T.C

İNÖNÜ UNIVERSITY

Institution of Natural and Applied Sciences

PRIVACY PRESERVING DATA MINING

Afrah Najib FAREA

Master Thesis

Computer Engineering Department

Malatya

JUNE 2015

Tezin başlığı: "Privacy Preserving Data Mining"

Tezi Hazırlayan: Afrah Farea

Sınav tarihi: 30.07.2015

Yukarıda adı geçen tez jürimizce değerlendirilerek Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans olarak kabul edilmiştir.

Sınav Jürisi Üyeleri	[illegible]
Prof. Dr. Ali KARCI	İnönü Üniversitesi
(Danışman)	[illegible]
Doç. Dr. Davut Hanbay	İnönü Üniversitesi
(Üye)	[illegible]
Doç. Dr. Ahmet Bedri ÖZER	Fırat Üniversitesi

İnönü Üniversitesi Fen Bilimleri Enstitü Onayı

Prof. Dr. Alaattin ESEN

Enstitü Müdürü

İnönü University Honor Code Pledge

Afrah FAREA

ONUR SÖZÜ

Yüksek Lisans tezi olarak sunduğum “Privacy Preserving Data Mining” başlıklı bu çalışmanın bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurmaksızın tarafımdan yazıldığını ve yararlandığım bütün kaynakların, hem metin içinde hem de kaynakçada yöntemine uygun biçimde gösterilenlerden oluştuğunu belirtir, bunu onurumla doğrularım.

Afrah FAREA

ABSTRACT

Thesis for Master

Privacy Preserving Data Mining

Afrah Farea

İnönü University
Institution of Natural and Applied sciences
Computer Engineering Department

65 + x pages

2015

Supervisor of: Prof. Dr. Ali KARCI

Data Mining allows large database owners to share and extract useful knowledge that could not be deduced with traditional approaches like statistics. However, these sometimes reveal sensitive knowledge or breach individuals' privacy. The term sanitization is given to the process of changing original database into another one from which we can mine without exposing sensitive knowledge. This process should be guided by little distortion on the database. In this dissertation, we address these issues in a data mining branch called Privacy Preserving Data Mining. In particular, we focus on association rule hiding (ARH) and evaluate the heuristic approaches for this purpose. We also apply these heuristic approaches on a number of publically available datasets and examine the results.

Keywords: Data mining, Privacy preserving data mining, Association rules hiding

ÖZET

Yüksek Lisans Tezi

Kişisel Bilgilerin Gizlenmesi Veri Madenciliği

Afrah Farea

İnönü Üniversitesi
Fen Bilimler Enstitüsü
Bilgisayar Mühendisliği Bölümü

65 + x sayfa

2015

Danışman: Prof. Dr. Ali KARCI

Veri Madenciliği kendi büyük veri tabanını paylaşırken istatistik gibi geleneksel yaklaşımlarla elde edilemeyen yararlı bilgileri elde etmeye denir. Ancak, bu bazen hassas bilgileri açığa çıkarır veya bireysel kişisel gizlilik aşikâr eder. Sanitization terimi veri tabanının değiştirilmesi ve yeni bir veri tabanı elde edilerek bu veri tabanı üzerinde yapılan veri madenciliği işlemleri hassas veriyi ortaya çıkarmama işlemine verilen isimdir. Bu işlem veri tabanı üzerinde çok az bozulma ile yönlendirilmelidir. Bu tez çalışmasında, Kişisel Bilgilerin Gizlenmesi Veri Madenciliği üzerine çalışmalar yapılmıştır. Özellikle, Birliktelik Kural Gizleme üzerine odaklanıldı ve sezgisel yaklaşımlarla değerlendirmeler yapıldı. Ayrıca kamuya açık veri kümelerinde bir dizi bu sezgisel yaklaşımları uygulandı ve sonuçlar değerlendirildi.

Anahtar Kelimeler: Veri madenciliği, Kişisel Bilgilerin Gizlenmesi Veri Madenciliği, Birliktelik Kural Gizleme.

ACKNOWLEDGEMENTS

I would like to thank you Malatya for your beauty and kindliness.

I would like to thank my supervisor, Prof.Dr.Ali KARCI for his patience, guidance, and support. In many times, I used to talk to him in a language that is neither English, nor Turkish or Arabic but every time he understands what I want to mean.

I would like to thank Doç. Dr. Celaleddin YEROĞLU, the head of the department, Doç. Dr. Davut HANBAY, Yrd. Doç. Dr. Fatih KOCAMAZ, Yrd. Doç. Muhammed Fatih TALU, and all the academic personals in Computer Engineering department, İnönü university for their help and support.

Many thanks to my family in Yemen who are facing hard times because of the war.

Many thanks to my husband for his infinite support and patience.

Many thanks to my friends and relatives.

Last but not least, many thanks to Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı ,YTB, for this wonderful experience.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBRIVIATIONS

ARH	:	Association Rule Hiding
PPDM	:	Privacy Preserving Data Mining
KDD	:	Knowledge Discovery In Data Mining
OLAP	:	Online Analytical Processing
CART	:	Classification And Regression Trees
ID3	:	Iterative Dichotomiser 3
SPADE	:	Sequential Pattern Discovery Using Equivalence Classes
GSP	:	Generalized Sequential Patterns
CLARA	:	Clustering For Large Applications
CLARANS	:	Clustering Algorithm Based On Randomized Search
BIRCH	:	Balanced Iterative Reducing And Clustering Using Hierarchies
RAM	:	Random Access Memory
DBMS	:	Data Base Management Systems
IGA	:	Item Grouping Algorithm

CHAPTER 1: INTRODUCTION

1.1 Motivation

Advances in Computer hardware and software, in particular database system technologies, allow collecting data at dramatic pace. These collected data are undoubtedly useful for many applications such as scientific researches, business investments, law enforcements, national and international securities, etc. In the same direction, with the help of machine learning algorithms, data mining emerges as a technology that uses vast amount of data to generate hypotheses and discover general patterns. However, this raises concerns about individual/organizational privacies in two scopes: data and knowledge [12]. In terms of data privacy, we mean that when data are collected in plain texts and stored in centralized warehouses, they become more prone to attack and exposure especially medical and criminal records. Similarly, knowledge privacy means preserving knowledge that is inferred from data including data mining tools.

Thus with these tradeoffs, there must be an appropriate balance between preserving privacy and utilizing knowledge discovered by data mining tools. Privacy Preserving Data Mining (PPDM) is a new field in Data Mining discipline that concerns with developing algorithms to fulfill these aims.

1.2 Knowledge Discovery and Data Mining

The term Data Mining is sometimes referred to as Knowledge Discovery in Data Mining (KDD). Others consider it as a step in the ‘Knowledge Discovery Process (KDD)’, albeit the most important one. In general, this process has several main steps, as can be seen in figure in 1.1 [18, 20].

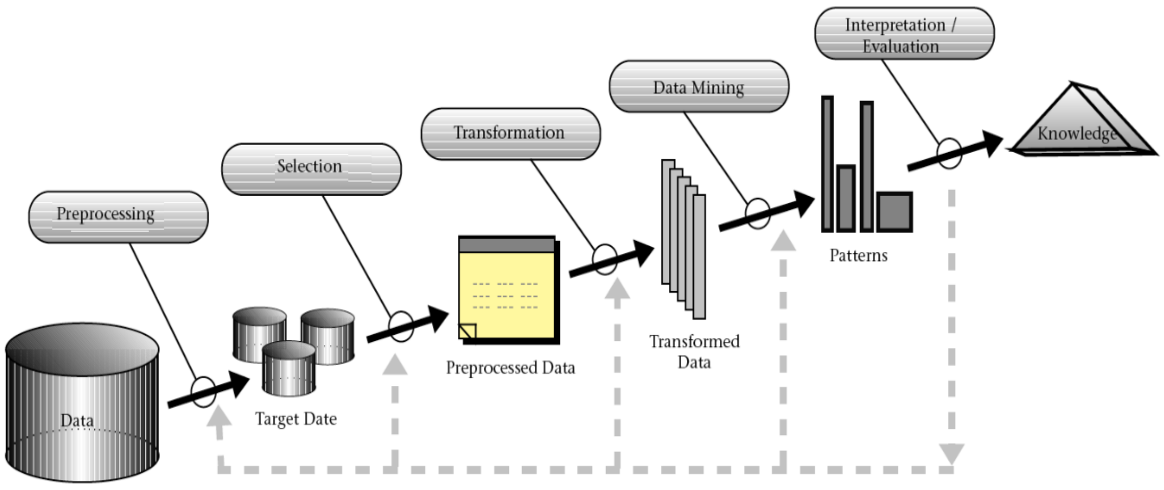


Figure 1.1 KDD process [43]

The major steps in the KDD process can be defined as follows:

- **Data Cleaning:** Real data usually contains noise, duplicate, missing, or inconsistent values that should be removed before any further processing.
- **Data Integration:** when data from different sources are combined.
- **Data Selection:** This could mean selecting data relevant to the task. It could also mean sampling data; selecting some features using subset selection and feature extraction techniques.
- **Data Transformation:** changing data into another form suitable for mining for example changing attributes from nominal into numeric, binary or vice versa. It could also mean combining attributes, splitting them or swapping their values.
- **Data Mining:** is the main step in knowledge discovery process in which data mining analyzes data to get hypotheses and patterns.
- **Pattern Evaluation:** Not all patterns generated by data mining are important (interesting). There are measures to get significant patterns only such as support and confidence in association rules mining.
- **Knowledge presentation**: results are shown using visualization tools such as bar chats, pie chart, etc.

1.3 Data mining vs Statistics

Both Data Mining and Statistics concern with learning from data. So what are their differences and relationships? Statisticians claim that the term data mining is synonymous with “data dredging” or “data fishing”. These terms describe the process of trawling data in the hope of finding patterns [14]. However, data mining has broader connotations that set at the boundaries of Artificial Intelligence, Pattern Recognition, Statistics, and Machine Learning. It tends to be more practical than Statistics due to its origin. The main part of Data

Mining involves the analysis of data and the application of Machine Learning algorithms to find useful patterns. On the other hand, Statistics includes everything about data from data collection up to data visualization [22]. Phases of statistics include experimental design & sampling, Exploratory Data Analysis, Statistical Graphics, Statistical Modeling, and Statistical Inference [15].

As stated in [18], databases in data mining are usually very large in terms of records and number of variables. However, databases in statistics are gathered from a specific domain that fulfills some objectives. This makes data used in statistics smaller, less noisy, and more object-oriented. On the other hand, data mining is called secondary data analysis because data is collected for some other purpose and then sent to data mining for further analysis.

Cerrito in [8] gives a set of points why data mining is different from Statistics. First, data mining deals with heterogonous data from different sources .These data could be texts, images, signals, audios, videos that have complex formats and structures. Second, data mining applies preprocessing techniques on the raw datasets, which might have more influence on the results than the data mining tools selected. Third, Data mining algorithms should be scalable and practical and some of them (such as Neural Networks, Support Vector Machines, Decision Trees, etc.) have stronger mathematical foundations than statistical justifications. Besides, data mining techniques often transform input data into more abstract forms by performing systematic compressions of data input. In addition, data mining seeks for finding local and global models as opposed to global models in Statistics.

Finally, Aggarwal [1] states that statistical approaches are mathematically more precise however, suffer from simplified assumptions about data representations, poor algorithmic scalability, and a low focus on interpretability.

