }
        }
        else
            currentNode.childRef[index] = null;
    }
}
```

```
protected boolean testCombinations(short[] currentItemSet)
```

```
{
    if (currentItemSet.length < 3)
        return(true);

    short[] itemSet1 = new short[2];
    itemSet1[0] = currentItemSet[1];
    itemSet1[1] = currentItemSet[0];

    int size = currentItemSet.length-2;
    short[] itemSet2 = removeFirstNelements(currentItemSet,2);

    return(combinations(null,0,2,itemSet1,itemSet2));
}
```



```

    {
        if (linkRef[itemSet[index]] != null)
        {
            if (index == lastIndex)
                return(true);
            else
                return(findItemSetInTtree2(itemSet,index+1,lastIndex,
                    linkRef[itemSet[index]].childRef));
        }
        else
            return(false);
    }
}

/* ----- */
/*           GET MINIMUM SUPPORT VALUE           */
/* ----- */

private void addSupport()
{
    try
    {
        while (true)
        {
            String stNum1 = JOptionPane.showInputDialog("Input minimum " +
                " support value between " + MIN_SUPPORT + " and " +
                MAX_SUPPORT);

            if (stNum1.indexOf('.') > 0)
                support = Double.parseDouble(stNum1);
            else
                support = Integer.parseInt(stNum1);
            if (support >= MIN_SUPPORT && support <= MAX_SUPPORT)
                break;
            JOptionPane.showMessageDialog(null,
                "MINIMUM SUPPORT VALUE INPUT ERROR:\n" +
                "input = " + support +
                "\nminimum support input must be a floating point\n" +
                "number between " + MIN_SUPPORT + " and " +
                MAX_SUPPORT);
        }
        textArea.append("Minimum support = " + support + "%\n");
        hasSupportFlag=true;
    }
    catch(NumberFormatException e)
    {
    }
}

```

```

        hasSupportFlag=false;
        runButton.setEnabled(false);
    }

    if (haveDataFlag && hasSupportFlag)
        runButton.setEnabled(true);
}

/* ----- */
/*          OPEN NAME          */
/* ----- */

private void getFileName()
{
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setSelectionMode(JFileChooser.FILES_ONLY);
    int result = fileChooser.showOpenDialog(this);

    if (result == JFileChooser.CANCEL_OPTION) return;

    fileName = fileChooser.getSelectedFile();
    if (checkFileName())
    {
        readFile();
    }

    if (inputFormatOkFlag)
    {
        if (checkOrdering())
        {
            if (haveDataFlag && hasSupportFlag)
                runButton.setEnabled(true);
            outputDataArray();
            textArea.append("Number of records = " + numRows + "\n");
            countNumCols();
            textArea.append("Number of columns = " + numCols + "\n");
        }
        else
        {
            haveDataFlag = false;
            inputFormatOkFlag = true;
            textArea.append("Error reading file: " + fileName + "\n\n");
            runButton.setEnabled(false);
        }
    }
}

```

```
    }  
}  
  
private boolean checkFileName()  
{
```

```
    if (fileName.exists())  
    {
```

```
        if (fileName.canRead())  
        {
```

```
            if (fileName.isFile())
```

```
                return(true);
```

```
            else
```

```
                JOptionPane.showMessageDialog(null,"FILE ERROR: File is a  
                directory");
```

```
        }  
    }  
    else
```

```
        JOptionPane.showMessageDialog(null, "FILEERROR: Access denied");
```

```
    }  
    else
```

```
        JOptionPane.showMessageDialog(null, "FILE ERROR: No such file!");
```

```
    return(false);  
}
```

```
  
private void readFile()  
{
```

```
    try
```

```
    {
```

```
        inputFormatOkFlag=true;
```

```
        getNumberOfLines();
```

```
        if (inputFormatOkFlag)
```

```
        {
```

```
            dataArray = new short[numRows][];
```

```
            inputDataSet();
```

```
            haveDataFlag = true;
```

```
        }  
    }  
    else
```

```
    {
```

```
        haveDataFlag = false;
```

```
        textArea.append("Error reading file: " + fileName + "\n\n");
```

```
        runButton.setEnabled(false);
```

```
    }  
}
```

```

catch(IOException ioException)
{
    JOptionPane.showMessageDialog(this,"Error reading File",
        "Error 5: ",JOptionPane.ERROR_MESSAGE);
    closeFile();
    System.exit(1);
}
}

private void getNumberOfLines() throws IOException
{
    int counter = 0;
    openFile();