1.4 Data Mining Functionalities

Researchers usually classify data Mining tasks into two categories: descriptive and predictive**:**

- **Descriptive mining tasks** are used to completely describe data, like clustering, segmentation, variables relationships, etc.
- **Predictive mining tasks** are used for making predictions from data, such as Classification and Regression.
- **Exploratory Data Analysis (EDA) tasks** are used for visualizing and exploring data without clear idea of what to look for.
- **Discovering patterns and Rules**: are used for finding common items that frequently occur together using association rules and itemsets algorithms. Outlier detection also comes under this category.

- **Retrieval by Content**. Given a pattern (usually a text or an image), system is required to find similar patterns in the data set.

Data mining functionalities, and the kinds of patterns they can discover, are described below [20].

- **Concept/Class Description: Characterization and Discrimination**

  Data can be associated with concrete classes or abstract concepts. It can be useful to describe individual classes and concepts in summarized, concise, and yet precise terms. Such descriptions of a class or a concept are called class/concept descriptions. These descriptions can be derived via

  - *data characterization*, by summarizing the data of the class under study .There are several summarization methods such as OLAP for user-controlled data summarization. Results of summarization are shown in pie charts, bar charts, etc.
  - *data discrimination,* by comparing the target class with one or a set of comparative classes, or
  - Both data characterization and discrimination.

- **Mining Frequent Patterns, Associations, and Correlations**

  Association analysis is useful for finding hidden relations between elements in the datasets. An itemset consists of one or more items in the dataset. Association rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets, i.e. $X \cap Y = \emptyset$. There are two important measures that show the strength of an association rule Support and Confidence. Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in **Y** appear in transactions that contain **X**. The formal definitions of these metrics are:

  $$\textbf{Support, } s(X \rightarrow Y) = \sigma(X \cup Y) / N$$

  $$\textbf{Confidence, } c(X \rightarrow Y) = \sigma(X \cup Y) / \sigma(X).$$

  These measures are called interesting measures. An itemset satisfying support threshold is called frequent itemset. If an itemset is frequent, then all of its subsets are frequent (this property is called **Apriori Principle**). Similarly, a rule that satisfies confidence threshold is called strong or interesting rule. Uninteresting rules are eliminated.

  A common strategy adopted by many Association Rule algorithms to solve the problem is to divide it into two subtasks: First finding frequent itemsets. Second from frequent itemsets found in the previous step, we find the strong rules. We elaborate further on association rules, as it is the measure concern of this thesis.

- **Classification and Prediction**

Classification is the task of assigning objects to one of several predefined classes. Regression is a predictive modeling task in which the class is a continuous attribute.

Decision tree induction uses divide-and-conquer partitioning strategy to build a tree-like model. Thus, we have a root node, which has no parents, intermediate nodes, which have both incoming (parents), and outgoing edges (children) and leaf nodes which have no children. Each leaf node is assigned a class label. Many measures are used to decide the best way to split at each node. The most common ones are entropy, gini index, and classification error. Redundant or insignificant attributes may affect results of decision trees or make them larger. Therefore, attribute selection should be used before classification. Decision Trees are usually simple, easy to understand, efficient, fast, and robust to the presence of noise. This make them widely used. Examples of Classification Tree Induction algorithms are ID3, C4.5, and CART.

Rule-based classifiers (also called Covering Algorithms) construct a model by frequently creating rules that cover the largest possible number of instances. There are two approaches for rule-based classifier: extracting rules either directly from data, or indirectly from other classification approaches such as Neural Networks and Decision Tree induction.

K-Nearest Neighbors classifier is an example of Lazy **Learners** where classification is postponed until it is needed to classify the test instance. This classifier represents each example, as a data point in a d-dimensional space, where d is the number of attributes. Given a test example, we compute its proximity to the rest of the data points in the training set, using one of the proximity measures such as Euclidean distance. Then the test instance takes the label of the majority in the neighboring K instances.

When the relationship between the attribute set X and the class variable Y is not clear due to noise or some confounding variables, we can model such datasets with help of Bayesian probability. We treat X and Y as random variables and capture their relationships using P (Y|X) probability. This type of classification is known as **Bayesian Classifier**.

Artificial Neural Networks (**ANN**) is one of the old & powerful classification algorithms, which is based on biological neural system imitation. Every attribute has a corresponding small random weight that is frequently updated according to minimum total sum of squared errors. Weight updates continue until convergence. The measure defect of ANN is that they are very slow especially when the number of hidden layers is large. Besides, they are quite sensitive to noise.

Support Vector Machines (SVM) receives considerable attention especially in pattern recognition and text categorization. This technique has its roots in statistical learning theory and represents the decision boundary using a subset of the training examples, known as the **support vectors**.

Bagging (Bootstrap aggregating): this and the following algorithms are called ensemble algorithms because they use more than one cluster at a time for classification. In this algorithm, **K** samples of training instances are repeatedly chosen (with replacement) from the original database according to a uniform probability distribution. These samples are called bootstrap replicate and each bootstrap sample has the same size as the original data. After training the samples using any base classifier, the test instance gets the label of the class with the highest vote.

Boosting maintains a set of weights for each training instance by initially giving them equal weights and then update them repeatedly according to the error rate of the base classifier. A sample from instances is chosen depending on the higher weights at every iteration. Sometimes Boosting works better than Bagging but it also tends to overfit the training data [11].

Random Forest aggregates results of a number of decision trees. Each tree gets a set of instances selected randomly with replacement. At each node, L attributes are randomly selected as candidates for split at that node. Trees grow to the maximum size without pruning and the CART methodology is used to expand the tree. During classification, each tree votes a class. Finally, the most popular class is returned. Random Forests are powerful classification algorithms and they are robust to noise and outliers [20].

Stacking (or stacked generalization) uses a different idea to combine classifiers. First, we divide original data into training and testing. We train the classifiers with the training data and test the generated models with the test data. In the next level, we used the predictions on the previous step as inputs and the desired responses as outputs.

- **Sequence Pattern Mining**

A sequence is an ordered set of items that appear together with repetition allowed. Sequential pattern mining examines finding frequent subsequences as patterns in sequential databases [24]. Sequential database consists of a set of sequential records (called sequences); each record has an ordered set of events (or itemsets). Thus a sequence S = < a (be) c (ad)> consists of four events or itemsets. Common sequences are biological sequences such as DNA sequence, Protein sequence and amino-acid sequence.

Similar to frequent itemsets mining, a sequence is considered frequent if its frequency is greater than or equal to a certain threshold value called minimum support threshold. In addition, the concepts of frequent closed sequence and frequent maximal sequence are also similar. That is, a frequent sequence is called maximal frequent sequence if there is no immediate frequent super-sequence and is called frequent closed sequence if there is no super-sequence with the same support. [24] states that frequent sequential pattern discovery can be thought of as association rules discovery over temporal datasets in that association rule mining only covers the intra-transaction itemsets while sequential pattern mining also investigates the inter-transaction patterns where ordering of items and itemsets are very important. Sequence Pattern Mining gains an increasing importance with the increase of using World Wide Web in E-commerce businesses and web services. Algorithms for mining sequential patterns are categorized into Apriori-based algorithms like SPADE, AprioriAll, GSP, pattern growth-based algorithms like PLWAP and PrefixSpan. The third category is early pruning-based algorithms like HVSM and DISC-all [24, 37].

- **Clustering and Segmentation Analysis**

Cluster analysis divides data into meaningful groups (clusters) in which objects of the same group are similar. The objective is to maximize the intra-class similarity between objects of the same cluster and to minimize the inter-class dissimilarity between objects of different clusters. Clustering is used as a data analysis tool or as a preprocessing step for other data mining models like classification. It can also be used for data summarization, data compression, dimensionality reduction, and anomaly detection.

There are many clustering algorithms. However, most of them can be classified into one of the following categories:

- **Partition-based approaches:** the simplest way of clustering is to divide dataset into **K** exclusive partitions. Partitioning is based on maximizing similarity between objects within the same partition and maximizing dissimilarity between objects of different partitions. The similarity measure used is usually distance. Examples of such algorithms are K-means, K-Medoids, CLARA, and CLARANS.
- **Hierarchical-based approaches** represent data as a hierarchy or a tree of clusters. There are two variations: Agglomerative approach (bottom-up) and Divisive approach (top-down). In the first variation, each data point is considered a cluster by itself and then we start growing each cluster by merging them with the nearest clusters until a single cluster remains. The other variation, divisive approach, starts by considering all points as a single whole cluster and then splitting them until only individual clusters of single points remain. Example of such approach is BIRCH algorithm.
- **Density-based approaches:** Density here means number of data points in the neighborhood. A cluster is constructed from a point if its neighboring points exceed predefined min-points within certain radius value. The cluster grows as long as this condition satisfies. This feature allows such algorithms to make non-spherical clustering shapes. Examples of this approach are DBSCAN and OPTICS.

- **Grid-based approaches** construct grid structure and then quantize data space into a finite number of cells. The main advantage of this method is that they are fast. Examples of these approaches are STING and CLIQUE.

Following are the most commonly clustering algorithms in literature:

- **K-means** is an iterative approach in which **K** number of points is selected to be initial cluster centroids, i.e. center of the clusters where **K** is a parameter defined by the user. In the next step, we assign every point a label of the nearest centroid. Then we update the cluster centroids to be the mean of the points within each cluster. This process is repeated until the centroids positions remain the same. The main limitation of this approach is its sensitivity to outliers. Besides, K-means convergence depends on initial centroid positions and that there must be criteria to find number of clusters **K**. To overcome these limitations, there are many extensions of K-means like K-modes, K-medoids, C-means, and K-means++ [23].
- **DBSCAN (**Density-Based Spatial Clustering of Application with Noise**)** is a simple and efficient density-based clustering algorithm. A point is called core point if number of its neighboring points are larger than (or equal to) min-points defined by the user with fixed radius value. A point is called border if it lies within the radius of any core point and is called a noise if it is neither core nor border. Core points lying within the radius of one other are grouped (along with their neighbors) under the same cluster. Noise points are discarded [13].
- **BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies) integrates hierarchal clustering with other clustering techniques by first building a "clustering-feature" tree. Clustering feature is essentially a statistical summary for the cluster at hand. Insertion into the CF-tree is similar to B+ trees i.e. a new object is inserted to the closest leaf entry and if the diameter of the leaf after insertion becomes larger than a threshold value, then that leaf node and may be other nodes are split. In the next stage, the approach uses any clustering algorithm to group the leaf nodes of the CF-tree in order to form a larger cluster and remove the sparser ones as outliers [38].
- **OPTICS** (Ordering Points to Identify the Clustering Structure) is an extension of DBSCAN algorithm. The algorithm loops over the data points and if there is an unprocessed core data point with respect to a given maximum radius and min-points then for every unprocessed neighboring point, find their neighbors and insert them into an ordered list according to their reachability distance. The algorithm then loops over the ordered list to check if there is any core point and insert their neighbors into the ordered list with the same criteria. The algorithm stops when there are no unprocessed points. This cluster ordering gives us the basic clustering

information such as cluster centers, intrinsic clustering structure, as well as visualization about clusters [4].

- **Anomaly Detection**

An outlier is a data point that is significantly different from other data points. Hawkins [17] defines an outlier as "an observation that differs so much from other observations as to arouse suspicion that it was generated by a different mechanism". In statistics, an outlier is an event that is at least three standard deviations away from the mean. Outlier detection has many applications such as fraud detection, intrusion detection systems, medicine, public health, law enforcement, fault detection in safety and critical systems, etc. Aggarwal in [1] makes a distinction between normal objects and noise; noise, outlier, and anomaly. Usually noise has a random distribution different from the distribution of the original data. An outlier is a data point that could be considered either an abnormality or noise. An anomaly is a special kind of outlier, which is of interest to analyst. Usually, noise should be detected and removed because it provides no interesting information to the results. Besides it reduces the quality of data. Noise are usually generated by human errors, measuring device errors, etc. However, anomalies are important because they might be a new phenomenon or show an unusual behavior or a critical problem. Hence, such terms as "anomaly detection", "outlier detection," and "noise removal" are commonly used in data mining literature.

Chandola et.al [9] states that the importance of anomaly detection is due to the fact that anomalies in data translate into important actionable information in many applications. For example, an anomalous traffic pattern in a network could mean that a hacked computer is sending out sensitive data to an unauthorized destination. An anomalous MRI image may indicate presence of malignant tumors; anomalous readings from a spacecraft sensor could mean a fault in some component of the spacecraft.

There are various anomaly detection techniques:

- **Model-based techniques**: Firstly, a model is constructed such that it fits the data well and if any data point does not fit the model, it would be considered anomaly. For instance, in a probabilistic model with a certain probability distribution, if any data point does not fit the model, it would be an outlier. In clustering model, a point that does not belong to any cluster, would be an outlier. In linear model, a data point that has a very large residual is more likely to be an outlier [33].
- **Proximity-based techniques**: Points that are remote from other data points are considered outliers.
- **Density-based techniques**: Similar to the previous approach as density is one of the proximity measures. Here, objects that belong to low density regions are likely to be outliers.

- **Evolution Analysis**

Evolution analysis deals with using data mining techniques to mine complex internal structures of data types that evolve over time such as temporal data, spatial data, spatiotemporal data, text data, web data, multimedia, sequential patterns, (sub)graph patterns, features in interconnected network, etc. It also concerns with finding regularities in datasets that help predict future trends and decision-making.

- **Mining Streams, Time-series, and sequence data**: These data sets are characterized by being temporally ordered, fast changing, massive and potentially infinite. Therefore, such data cannot practically be scanned more than once. Besides, they cannot be stored in any RAM or disk. For effective mining of such data, new data structures, algorithms, and techniques are needed. Some of the techniques used are random sampling and sketching, sliding window, histogram and multiresolution methods [19].
- **Graph mining, social, and multi-relational data networks:** Graphs are used for complex data structures such as circuits, images, web, biological and social networks. Frequent subgraph pattern is the basic graph pattern used for finding frequent patterns in the graphs. Social networks are characterized by being large heterogeneous datasets. They are usually represented by graphs where nodes correspond to objects and edges correspond to relations between these objects.
- **Other kinds of data:** this includes space-related data, multimedia data such as audio, video, image, graphics, text, speech, etc.

1.5 Privacy Preserving Data Mining

As the name suggests, this research area is a branch of data mining that investigates the alleviation of adverse side effects of data mining methods whereby the privacy of individuals and organizations are in dispute. There are number of techniques drawn from other fields such as information security, information hiding, statistical disclosure control (SDC) and cryptography. Although a new field, privacy preserving data mining (PPDM) becomes very important due to its necessity and possibility at the same time. Besides, the increasing numbers of different sophisticated data mining techniques make this field a major concern by many researches. Aggarwal in [5] classifies privacy issues arising from data mining into three categories:

- **Input Privacy (data hiding)**

Raw data may provide confidential information about its owners such as medical records, criminal records, records related to national and international security. Therefore, input privacy examines how to obfuscate raw data in such a way that no private data could be inferred while still allowing data miners to build accurate data mining models from the released data. A naive solution to this problem is to remove all personal identifying, sensitive data, and then releasing the results. However, in many cases it is hard to identify those sensitive data. Sometimes removing such data may deteriorate data utility. Another limitation called "inference problem", meaning that sensitive data can be inferred from other non-sensitive data [34]. Better solutions are suggested, here are some:

- Limiting access: this can be done by giving certain privileges to those who use the datasets. This technique is mainly used in DBMS.
- Data perturbation/obfuscation techniques such as swapping between data values, noise addition, randomization, and summarization.
- Eliminate unnecessary groupings: remove non-relevant information that do not contribute in the data mining and may reveal sensitive information with certain reliability.
- Distributed databases: instead of keeping data in one centralized position, we can divide these datasets into vertically or horizontally partitions and each partition send the data mining results without exposing the real data.

- **Output Privacy (knowledge hiding)**

When results of data mining methods reveal information that owners do not want them to be revealed such as business or trade secrets that provide competitive advantage to business competitors. A number of good examples are mentioned here [10]. Some devised algorithms incorporate privacy into data mining techniques such as association rule mining as discussed in the following chapters. Other classes of knowledge hiding include classification rule hiding, sequence hiding, cluster model hiding.

- **Classification rule hiding**: similar to association rule hiding approaches, where a set of classification results are sensitive and we should hide them.
- **Cluster Model Hiding (or privacy preserving Clustering PPC):** the problem appears most when sharing data between parties for clustering purposes. For example, sharing a cluster of patients who have the same disease may reveal the identity of those persons even if the identifying information (like id, name, address, etc.) is removed. Another example, when sharing data between two different companies for clustering analysis may reveal the attribute values of each other. In general, depending on the way of data sharing, PPC is classified into centralized-data PPC, horizontally partitioned PPC, and vertically partitioned PPC [28]. Methods for solving the problem are categorized into transformation-based methodologies and protocol-based methodologies.

Transformation-based methodologies are usually independent of the clustering algorithm used. Protocol-based approaches assume a protocol that control the flow of information between the parties and ensure that no sensitive knowledge is learned from the model [12].

- ❖ **Sequence Pattern Hiding**. A challenging and new research direction in PPDM. A heuristic approach in [3] that is similar in principle to Association rule hiding heuristic approaches is discussed in the next chapter.

- **Owner Privacy**

Sometimes data owners want to collectively mine their data without letting the other parties learn the content of these data. A common algorithm called secure multiparty computation (SMC) covers this problem.

1.6 Organization of the Dissertation

The rest of the dissertation is organized as follows:

- **Chapter 2** introduces Association Rule Hiding problem and association rule hiding approaches. In this chapter, we investigate data sanitization methods that hide sensitive itemsets/association rules by reducing either support or confidence of these sensitive patterns. The protection of sensitive knowledge is achieved by modifying a set of carefully selected transactions. In some cases, a number of items are deleted from a group of transactions with the purpose of hiding these sensitive patterns derived from those transactions.
- **Chapter 3** explains a number of heuristic approaches along with examples for each one. The chapter also contains the original sketch of every algorithm.
- In **Chapter 4**, we show experimental results of these algorithms, advantages and their side effects.
- **Chapter 5** concludes this dissertation with a brief summary of the work presented, a discussion of the main results achieved in this research, drawing some conclusions, and pointing to future work directions as a continuation of this research.

CHAPTER 2: ASSOCIATION RULE HIDING

In the previous chapter, we made a brief introduction to association rules concepts. In this chapter, we elaborate the two main steps of finding association rules namely; frequent itemsets and rule generation. We divide this chapter into two broad sections, in the first section we talk about the above mentioned association rules generation steps, their principles and major techniques and algorithms in literature. In the second section, we talk about association rule hiding concepts and algorithms.

2.1 Association Rules Generation

Finding significant association rules has two phases: first mine the database to find large/frequent itemsets and then generate rules from them. Usually the first part is computationally more challenging and therefore most of the research in this area focuses on this part. As mentioned before, an itemset is a set consists of one or more items in the original dataset and the frequency of these items to occur together is greater than (or equal to) minimum support threshold. This frequency occurrence is called itemset support. Each rule has a non-overlapping antecedent and consequent. Both of them are frequent itemsets. Strong rules are rules whose confidence is greater than minimum confidence threshold.

2.1.1 Frequent Itemsets Generation

Let I be a set of different items in the dataset D, i.e. I = $\{i_1 , i_2 , i_3$, ….., $i_n\}$. A brute force approach for finding frequent itemsets is to compare each candidate itemset with each transaction and count their frequencies. Such an approach is simple but computationally expensive because it requires O (N M w) comparisons, where N is the number of transactions and M = $2^k - 1$ number of candidate itemsets, k is the number of candidate items and w is the maximum transaction width [33]. Thus, there are other approaches more efficient than this as we will see in the next section.

In general, frequent itemset mining algorithms can be classified into the following categories:

A. **Join-based algorithms**: these algorithms generate candidate (k+1)-itemsets from frequent k itemsets using joins and then validate these (k+1)-itemset candidates to extract frequent itemsets from them. Apriori algorithm is the first in using this approach.

B. **Tree-based algorithms**: represent datasets in a compressed tree form. This technique reduces the search space dramatically. The first and most common approach that uses this approach is FP-growth.

C. **Vertical-based algorithms:** Rather than transactional convention that has<TID-Items> pairs; <Item-TIDs> can be used instead. This representation is very effective in finding the support of itemsets by intersection of TID list and automatically pruning irrelevant data. Partition and Eclat are examples of such approach.

2.1.1.1 Apriori Algorithm

For n different items in the dataset, there are 2^n -1 different candidate itemsets to be frequent. Fortunately, we do not need to consider them all. There is an important property of frequent itemsets called **Apriori Principle**. This property states that if an itemset is frequent, then all of its subsets must also be frequent. As an illustration, if itemset $\{i_1, i_2, i_3\}$ is frequent, then all the different combinations, i.e. $\{i_1\}$,$\{i_2\}$,$\{i_3\}$,$\{i_1, i_2\}$,$\{i_1, i_3\}$,$\{i_2, i_3\}$ are also frequent because any transaction contains$\{i_1, i_2, i_3\}$ should also supports its subsets. Similarly, if an itemset is not frequent, then all of its supersets cannot be frequent in any way. For example, $\{i_6\}$ is not a frequent itemset, then any itemset containing i_6 cannot be frequent. This property is also called anti-monotone (or downward-closed) property of the support measure. This property is used for pruning candidate itemsets in the search space.

Apriori algorithm makes use of this property to early prune infrequent itemsets and not to consider any of their supersets. It initially scans the dataset to find the support of each item and frequent 1-itemsets F_1. In the next step, the algorithm iteratively generates the candidate k-itemsets using frequent (k-1)-itemsets found in the previous iteration. To find the frequent itemsets, the algorithm makes an additional pass over the dataset to find the support of every candidate itemset and prone those whose supports are less than the minimum support threshold. The algorithm terminates when no generated candidate itemsets found. Using this approach, the total number of iterations required is K_{max}+1, where K_{max} is the maximum frequent itemset size.

There are many algorithms to find candidate itemsets in the Apriori algorithm. The most common one is to merge frequent (k-1)-itemsets only if their first (k-2)-itemsets are identical. For example, let $L_2 = \{\{i_1, i_2\}, \{i_1, i_4\}, \{i_2, i_3\}, \{i_2, i_5\}, \{i_3, i_5\}, \{i_5, i_6\}\}$. After the join step, C_3 will be$\{\{i_1, i_2, i_4\}, \{i_2, i_3, i_5\}\}$ only because $\{i_1, i_2\}$ and $\{i_1, i_4\}$ are merged to form $\{i_1, i_2, i_4\}$ and $\{i_2, i_3\}$, $\{i_2, i_5\}$ are merged to form $\{i_2, i_3, i_5\}$ since these are the only frequent 2-itemsets with the first item identical. This procedure is known as $\boldsymbol{F_{k-1}} \times \boldsymbol{F_{k-1}}$ **method** [33]. In this particular example, after the pruning step, only $\{i_2, i_3\ , i_5\}$ as a candidate itemset and $\{i_1, i_2, i_4\}$ will be pruned because $\{i_2\ , i_4\}$ is not in the frequent itemset list.