    String line = fileInput.readLine();
    while (line != null)
    {
        checkLine(counter+1,line);
        StringTokenizer dataLine = new StringTokenizer(line);
        int numberOfTokens = dataLine.countTokens();
        if (numberOfTokens == 0) break;
        counter++;
        line = fileInput.readLine();
    }

    numRows = counter;
    closeFile();
}

private void checkLine(int counter, String str)
{
    for (int index=0;index <str.length();index++)
    {
        if (!Character.isDigit(str.charAt(index)) &&
            !Character.isWhitespace(str.charAt(index)))
        {
            JOptionPane.showMessageDialog(null,"FILE INPUT ERROR:\ncharcater " +
                "on line " + counter + " is not a digit or white space");
            inputFormatOkFlag = false;
            break;
        }
    }
}
}

```



```

public void inputDataSet() throws IOException
{
    int rowIndex=0;
    textArea.append("Reading input file\n" + fileName + "\n");
    openFile();

    String line = fileInput.readLine();
    while (line != null)
    {
        StringTokenizer dataLine = new StringTokenizer(line);
        int numberOfTokens = dataLine.countTokens();
        if (numberOfTokens == 0) break;
        short[] code = binConversion(dataLine,numberOfTokens);
        if (code != null)
        {
            int codeLength = code.length;
            dataArray[rowIndex] = new short[codeLength];
            for (int colIndex=0;colIndex<codeLength;colIndex++)
                dataArray[rowIndex][colIndex] = code[colIndex];
        }
        else
            dataArray[rowIndex]= null;
        rowIndex++;
        line = fileInput.readLine();
    }

    closeFile();
}

```

```

private short[] binConversion(StringTokenizer dataLine, int numberOfTokens)
{
    short number;
    short[] newItemSet = null;
    for (int tokenCounter=0;tokenCounter < numberOfTokens;tokenCounter++)
    {
        number = new Short(dataLine.nextToken()).shortValue();
        newItemSet = realloc(newItemSet,number);
    }
    return(newItemSet);
}

```

```

private boolean checkOrdering()
{

```

```

boolean result = true;
for(int index=0;index<dataArray.length;index++)
{
    if (!checkLineOrdering(index+1,dataArray[index]))
        result=false;
}
return(result);
}
private boolean checkLineOrdering(int lineNum, short[] itemSet)
{
    for (int index=0;index<itemSet.length-1;index++)
    {
        if (itemSet[index] >= itemSet[index+1])
        {
            JOptionPane.showMessageDialog(null,"FILE FORMAT ERROR:\n" +
                "Attribute data in line " + lineNum + " not in numeric order");
            return(false);
        }
    }

    return(true);
}

private void countNumCols()
{
    int maxAttribute=0;
    for(int index=0;index<dataArray.length;index++)
    {
        int lastIndex = dataArray[index].length-1;
        if (dataArray[index][lastIndex] > maxAttribute)
            maxAttribute = dataArray[index][lastIndex];
    }
    numCols = maxAttribute;
}

/* ----- */
/*          OUTPUT METHODS          */
/* ----- */

public void outputDataArray()
{
    for(int index=0;index<dataArray.length;index++)
    {
        outputItemSet(dataArray[index]);
    }
}

```

```

        textArea.append("\n");
    }
}

protected void outputItemSet(short[] itemSet)
{
    String itemSetStr = " {";
    int counter = 0;
    for (int index=0;index<itemSet.length;index++)
    {
        if (counter != 0) itemSetStr = itemSetStr + ",";
        counter++;
        itemSetStr = itemSetStr + itemSet[index];
    }
    textArea.append(itemSetStr + "}");
}

public void outputFrequentSets()
{
    int number = 1;

    textArea.append("FREQUENT (LARGE) ITEM SETS (with support
counts)\n" +
        "-----\n");
    short[] itemSetSofar = new short[1];
    for (int index=1; index <= numCols; index++)
    {
        if (startTtreeRef[index] !=null)
        {
            if (startTtreeRef[index].support >= minSupportRows)
            {
                textArea.append("[ " + number + " ] { " + index + " } = " +
                    startTtreeRef[index].support + "\n");
                itemSetSofar[0] = (short) index;
                number = outputFrequentSets(number+1,itemSetSofar,
                    index,startTtreeRef[index].childRef);
            }
        }
    }