Apriori algorithm scans the dataset more than once and its performance degrade when the size of frequent itemsets are large. There are many variants of Apriori that attempt to optimize the Apriori performance. The following is a short explanation of the most common ones:

- **AprioriTid algorithm:** In the Apriori algorithm, we see that some transactions are scanned although they do not support any frequent itemset. At the same time, some items within the supporting transactions are also examined although they do not belong to any frequent itemsets. AprioriTid algorithm keeps a separate set $\overline{\overline{C_k}}$ which holds information <TID, $\{X_k\}$ > where X_k is the large itemset supported by transaction Tid [6]. In the subsequent iterations, candidate k-itemsets are generated from frequent (k-1)-itemsets as in the Apriori algorithm but the transactions to be scanned are only those in the $\overline{\overline{C_k}}$ .If any transaction supports none large frequent itemset, it will be removed from$\overline{\overline{C_k}}$. Thus at every iteration, size of the scanned database and the transaction length are reduced.
- **Apriori Hybrid algorithm:** It was seen that the performance of the AprioriTid degraded when a single transaction supports many candidate subsets because its corresponding entry in $\overline{C_k}$ would be large and consequently raises the overhead of the AprioriTid. Besides, sometimes $\overline{\overline{C_k}}$ is too large that it may not fit the main memory. This mainly happens with small values of k [5]. Therefore, AprioriHybrid algorithm uses Apriori first and then switches to AprioriTid when it expects that the set $\overline{\overline{\mathrm{C_k}}}$ at the end of the pass will fit in memory [6].
- **Apriori-LB:** we saw how Apriori principle could save a lot of computations. Here also is another trick to prune irrelevant data. In this idea, it is sufficient to know that the itemset is larger than the minimum support threshold i.e. the exact support value is not required. Let A and B be k-itemsets with (k-1) items common between them. Then it follows from the set theory that:

  $$\mathbf{Sup\ (A \cup B) \geq sup\ (A) + sup\ (B) - sup\ (A \cap B)}$$

  That is, if we know the support of the right hand side, which is already calculated, greater than or equal to the min-sup, then it is not necessary to explicitly find the support of A ∪ B since its lower bound is greater than or equal to min-sup [30].

2.1.1.2 Direct Hashing and Pruning (DHP)

DHP has two main advantages over the Apriori algorithm. The first is reduction of the candidate itemsets and the second is the reduced dataset size. DHP first scans the dataset to find the frequent 1-itemset, at the same time generates the 2-itemset for each transaction, and hash them into buckets in a hash table. Each bucket has a corresponding counter that counts how many itemsets are hashed into this bucket thus far. Each itemset in each bucket has a number that shows the frequency of the itemset. Itemsets in the bucket are candidates if the corresponding counter is greater than or equal to min-supp. Similar to the Apriori algorithm, a large itemset is a candidate itemset whose frequency is greater than min-sup. DHP reduces database size by removing transactions or items from transactions that do not contribute for the later large itemset generation. At iteration k, a transaction t is removed from D_k if it contains less than (k+1) large k-itemsets in the previous pass. An item is removed from transaction t if it appears less than k times in the large k-itemset [29].

The problem with this DHP is that more than one itemset can be hashed into the same bucket. For example suppose that we have a Bucket X: AD (3); DE (1) with min-sup = 4. The two itemsets will be taken as candidates because the total number of itemsets in X = 4 while none of them exceeds min-sup threshold. That is why each itemset should have a separate counter inside the bucket.

2.1.1.3 Partition Algorithm

The key observation of this algorithm is that an itemset can be globally frequent only if it is locally frequent in at least one partition. Thun this algorithm consists of two phases: in the first phase, the dataset is logically divided into non-overlapping partitions. Size of each partition should be small enough to fit into main memory. Each partition is mined sequentially to find all large itemsets (of all sizes) for each partition. These local large itemsets are merged to generate all sets of large itemsets that are considered candidate itemsets for the next phase. In the next phase, the actual support of the candidate itemsets is found and the actual large itemsets are identified [31].

This algorithm scans database only twice: once in the first phase to find large 1-itemset and once again in the second phase to find global large itemsets. In the first phase, to find the local large itemsets, **Partition** algorithm scans the transactions in each partition to find large 1-itemsets and stores the id of supporting transactions in a list called tidlist. This representation is known as **vertical database format** and used in many other algorithms such as Eclat as we will see next. This representation is opposed to the horizontal representation, which shows Tid and the supporting items. In the next iterations, the tidlist of candidate k-itemsets are generated by joining the tidlists of the two (k - l)-itemsets that were used to generate the candidate k-itemset. The way of generating candidate itemsets is similar to candidate generation of the Apriori algorithm, except that as each candidate itemset is generated, its count is determined immediately by intersecting the common tidlist of the (k-1)-itemsets. For example, let <itemset, *tidlist* > notation represents the itemset and supporting transactions pair. If <$\{i_1, i_2, i_3\}$,{1,2,4,6}> and <$\{i_1, i_2, i_5\}$, {1,2,6}> are two frequent itemsets at iteration ***k***. then the candidate itemset at iteration ***k+1*** is <$\{i_1, i_2, i_3, , i_5\}$, {1,2,6}> since {1,2,6} are the joined transactions between them.

When partitioning, there are two ways of selecting transactions in each partition: either sequentially or at random. Random selection is more preferred than sequential to avoid the problem of **data skew**.ie. Gradual change in data characteristics which may cause generation of local large itemsets to get high local supports while might not be frequent in the global large itemsets. We can avoid this effect by randomizing data in each partition.

Another problem with **Partition** is how to determine the number of partitions. As the dataset grows, the itemsets and their tidlist grow and may not fit in main memory due to joining tidlist of itemsets. Therefore, partition sizes should be chosen such that at least those itemsets and their tidlist can fit in main memory. One of the solutions to this problem is called **diffset** [36]. Instead of taking the intersecting itemsets, we keep track of their differences. For instance, in the previous example, instead of taking {1, 2, 6} as the intersecting transactions, we take {4}, the difference between the two tidlists.

2.1.1.4 FP-Growth Algorithm

In this algorithm, transactions are represented in an efficient data structure called "FP-tree". Each transaction is represented as a path in the FP-tree. A single path can be shared by many transactions as long as they have items in common. Thus, the best-case scenario is when all transactions have the same set of items and the worst-case scenario happens when all transactions have unique items [33].

The tree is constructed as follows: initially the database is scanned to find the frequent 1-itemset and discard all infrequent items. Items in the transactions are then sorted in a descending order of large 1-itemset frequency. The tree is built by examining one transaction at a time. Each node in the path represents an item and has a counter that counts the repetition of its corresponding item in that path. Same items in different paths have pointers between them.

To find the frequent itemsets, the tree is traversed from the leaves in a bottom-up fashion. From a certain leaf, we examine the paths that contain the leaf item. These are called conditional pattern-bases from which we construct a **conditional FP-tree** and mine it recursively to find the frequent itemsets.

2.1.1.5 Eclat Algorithm (Equivalence Class Transformation)

Ecat uses the vertical layout of the database. This algorithm first scans the dataset to find the unique items and the transactions supporting them. It then finds the large 1-itemset and discards the irrelevant ones. As **partition** algorithm, every 1-itemset has a list of supporting transactions called tidlist. Eclat uses the same approach as Apriori to generate the candidate (k+1)-itemsets from large k-itemsets. However, supporting transactions for candidate (k+1)-itemset are found by joining the tidlist of the large k-itemsets generating them. This guarantees minimal scans of the datasets. As Partition, a problem may rises when tidlist is too long so that it could not be fit into the main memory and thus diffset can be used instead.

2.1.1.6 Maximal and Closed Frequent Itemset Mining

Important concepts in frequent itemset mining are maximal and closed frequent itemsets because they provide a compressed representation of frequent itemsets. An itemset is considered closed itemset if none of its immediate supersets has the same support. In addition, it is called frequent closed itemset if it is closed and its support is greater than or equal to the minimum support threshold. However, an itemset is called maximal frequent itemset if none of its immediate supersets is frequent. Thus, maximal frequent itemsets are subsets of closed frequent itemsets. However, there is a difference between mining maximal itemsets and mining closed itemsets in that maximal itemsets loses information about supports of the underlying itemsets while mining closed itemsets preserves such information. Han et.al in [21] states that mining closed frequent itemsets has the same power as mining the complete set of frequent itemsets and substantially reduces the generated association rules which are not significant and thus increases the efficiency of data mining results. Here we briefly describe some of the well-known algorithms for mining closed and maximal frequent itemsets.

A. **CLOSET and CLOSET+:** generates frequent itemsets from the closed itemsets since closed itemsets are lossless compression of the frequent itemsets**.** An itemset is called closed itemset if no immediate superset has the same support. A closed itemset is frequent if its support passes the support threshold value. **CLOSET** uses FP-tree to construct a compressed database in order to extract the closed frequent itemsets and thus, avoid generating candidate itemsets of the Apriori algorithm. The algorithm works as follows: it first scans the dataset to find the frequent 1-itemsets and sorts them in a descending order. The result is sorted in a list f-list of size N. In the next step, the algorithm divides the search space into N non-overlapping subsets. For example, if f-list = <c:4, a:3, d:2> , then the frequent closed itemsets can be divided into 3 non-overlap subsets:

a. The ones containing d

b. The ones containing a but no d

c. The ones containing only c

Now, staring from d, the algorithm scans the dataset to find transactions that contain d. Item d, all infrequent items, and items following d in the f-list are deleted. The result is called conditional database under d or, $TDB|_d$. Here there are no items following d since it is the last element. However, the case applies for other elements. The closed frequent itemset from $TDB|_d$ is a frequent itemset in which no superset has the same support. This process is continued for other elements in f-list to find other frequent closed itemsets [21].

B. Max-Miner algorithm:

This algorithm returns only the maximal frequent itemsets and is useful in situations when all frequent itemsets are large and computationally expensive. Max-Miner uses an enumeration tree and breadth-first search to reduce the search space. To maximize the optimization utility, Max-Miner imposes an ordering on the set of items. Each node in the tree represents a "**candidate group**". This candidate group consists of the head of the group h (g) and the tail of the group t (g). At the root node, it examines if the whole set of items are frequent. If so, it returns the whole itemset as the maximal frequent itemset. If not, it examines every single item and removes those infrequent ones. In the next level, with certain ordering, it takes every single item as a head of the candidate group and the remaining elements as the trail and examines if h (g) ∪ t (g) is frequent. If so, it returns it as the maximal frequent itemset. Otherwise, it examines h (g) ∪ {i} where {i}∈ t (g), if not frequent, then {i} will be moved to that node. As a concrete example, let {ABCD} an itemset at node {A}. **Max-Miner** scans the dataset to see if {ABCD} is frequent. If so, it returns it as the maximal frequent itemset. Otherwise, it examines if {AB} is frequent, if not, it removes {B} from the itemsets in the following nodes and does the same thing with the other items. In the child nodes, it enumerates the remaining items and repeats the same process. For instance, in the previous example, suppose the remaining items after pruning is {ABC}, and then the following children will be:

A(BC), B(C), C() where, for example, in the first node, {A} is the head of the candidate group and (BC) is the tail. Again, **Max-Miner** examines if {ABC} is frequent and returns it as the maximal frequent itemset if so. Thus, Max-Miner uses subset-infrequency and superset-frequency to prune branches and nodes of the tree [14].

2.1.2 Rules Generation

As stated before, the first step for finding association rules is more costly and determines the overall performance of the process. However, the second step is straightforward and requires fewer computations. In general, association rules are generated from the frequent itemsets. Each frequent k-itemset can produce up to 2^k -2 association rules; excluding the empty set at the antecedent and consequent of the rule. Before discussing this approach, we should mention an important principle for pruning insignificant rules. This principle is known as confidence-based pruning principle.

- ❖ **Confidence-based pruning principle**: states that if the rule X → X - Y does not satisfy the minimum confidence threshold, then any rule $X' \rightarrow X' - Y$, where $X' \subseteq$ X, must not satisfy the confidence threshold as well.

The most common Association Rule Generation algorithm is also based on the Apriori principle.

- **Rule generation using Apriori Algorithm**: association rules are extracted from frequent itemsets by partitioning each frequent itemset X into Y and X-Y such that Y → X-Y satisfies the minimum confidence threshold. Apriori algorithm uses a level-wise approach where each level corresponds to number of items in the rule consequent. This guarantees starting from the highest confidence rule and thus pruning any rule that is not significant and making use of the confidence-based pruning principle. For example, if the rule confidence {bcd} → {a} is lower than minimum confidence threshold, then any rule containing item a in its consequent can be pruned immediately [33].

2.2 Association Rules Hiding (ARH)

This section is the major concern of our thesis. As stated before, association rule hiding comes under the category of output privacy (or knowledge hiding). Association rule mining is one of the data mining tasks that investigate the relationships between elements in the datasets. As such, knowledge might sometimes be undesirable to be revealed by the database owner, many researches are made to solve this problem. Many research papers refer to the resultant database after applying the ARH algorithm by the **sanitized database** and the process as **sanitization**. There are two variants of ARH algorithms, either to hide sensitive rules directly or to hide the sensitive itemsets generating them.

In general, association rule hiding approaches should achieve the following goals under the same support and confidence thresholds or higher [12]:

1. No sensitive pattern determined by the database owner is revealed after the application of ARH algorithm.

2. Non-sensitive pattern that can be mined from the original database should also be mined from the sanitized database. Such an original pattern that could not be mined from the sanitized database is known as **missing pattern**.
3. No new pattern generated from the sanitized database. Such a pattern is known as false or ghost pattern.

2.2.1 Heuristic Approaches

This set of algorithms are fast and efficient but in many cases they do not guarantee the optimal solution and they are not free from the side-effects in terms of missing patterns, artificial patterns and even failure in hiding sensitive patterns. Heuristic approaches selectively sanitize a set of transactions from the original database in order to hide sensitive patterns. There are two schemes of these approaches. Both of them can be either support-based or confidence-based depending on whether the algorithm uses the support or the confidence of the pattern to drive the hiding process.

- **Distortion scheme:** depends on reducing the support or the confidence of the sensitive pattern until they lie below specified thresholds. This is done by addition or deletion of item(s) in the sanitized transactions. The choice of items and transactions relies on the maximum effect on the sensitive patterns and the minimal effect on the non-sensitive ones. In the following chapter, we elaborate on some of the common algorithms under this scheme.
- **Blocking scheme:** instead of distorting original datasets by deletions and additions, an interesting line of research proposes replacing sensitive data with unknowns (question mark"?"). The justification of using unknowns is that distortion scheme add false information to the original datasets which could affect mining results afterwards especially in critical life applications such as medical records. Due to the introduction of unknowns, support and confidence of the interesting patterns mined from the sanitized datasets will be fuzzified to an interval (not exact as in distortion-based scheme) [32].

2.2.2 Border-Based Approaches

These approaches provide an improvement over heuristic approaches discussed in the previous section by tracking impact of deleting items on the border separating frequent/infrequent itemsets.

For frequent and infrequent itemsets deduced from any dataset, there is a borderline separating these frequent and infrequent patterns. Border-based approaches hide sensitive knowledge by modifying the borders of the lattice of frequent and infrequent patterns in the dataset. The maximal frequent itemsets whose immediate supersets are not frequent are referred to as the positive border and denoted $Bd^{+}(F_D)$. Similarily, the minimal set of infrequent itemsets whose all-immediate subsets are frequent are known as the negative border and is denoted as $Bd^{-}(F_D)$. The modified borders are known the revised negative/positive borders and the process is known as **Border Revision Algorithm** [16].

More formally, let F_D= {I ⊆ Γ: freq(I , D) ≥ minf} be the set of all frequent itemsets in D, and $\mathcal{P} = \wp(I)$ be the set of all patterns in the lattice of D. The positive and the negative borders of $F_D$are defined as follows: $Bd^{+}(F_D)$ = {I ∈ F_D| for all J ∈ $\mathcal{P}$ with I ⊂ J we have that J ∉ F_D} and $Bd^{-}(F_D)$ = {I ∈ $\mathcal{P}$ - F_D| for all J ⊂ I we have that $J \in F_D$}.

Other important terms in this respect are minimal sensitive itemsets S_{min} which is the minimal set of itemsets in S. Mathematically defined as S_{min}= {I ∈ S | for all J ⊂ I , J ∉ S }. Maximal sensitive itemset S_{max} is the sensitive itemsets and their supersets. Thus S_{max} = {I ∈ F_{D_O} | ∃ J ∈ S_{min} , J ⊆ I }. For example, if the sensitive itemset S = {e,bc,ae}, S_{min} can be = {e,bc } and S_{max} can be = {e, ae ,bc, be, ce, abc, ace}.

To find the negative border, the algorithm first checks the 1-itemsets and if the support of any itemset is smaller than min-support, then it belong to the negative border $Bd^{-}(F_D)$. In the next step, the algorithm generates the candidate itemsets from frequent 1-itemset using Apriori-Gen procedure of the original version of Apriori. Starting from candidate 2-itemset onward, if the support of the candidate itemset is smaller than min-supp; it will belong to the negative border $Bd^{-}(F_D)$.

After finding the original negative border, the next step is to find the original positive border. The algorithm for finding positive border works as follows: For each frequent itemset, a counter is initialized to zero. Then we sort the frequent itemsets in decreasing order of size. In the next step, for each large (k-1)-itemsets, if there is any frequent (k-1)-itemset that is a subset of the current k-itemset, we increase the corresponding counter of the k-itemset by one. The algorithm iterates from k-itemset up to 1-itemset. Finally the itemsets of counter zero belongs to the positive border $Bd^{+}(F_D)$. This algorithm applies for finding original and revisited positive borders [16].

One final algorithm in this respect is to find the revised negative border. This algorithm is straightforward. It works as follows: for each 1-itemset, if it is not frequent, then it belongs to the revised negative border. In the next step for each frequent 1-itemset, join them pairwise and examine if they are frequent. If they are not so, then they belong to the revised negative border. Finally for each k-itemset where k > 3, the algorithm joins (k-1)-itemsets. For each generated itemset, if it is not frequent and none of its (k-2) subsets is frequent, then it belongs to the revised negative border [16].

- **Border-Based Algorithm (BBA) algorithm:** The key idea of this algorithm is to keep track of the positive border elements by assigning them weights showing their vulnerability of being affected by item deletion. These weights are continuously updated based on its corresponding itemset support during the hiding process. The authors suggest the following equation to track these weights:

$$\mathbf{W(I\in\ Bd^{+})=}\begin{cases}\frac{sup(I,D_0)-sup\,(I\,,D')+1}{sup(I,D_0)-msup}\ , & sup(I,D')\geq msup+1\\ \lambda+msup-sup(I\,,D')\,, & 0\ \leq\ sup(I\,,D')\ \leq msup\end{cases}$$

where λ is an integer greater than number of itemsets in the revised positive border,D' represents database during sanitization.

In order to find the victim item, the algorithm associates an interval with each item in the sensitive itemset where the left bound is the summation of the weights of the direct positive border elements and the right bound is the summation of the weights of all relevant positive border elements including the direct positive border elements. Having known the impact of each item on the border elements, a partial order relation $\gtrsim$ is used to find the item with minimal impact on the border [44].

An intelligent way to reduce the search space for finding the victim transaction is to associate a vector map with each supporting sensitive transaction. The length of the vector equals to $|Bd^{+}|_{x_i}|$ i.e. the positive border elements supported by the victim item found in the previous step. Then the victim transaction is the one with the least weight summation [44].

- **MaxMin1 Algorithm:** This and the next algorithm are based on the maxmin criterion that is a method in decision theory to maximize the minimum gain. They hide sensitive itemsets according to a set of theories devised by authors [45, 46]. In MaxMin1, each item belonging to the sensitive itemset has a list called affinity list defining the possibly affected positive border elements. From each list, we take the itemsets with the minimum supporting itemsets and from the result; we take the maximum of this minimum itemsets. That is the reason behind the name maxmin itemset. From this maxmin list, we select an itemset and its corresponding item in the affinity list to be the victim item. Finally, we remove this item from the first transaction supporting the sensitive itemset.
- **MaxMin2 Algorithm:** This algorithm improves over the previous algorithm. Based on the theories suggested by the authors in [45, 46], the algorithm distinguishes between three cases the first case scenario is when the maxmin itemsets $Bd^{+}|_{j}$ belong to only one tentative victim item *j*. Here we delete this item from transactions supporting the current sensitive itemset without supporting any of the maxmin itemsets. Otherwise, it chooses a transaction at random. The second case scenario is when the maxmin itemsets are all derived from different tentative victim items. In this case, the algorithm iterates over each maxmin itemset relevant to each tentative

victim item to find the transaction supporting current sensitive itemset without supporting any of the maxmin itemsets relevant to that item. If there are any, it removes the victim item from a transaction in the resultant list. . When all cases fail, the algorithm iterates over all possible pair itemsets in the maxmin list to find transactions affecting only one list, if there are any, it removes the corresponding victim item from one of them. Otherwise, it removes victim item from a random selected transaction supporting the first list [45, 46].

2.2.3 Exact Approaches

Exact approaches use integer or linear programming to hide the sensitive knowledge. They guarantee the optimal solution provided that the optimal solution exists. However, they are slower than heuristic approaches by several orders of magnitude [12]. Many algorithms use this approach like Menon algorithm, which was the first to introduce the concept on integer programming to solve ARH problem. Other algorithms include Inline algorithm, Two-phase iterative algorithm and Hybrid algorithm.