    textArea.append("\n");
}

```

```

private int outputFrequentSets(int number, short[] itemSetSofar, int size,
                               TTreeNode[] linkRef)
{
    if (linkRef == null)
        (number);
    for (int index=1; index < size; index++)
    {
        if (linkRef[index] != null)
        {
            if (linkRef[index].support >= minSupportRows)
            {
                short[] newItemSetSofar = realloc2(itemSetSofar, (short) index);
                textArea.append("[ " + number + " ] ");
                outputItemSet(newItemSetSofar);
                textArea.append(" = " + linkRef[index].support + "\n");
                number = outputFrequentSets(number + 1, newItemSetSofar,
                                           index, linkRef[index].childRef);
            }
        }
    }
}

```

```

return(number);
}

```

```

/* ----- */
/*      FILE HANDLING UTILITIES      */
/* ----- */

```

```

private void openFile()

```

```

{
    try
    {
        FileReader file = new FileReader(fileName);
        fileInput = new BufferedReader(file);
    }
    catch(IOException ioException)
    {
        JOptionPane.showMessageDialog(this, "Error Opening File",
                                     "Error 4: ", JOptionPane.ERROR_MESSAGE);
    }
}

```

```

private void closeFile()

```

```

{

```

```

if (fileInput != null)
{
    try
    {
        fileInput.close();
    }
    catch (IOException ioException)
    {
        JOptionPane.showMessageDialog(this,"Error Opening File",
            "Error 4: ",JOptionPane.ERROR_MESSAGE);
    }
}
}

/* ----- */
/*           ARM UTILITIES           */
/* ----- */

```

```

protected short[] realloc1(short[] oldItemSet, short newElement)
{
    if (oldItemSet == null)
    {
        short[] newItemSet = {newElement};
        return(newItemSet);
    }

    int oldItemSetLength = oldItemSet.length;
    short[] newItemSet = new short[oldItemSetLength+1];

    int index;
    for (index=0;index < oldItemSetLength;index++)
        newItemSet[index] = oldItemSet[index];
    newItemSet[index] = newElement;

    return(newItemSet);
}

```

```

protected short[] append(short[] itemSet1, short[] itemSet2)
{
    if (itemSet1 == null)
        return(copyItemSet(itemSet2));
    else if (itemSet2 == null)
        return(copyItemSet(itemSet1));
}

```

```

short[] newItemSet = new short[itemSet1.length+itemSet2.length];

int index1;
for(index1=0;index1<itemSet1.length;index1++)
{
    newItemSet[index1]=itemSet1[index1];
}

for(int index2=0;index2<itemSet2.length;index2++)
{
    newItemSet[index1+index2]=itemSet2[index2];
}

return(newItemSet);
}

```

```

protected short[] realloc2(short[] oldItemSet, short newElement)
{
    if (oldItemSet == null)
    {
        short[] newItemSet = {newElement};
        return(newItemSet);
    }

    int oldItemSetLength = oldItemSet.length;
    short[] newItemSet = new short[oldItemSetLength+1];

    newItemSet[0] = newElement;
    for (int index=0;index < oldItemSetLength;index++)
        newItemSet[index+1] = oldItemSet[index];

    return(newItemSet);
}

```

```

protected short[] removeFirstNelements(short[] oldItemSet, int n)
{
    if (oldItemSet.length == n) return(null);
    else
    {
        short[] newItemSet = new short[oldItemSet.length-n];
        for (int index=0;index<newItemSet.length;index++)
        {

```

```
        newItemSet[index] = oldItemSet[index+n];
    }
    return(newItemSet);
}
}
```

```
protected short[] copyItemSet(short[] itemSet)
{
    If (itemSet == null)
        return(null);

    short[] newItemSet = new short[itemSet.length];
    for(int index=0;index<itemSet.length;index++)
    {
        newItemSet[index] = itemSet[index];
    }
    return(newItemSet);
}
```

```
/* ----- */
/*           MAIN METHOD           */
/* ----- */
```

```
public static void main(String[] args) throws IOException
{
    Apriori newFile = new Apriori("Apriori");
    newFile.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    newFile.setSize(500,800);
    newFile.setVisible(true);
}
}
```