- **Menon Algorithm:** consists of two parts: the exact part and the heuristic part. The purpose of the exact part is to identify the minimal number of transactions required to hide the sensitive itemsets. This is done by finding Constraint Satisfaction Problem(CSP) from the sensitive patterns and the transactions supporting them as follows:

$$\textbf{(FIH)} \qquad \min \sum_{i \in \mathfrak{D}} X_i \, ,$$

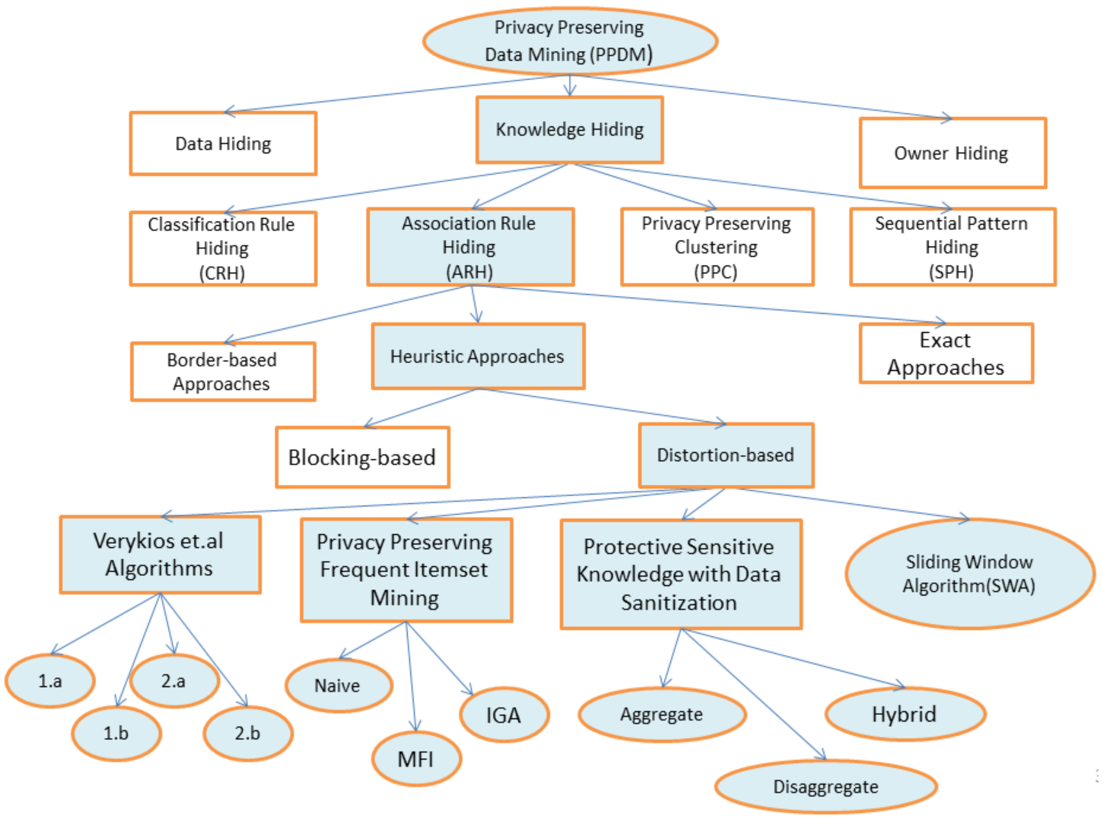
$$\text{s.t} \sum_{i \in \mathfrak{D}} a_{ij} X_i \geq (\sigma_j - \sigma^{j}_{min} + 1) \ \forall\, j \in \mathfrak{T}^R(\sigma_{min}), \qquad (1)$$

$$X_i \in \{0,1\} \quad \forall \ i \in \ \mathfrak{D} \qquad (2)$$

After forming the CSP, the algorithm solves them using integer programming in order to find the minimal required transactions to modify. In the next step, a heuristic algorithm is used to identify the items to delete from the transactions found in the previous step. Authors also propose two heuristic strategies, both of which are inspired from the heuristic approaches of previous works especially that in [27]. The first proposed strategy, called Blanket ,similar to Naive algorithm in [27], operates by deleting all items of the sensitive pattern from the sensitive transaction. The second strategy, known as Intelligent, also similar to IGA in [27] operates by deleting items that appear in the largest number of sensitive patterns [25]. We will discuss these algorithms in the next more with details.

CHAPTER 3: ARH using HEURISTIC APPROACHES

As stated before, Heuristic approaches carefully select a set of transactions and sanitize them in order to hide sensitive knowledge. These algorithms are characterized by being simple, fast, efficient, and scalable. However, they do not guarantee the optimal solution. Besides, they usually suffer from the local optima problem. There are two schemes of heuristic approaches. In this chapter, we examine the distortion-based ones. The following graph shows the sketch of our study.



Privacy Preserving Data Mining (PPDM)
Data Hiding
Knowledge Hiding
Owner Hiding
Classification Rule Hiding (CRH)
Association Rule Hiding (ARH)
Privacy Preserving Clustering (PPC)
Sequential Pattern Hiding (SPH)
Border-based Approaches
Heuristic Approaches
Exact Approaches
Blocking-based
Distortion-based
Verykios et.al Algorithms
Privacy Preserving Frequent Itemset Mining
Protective Sensitive Knowledge with Data Sanitization
Sliding Window Algorithm(SWA)
1.a
2.a
1.b
2.b
Naive
IGA
MFI
Aggregate
Hybrid
Disaggregate

Figure (3.1) sketch of ARH heuristic approaches

3.1 Verykios et.al Algorithms

3.1.1 Algorithms Strategies

There are two hiding strategies in this set of database sanitization algorithms. Both of them are based on decreasing the confidence of the sensitive rules to a value lower than minimum support threshold directly or indirectly by reducing the support of the frequent itemsets generating them.

We know that the Interesting measures of the association rules are calculated as:

$$\text{Support,}\quad s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \qquad (1)$$

$$\text{Confidence,}\quad c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \qquad (2)$$

where **X** is the rule antecedent , **Y** is the rule consequent, **σ(X U Y)** is number of transactions in which both **X** and **Y** occur , **σ(X)** number of transactions in which **X** occur, **N** is the size of the database.

For these two measures, support and confidence, there are two corresponding thresholds: support threshold, confidence threshold defined Apriori by the user or the database owner.

For instance, if we want to decrease the support of an itemset, we can decrease the value of the numerator in equation 1 by finding transactions that fully support this itemset and removing one or more items that exist in the current itemset from the transaction. This item is usually called victim item. Sometimes more than one item is chosen as we see later.

We can hide sensitive rules either by:

1. Decreasing the numerator of equation 2 while keeping the denominator fixed. This is done by finding transactions that fully support both sides of the rule and removing one or more items form the right hand side of the rule.
2. or increasing the denominator of equation 2 while keeping the numerator fixed. This is done by modifying transactions that partially support rule antecedent and consequent.

Thus, the hiding strategies devised by Verykios et.al [35] are the following:

- Decreasing the support of the rule by decreasing support of the rule antecedent or rule consequent. We can do that by removing one or more items from the transactions that fully support both sides of the rule.
- Decreasing the confidence of the sensitive rule by either decreasing the support of the rule consequent in the transactions fully supporting it or increasing the support of the rule antecedent in the transactions that partially support rule antecedent and consequent.

3.1.2 Algorithms Assumptions

The following are the assumptions that we should take into account when applying these heuristic strategies:

1. We can hide only rules that are supported by disjoint large itemsets. That is, if we have two sensitive itemsets like [ABC, ABD], we have to choose only one of them

since there is an intersection between them i.e.[AB]. The justification behind this assumption is that hiding overlapping rules may bring back other rules that have already been hidden." This may increase the time complexity of the algorithms since hiding rules may cause an already hidden rule to haunt back"[35]. The writers also mention that if we want to hide overlapping sensitive rules we should consider transaction selection mechanisms such as choosing the transactions that do not only support the antecedent of the overlapping rules.

2. We hide sensitive rules or the frequent itemsets generating them by decreasing either its support or confidence but not both.
3. We select to decrease either support or the confidence based on the side effects on the information that is not sensitive. Actually, all heuristic approaches mentioned in this thesis consider this assumption. Since the purpose of the data mining is the extract useful patterns from large amounts of data, Privacy Preserving Data Mining strategies in general and Association rule hiding approaches in particular should be used with care not to deteriorate non-sensitive information and apply minimal changes in the original database.
4. We hide only one rule at a time. This is a direct consequence of the first assumption since sensitive rules are assumed to be disjoint and their items are different.
5. We decrease either the support or the confidence one unit at a time. As we see later, most algorithms are based on choosing one sensitive transaction with certain criteria like minimum-length transaction (in case of deletion) or maximum supporting items (in case of addition).

3.1.3 Algorithms

3.1.3.1 Algorithm 1.a

This algorithm is based on the first strategy above i.e. finding transactions that partially support of the rule antecedent and consequent and adding all items that are missing from rule antecedent to those transactions. Using these steps repeatedly, the support of the rule antecedent increases and thus rule confidence decreases. Algorithm stops when rule confidence goes below than or equal to the minimum confidence threshold. We can find the number of iterations required to make the rule confidence lower than (or equal to) the minimum confidence in advance using the following lemma [35]:

$$N\_iterations = \left\lceil |D| \left(\frac{\sup(r)}{\min\_conf} - \sup p(l_r) \right) \right\rceil \tag{3}$$

Where supp (r) is the support of the rule, supp ($\boldsymbol{l_r}$) is the support of the rule antecedent. |D| is the size of the original database and min-conf is the minimum confidence.

Example 1: Let the database be the following table (Table 1), min-supp = 0.2 and min-conf = 0.4.

Table (3.1) a list of transactions

TID	Items
T0	1, 2 , 3 , 4
T1	1, 2 , 3
T2	1, 2, 4
T3	1, 3, 4, 5
T4	1, 2, 3
T5	1, 2, 4
T6	2, 3, 6, 8
T7	1, 2 , 4 , 8
T8	2, 3 , 4 , 6
T9	4, 5

With these thresholds, we have 15 non-singleton frequent itemsets and 37 significant rules. IF we relaxed the first assumption, using Alg.1a algorithm, we have the following values for corresponding sensitive rules chosen at random.

Table (3.2) Rule examples and their supports and confidences

Sensitive Rule	Left rule support	Support	Confidence (%)	Partial Supporting Transaction indices	Iterations Required (N-iterations)
$8 \rightarrow 2$	2	2	100	[3 , 9]	3
$2 , 3 \rightarrow 1$	5	3	60	[9]	3
$3 , 4 \rightarrow 1$	3	2	66	[6 , 9]	3
$6 \rightarrow 2 , 3$	2	2	100	[2,3,5 7,9]	3

Because this algorithm hides sensitive rules by decreasing confidence to a value lower than min-conf, it does that by increasing support of the rule antecedent one unit at a time.
In this particular example, we can hide the fourth sensitive rule only. The reason is that number of iterations required for making the confidence of the sensitive rule lower than minimum confidence, **N-iteration**, is larger than partial supporting transactions as seen in the table. In the fourth sensitive rule, partial supporting transactions are 5 while we need only three to make this rule's confidence lower than minimum confidence threshold.
Using this example, we see that 3 out of four sensitive rules cannot be hidden (Hiding Failure = 75%). Because this algorithm is based on augmentation, usually no missing found

(here missing costs = 0.0%). In contrast many new rules are generated (Artificial Rules = 55%). Database difference = 11.77%.

Table (3.3) an overview of this algorithm is depicted in Fig (3.2)

```
Algorithm 1: 1.a: [35]
INPUT: a set $R_H$ of rules to hide.the source database D, the number |D| of transactions in D, the min-conf,  threshold , the minimum support threshold.
OUTPUT: the database D transformed so that the rules in $R_H$ cannot be mined.
Begin
Foreach rule r in $R_H$ do
{
    1.  $T'_{l_r}$ = { t in D / t partially supports $l_r$ }
    2.  for each transaction of $T'_{l_r}$ count the number of items of $l_r$ in it.
    3.  sort the transaction  in $T'_{l_r}$ in descending order of the number of items of $l_r$ supported
    4.  repeat until (Conf (r) < min-conf)
        {
    5.  choose the transaction t ∈ $T'_{l_r}$ with the highest number of items of $l_r$ supported( t is the first transaction in $T'_{l_r}$ )
    6.  modify t to support $l_r$
    7.  increase the support of $l_r$ by 1.
    8.  recompute the confidence of r
    9.  remove t from $T'_{l_r}$
        }
    10. remove r from $R_H$
}
End
```

Figure (3.2) a sketch of algorithm 1.a.

In order to preserve the third assumption, the idea here of choosing transactions for modification is the following: choose transactions that partially support the rule antecedent and consequent. Then order them according to the number of items in the rule antecedent. This is because we are going to add the remaining items to these transactions. As an example, if ABC $\rightarrow$ DE is a sensitive rule and the transactions that satisfy this criteria T1 = { A, B, D }, T2 = { A, E }, T3 = { B,C D}. All these transactions are applicable, because they partially support the rule antecedent and consequent. Yet when choosing a transaction to modify, the order can be T1, T3, T2 or T3, T1, T2 because transactions supporting the largest number of the left-hand-side (LHS) are T1 and T3.

3.1.3.2 Algorithm 1.b

This algorithm is based on hiding sensitive rules through transactions that fully support the rule antecedent and consequent. The transactions are then sorted in ascending order by size. The item to be removed from the transaction is the item that belongs to the sensitive itemset with maximum support. This is to maintain the minimal effect on the original database and preserve integrity of the data mining results. Here also we can calculate the number of iterations required to hide the sensitive rule beforehand. A sensitive rule is hidden if either its support goes below support threshold or its confidence goes below confidence threshold. We have already defined the number of iterations necessary to make the confidence goes below confidence threshold in Alg.1a algorithm. Similarly, iterations required to make rule support goes below minimum support threshold is defined by the following equation:

$$N\_iterations = \left\lceil |D| \left(\frac{\sup(r)}{\min\_conf} \right) \right\rceil \quad (4)$$

An overview of this algorithm is shown in Fig (3.3).

```
Algorithm 2: 1.b. [35]
INPUT: a set $R_H$ of rules to hide, the source database D, the number |D| of transactions in D, the min-conf threshold , the minimum support threshold.
OUTPUT: the database D transformed so that the rules in $R_H$ cannot be mined.
Begin
Foreach rule r in $R_H$ do
{
    1. $T_r$= { t in D / t fully supports r}
    2. For each transaction of $T_r$ count the number of items in it.
    3. Sort the transaction in $T_r$ in ascending order of the number of items supported
    4. Repeat until (conf(r) < min-conf or supp(r) < min_supp)
       {
           4.1 Choose the transaction t ∈ $T_r$ with the lowest number of items( the first transaction in $T_r$ )
           4.2 Choose the item j in $r_r$ with the minimum impact on the (|$r_r$| - 1) – itemsets
           4.3 Delete j from t
           4.4 Decrease the support of $r$ by 1.
           4.5 Recompute the confidence of r
           4.6 Remove t from $T_r$
       }
    5. Remove r from $R_H$
    6. }
End
```

Figure (3.3) a sketch of Algorithm 1.b.

Example 2: using the same dataset in Table 1 with the same threshold values and sensitive rules, results are shown in Table3 below

Table (3.3) Rule examples and their supports and confidences

Sensitive Rule	Left rule support	Support	Confidence (%)	Fully Supporting Transaction indices	Iterations Required (N-iterations)
8 → 2	2	2	100	[6,7]	1
2 , 3 → 1	5	3	60	[1,4,0]	2
3 , 4 → 1	3	2	66.7	[0, 3]	1
6 → 2 , 3	2	2	100	[6,8]	1

Since N-iterations is smaller than number of fully supporting transactions, we can hide all sensitive rules using 1.b algorithm. Note that we purposely arrange supporting transactions for the second rule in that order because the algorithm arranges supporting transactions in an ascending order of size.

Using this algorithm, all sensitive rules are hidden (Hiding Failure = 0.0 %) but there is missing rules in the sanitized database (missing costs = 45.45%). There are no artificial patterns (Artificial Patterns = 0.0%). Difference between original and sanitized database sizes is 14.71 %. We justify these results since the algorithm is based on deletion.

3.1.3.3 Algorithm 2.a

This algorithm is similar to **1.b**. The only difference is the item chosen for removal. Here, we choose the item from the antecedent and consequent itemsets of the rule, as opposed to rule consequent only in algorithm 1.b. A sketch of this algorithm is shown in Fig (3.4).

```
**Algorithm 3: 2.a.** [35]

INPUT: a set $R_H$ of rules to hide.the source database D, the size of the database |D| ,the min_conf
threshold , the minimum support threshold.
OUTPUT: the database D transformed so that the rules in $R_H$ cannot be mined.
Begin
Foreach rule r in $R_H$ do
{
    1. $T_r$= { t in D / t fully supports r}
    2. For each transaction of $T_r$ count the number of items in it.
    3. Sort the transaction in $T_r$ in ascending order of the number of items supported
    4. Repeat until (conf(r) < min-conf or supp(r) < min_supp)
       {
          a. Choose the transaction t ∈ $T_r$ with the lowest number of items( the first transaction
             in $T_r$ )
          b. Choose the item j in $r$ with the minimum impact on the (|$r$| - 1) – itemsets
          c. Delete j from t
          d. Decrease the support of $r$ by 1.
          e. Re-compute the confidence of r
          f. Remove t from $T_r$
       }
    5. Remove r from $R_H$
}
End
```

Figure (3.4) a sketch of Algorithm 2.a.

Example 3: Using the same dataset in Table 1 with the same threshold values and sensitive rules, we get the results shown in Table 4.

Table (3.4) Rule examples and their supports and confidences

Sensitive Rule	Left rule support	Support	Confidence (%)	Fully Supporting Transaction indices	Iterations Required
8 → 2	2	2	100	[6,7]	1
2 , 3 → 1	5	3	60	[1,4,0]	2
3 , 4 → 1	3	2	66.7	[0, 3]	1
6 → 2 , 3	2	2	100	[6 , 8]	1

Here also all sensitive rules are hidden (Hiding Failure = 0.0 %), number of missing rules is large (missing costs = 51.0 %) and no artificial patterns (Artificial Patterns = 0.0 %).

3.1.3.4 Algorithm 2.b

Instead of hiding sensitive rules directly, as in the previous algorithms, this algorithm hides sensitive rules from the frequent itemsets generating them. Thus sometimes called multiple rule hiding since hiding a sensitive itemsets necessarily implies hiding more than one rule , e.g. if [AB] is a sensitive itemset then $A \rightarrow B$ and $B \rightarrow A$ are sensitive rules. In this algorithm, sensitive itemsets are sorted by their size then support in a decreasing order. Then, for every sensitive itemset, we find the supporting transactions and sort them in an ascending order of size. The rational is to limit the impact of sanitization on the non-sensitive patterns. The chosen item for deletion is the item in the sensitive itemset with the maximum support. An overview of this algorithm is in Fig (3.5).

```
Algorithm 4: 2.b. [35]

INPUT: a set L of large itemsets , the set $L_H$ of large itemsets to hide , the database D ,the min-supp threshold .
OUTPUT: the database D modified by the deletion of the large itemsets in $L_H$.
Begin
    1.  Sort $L_H$ in descending order of size and support of the large itemsets.
        Foreach Z in $L_H$
        {
        1.1   Sort the transactions in $T_Z$ in ascending order of transaction size
        1.2   N-iterations = $|T_Z|$ - min_supp * |D|
        1.3   For k = 1 to N-iterations do
              {
              1.3.1 Remove the maximal support item of Z from the next transaction in $T_Z$
              1.3.2 Update the database D
              }
        }
}
End
```

Figure (3.5) a sketch of Algorithm 2.b.

Example 4: using the same dataset in Table 1 with the same support threshold values, we choose a set of frequent itemsets selectively to be sensitive.

Table (3.5) Sensitive Itemsets and corresponding transactions

Sensitive Itemset	Support	Fully Supporting Transaction indices	Iterations Required
1, 2, 4	4	[2 , 5, 0 , 7]	3
3 , 6	2	[6 , 8]	1
4 , 5	2	[9 , 3]	1

We intentionally arrange sensitive itemsets by length starting from the largest one (here [1, 2, 4] is the first). The reason for that is to affect the largest number of sensitive itemsets. Supporting transactions are also arranged by size in ascending order. The Item chosen for deletion is the one with the maximum support. The rational again is to minimize effect of sanitization on the database.

In this example, all chosen sensitive itemsets are hidden (Hiding Failure = 0.0 %), two non-sensitive itemsets are deleted (missing costs = 10.53 %) and no artificial patterns found (Artificial Patterns = 0.0 %).

3.2 Privacy Preserving Frequent Itemset Mining

This set of algorithms hide sensitive itemsets. Yet besides minimum support threshold as a measure to find large frequent itemsets, they also suggest another threshold called **Disclosure Threshold**. "This threshold basically expresses how relaxed the privacy preserving mechanisms should be" [27]. Thus, it serves as a compromise between preserving privacy and maintaining accuracy of data mining results. If the disclosure threshold is 0%, no restrictive pattern is allowed to appear in the sanitized database and when 100%, there are no restrictions on them. Using disclosure threshold, we define number of transactions we need to modify by the following equation:

$$NumTrans_{rpi} = \left|T[r_{pi}\right| * (1 - \Psi) \qquad (5)$$

where | T [r_{pi}] | number of sensitive transactions supporting restrictive pattern r_{pi} , Ψ disclosure threshold value.

Example 5: In the following table (Table 6), suppose min-supp = 30%. Let sensitive frequent itemset Rp1 = [ABC]

Table (3.6) transactions and their items.

TID	Items
T1	A B C D
T2	A B C
T3	A B D
T4	A C D
T5	A B C
T6	B D

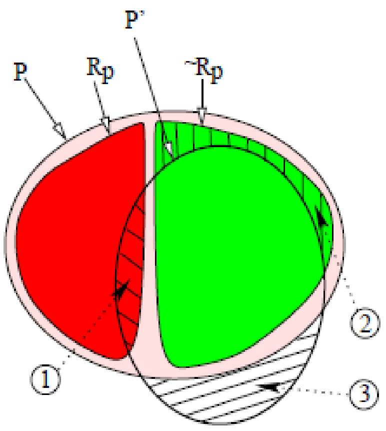


Figure (3.6) Visual Representation of restrictive & non-restrictive patterns [27].

Supporting sensitive transactions of Rp1, T_S= [T_1 , T_2 , T_5] , so the support of Rp1 = 3/6 or 50%. We should sanitize the three transactions if disclosure threshold 0% and two transactions if 50%, one if 75%. In the latter case, we do not hide Rp1 but we lower it support from 50% to 33%.

When hiding sensitive frequent itemsets (restrictive patterns Rp), three kinds of problems may appear as shown in figure (3.6):-

1. Restrictive patterns still exist in the sanitized database (number 1 in the figure).
2. Non-restrictive patterns disappear in the released database i.e. missing patterns (number 2 in the figure).
3. New set of restrictive patterns are created in the released database i.e. artificial patterns (number 3 in the figure).

All the algorithms in this section are based on removing certain information from the original database. That is why they come under the category of "**Restriction-Based Algorithms**". The rational for using this approach as mentioned by the writers is ," Algorithms that solely remove information create a smaller impact on the database since they do not generate artifacts such as illegal association rules that would not exist had the sanitization not happened"[27].

In contrast to Verykios et.al algorithms [35], these algorithms consider overlapping itemsets. If a transaction support more than one restrictive pattern, it is called **conflicting transaction**. A number associated with each sensitive transaction that measures **degree of conflict** and finds how many restrictive patterns are supported by this sensitive transaction.

All the algorithms in this section have four major steps:

1. For each restrictive pattern, identify sensitive transactions and sort them according to **degree of conflict**.
2. For each restrictive pattern, find the victim items; that is, items to be removed from the transaction.

3. Based on the **disclosure threshold** value, find the number of transactions we need to modify.
4. For each restrictive pattern, loop over the first number of transactions supporting it found in step1 remove the victim item(s) found in step 2 from them.

3.2.1 Naïve Algorithm

As mentioned above, the algorithm finds sensitive transactions for each restrictive pattern and sorts them in an ascending order according to the degree of conflict.

In step 2, for Naïve algorithm, all the items in the restrictive pattern are victims and need to be removed from the sensitive transactions. In step 3: we find the number of transactions necessary to be removed according to disclosure threshold (we call it **Num-Trans**). In step 4, we remove the victim items defined in step 2 from the first **Num-Trans** transactions defined in step1.

If the number of items in the restrictive patterns equals to number of items in the transaction, we need to leave one item since the size of the database is fixed. This item is the item that belongs to the restrictive pattern with maximum support in the database.

For example, if the restrictive pattern R_p (ABC) with item support 5, 4, 6 respectively .Suppose we have a supporting transaction T1 = [ABC]. If we need to modify T1 then it will become T1=[C] after sanitization because (C) has the maximum support.

This algorithm has the major impact on the database because as mentioned in step 2, it is based on removing all items in the restrictive pattern. The sketch of this algorithm is shown in figure (3.7).

```
Algorithm 5: **Naïve-Algorithm[27]**
Input: D, $R_p$, $\Psi$
Output: $\acute{D}$
Step 1. For each restrictive pattern $r_{pi} \in R_p$ do
    1.  T[$r_{pi}$] ← Find_Sensitive_Transactions($r_{pi}$ , D);
Step 2. For each restrictive pattern $r_{pi} \in R_p$ do
        1.  $Victim_{rpi}$ ← ∀ $item_k$ such that $item_k \in r_{pi}$
Step 3. For each restrictive pattern $r_{pi} \in R_P$ do
        1.  $NumTrans_{rpi}$ ← |T[$r_{pi}$]| × (1 - Ψ) // |T[$r_{pi}$]| is the number of sensitive transactions for
            $r_{pi}$
Step 4. $\acute{D}$ ← D
For each restrictive pattern $r_{pi} \in R_P$ do
        1.  Sort-Transactions(T[$r_{pi}$]) ; //in ascending order of degree of conflict
        2.  TransToSanitize ← Select first $NumTrans_{rpi}$ transactions from T[$r_{pi}$].
        3.  In $\acute{D}$ foreach transaction t ∈ TransToSanitize ,do
                3.1  t ← (t - $Victim_{rpi}$)
```

Figure (3.7) a sketch of Algorithm Naïve algorithm.

Example 6: Using the same dataset in Table 1 with the same support threshold values, we choose a number of frequent itemsets selectively to be sensitive.

Table (3.7) sensitive Itemsets and corresponding transactions.

Sensitive Itemset	Absolute Support	Disclosure Threshold (%) (Ψ)	Fully Supporting Transaction indices	N-iterations Required
2 , 6	2	50	[6 , 8]	1
3 , 6	2	0	[6 , 8]	2
2 , 3 , 4	2	50	[0 , 8]	1
1 , 2 , 3	3	75	[1 , 4 , 0]	1

Notice how the value of disclosure threshold determines how many transactions (N-iterations) we are going to modify. As stated above, all items in the restrictive pattern are removed from the supporting transactions. Here supporting transactions are arranged by **degree-of-conflict** in ascending order as shown in the table.

In this example, all sensitive itemsets are hidden (Hiding Failure = 0.0 %), out of ten non-sensitives (including supersets) four non-sensitive itemsets are deleted (missing costs = 40.0 %). No artificial patterns are generated (Artificial Patterns = 0.0 %). The difference between the original and sanitized database is 29.41%.

3.2.2 Minimum Frequency Item Algorithm (MFI)

This algorithm is similar to Naïve algorithm but instead of removing all items from the sensitive transaction, we just delete items with minimum support in each sensitive itemset. The rationale behind this selection is to minimize the impact on the original database and not to affect the non-sensitive frequent itemsets. A sketch of this algorithm is in Fig (3.8)

```
Algorithm 6: **MFI[27]**
Input: D, $R_p$, $\Psi$
Output: $\acute{D}$
Step 1. For each restrictive pattern $r_{pi} \in R_p$ do
            2.   T[$r_{pi}$] ← Find_Sensitive_Transactions($r_{pi}$ , D);
Step 2. For each restrictive pattern $r_{pi} \in R_p$ do
            2.1 $Victim_{rpi} \leftarrow item_v$ such that $item_v \in r_{pi}$ and $\forall\ item_k \in r_{pi}$ , sup($item_k$ , D) ≥ sup($item_v$ , D).
Step 3. For each restrictive pattern $r_{pi} \in R_P$ do
            2.   $NumTrans_{rpi} \leftarrow |T[r_{pi}]| \times (1 - \Psi)$ // |T[$r_{pi}$]| is the number of sensitive transactions for $r_{pi}$
Step 4. $\acute{D} \leftarrow$ D
For each restrictive pattern $r_{pi} \in R_P$ do
            4.   Sort_Transactions (T[$r_{pi}$]) ; //in ascending order of degree of conflict
            5.   TransToSanitize ← Select first $NumTrans_{rpi}$ transactions from T[$r_{pi}$].
            6.   In $\acute{D}$ foreach transaction t ∈ TransToSanitize ,do
            7.   t ← (t - $Victim_{rpi}$)
```

Figure (3.8) a sketch of Algorithm MFI-Algorithm.

If we use the same sensitive itemsets in example (7) and apply MFI algorithm, we get the same results as that shown in table (6) in terms of supporting transactions and number of iterations. Victim items here for each sensitive itemset are 6 , 6 , 3 , 3, respectively since these items have the smallest support among other items within the same sensitive itemset . Using this approach, all sensitive itemsets are hidden (Hiding Failure = 0.0 %), one non-sensitive itemsets is deleted (missing costs = 10.0 %) and no artificial patterns (Artificial Patterns = 0.0 %). The difference between the original and sanitized database is 11.76%.

3.2.3 Item Grouping Algorithm (IGA)

This is a more sophisticated algorithm than the previous ones because it is based on clustering the restrictive patterns we need to hide into common patterns called **restrictive group**. Each restrictive group has a **label**. The label is an item that belongs to the restrictive group and has the smallest support among other items in the group. Thus when removing a label from the conflicting sensitive transactions, we take care of more than one restrictive pattern at once. By this approach, we achieve the purpose of the hiding goal and the minimal impact on the database at the same time. However, when grouping restrictive patterns, there can be an overlapping between groups because clustering is done in a pair-wise basis and is not transitive.

For example, if R_{p1} = [ABC] , R_{p2} = [ABCD] , R_{p3} = [AC] , then R_{g1} = [ABC] , R_{g2} = [AC] , Supporting restrictive patterns of R_{g1}= { R_{p1} , R_{p2}}. Supporting restrictive patterns of R_{g2} = { R_{p1} , R_{p2}, R_{p3} }.Here { R_{p1} , R_{p2}} are found in both groups. This problem can be solved by comparing each group starting with the group of largest number of restrictive patterns and remove the common restrictive patterns from the smallest group. Thus {R_{p1} ,R_{p2}} are removed from R_{g1}.This results in only one restrictive group R_{g2} = [AC] with restrictive patterns {R_{p1} ,R_{p2} , R_{p3} }.Since R_{g1} becomes empty after comparison , we remove it. However, the restrictive groups have the same number of supporting sensitive itemsets, we remove the common restrictive patterns from the restrictive group that has the label of smallest support in the original database.

A sketch of this algorithm is in Fig (3.9).

```
Algorithm 7: Item_Grouping_Algorithm [27]
Input: D, $R_p$, $\Psi$
Output: $\acute{D}$
Step 1. For each restrictive pattern $r_{pi} \in R_p$ do
        3.   T[$r_{pi}$] ← Find_Sensitive_Transactions($r_{pi}$ , D);
Step 2.
    1.  Group restrictive patterns in a set of groups $GP$ such that ∀ G ∈ GP , ∀ $r_{pi}$ , $r_{pj}$ ∈ G, $r_{pi}$
        and $r_{pj}$ share the same itemset I.Give the class label $\alpha$ to G such that $\alpha$ ∈ I and ∀ β ∈ I ,
        sup($\alpha$ , D) ≤ sup(β,D).
    2.  Order the groups in GP by size in terms of number of restrictive patterns in the groups.
    3.  Compare groups pairwise $G_i$ and $G_j$ starting with the largest. For all $r_{pk} \in G_i \cap G_j$ do
        3.1  İf size($G_i$) ≠ size($G_j$) then remove $rp_k$ from smallest($G_{i\,,}$ $G_j$ )
        3.2  Else remove $r_{pk}$ from group with class label $\alpha$ such that sup($\alpha$ , D) ≤ sup(β,D) and  , β
             are class labels of either $G_i$ $or$ $G_j$ .
    4.  For each restrictive pattern $r_{pi}$ ∈ GP do
            4.1   $Victim_{rpi}$ ← $\alpha$ such that $\alpha$ is the class label of G and $r_{pi}$ ∈ G.
Step 3. For each restrictive pattern $r_{pi} \in R_P$ do
        3.1 $NumTrans_{rpi}$ ← |T[$r_{pi}$]| × (1 - Ψ) // |T[$r_{pi}$]| is the number of sensitive transactions for
$r_{pi}$
Step 4. $\acute{D}$ ← D
For each restricitve pattern $r_{pi} \in R_P$ do
    1.  Sort_Transactions(T[$r_{pi}$]) ; //in descending order of degree of conflict
    2.  TransToSanitize ← Select first $NumTrans_{rpi}$ transactions from T[$r_{pi}$].
    3.  In $\acute{D}$ foreach transaction t ∈ TransToSanitize ,do
        3.2  t ← (t - $Victim_{rpi}$)
```
Figure (3.9) a sketch of Algorithm Item-Grouping-Algorithm

Example 7: In this example, we use the same restrictive patterns in the previous example and we apply IGA to hide them.

Table (3.8) sensitive itemsets, support, and corresponding transactions.

Sensitive Itemset ID	Sensitive Itemset	Support	Disclosure Threshold (%) (Ψ)	Fully Supporting Transaction indices	N-iterations Required
1	2 , 6	2	50	[8 , 6]	1
2	3 , 6	2	0	[8 , 6]	2
3	2 , 3 , 4	2	50	[8 , 0]	1
4	1 , 2 , 3	3	75	[0 , 1 , 4]	1

The Table 8 is similar to the one in example (6) with the same disclosure threshold values. The difference is that transactions are sorted in a descending order according to degree of conflict. The following table shows the created groups.

Table (3.9) created groups before comparison

Group ID	Supporting Sensitive Itemsets indices	Common Items	Label
2	1 , 3 , 4	[2]	2
3	2 , 3 , 4	[3]	3
1	1 , 2	[6]	6
4	3 , 4	[2,3]	3

Groups are sorted in order of supporting sensitive itemsets starting from the largest. The following table shows groups after comparison.

Table (3.10) groups of table (3.9) after comparison

Group ID	Supporting Sensitive Itemsets	Common Items	Label
2	1 , 3 , 4	[2]	2
3	2	[3]	3

We compare each pair of the resulting groups and if there is an intersection in supporting sensitive itemsets, we remove these common sensitive itemsets from the smallest group. If they have the same length, we remove them from the group, which has the smallest label.

In the next step, for each restrictive pattern belonging to the remaining groups, we remove victim items (label of the group) from the supporting transactions as shown Table 3.10. Application of IGA to the previous example results in removing items 2, 3, and [2, 3] from transactions 0, 6, 8, respectively. The evaluation result shows that one sensitive itemset is not hidden (actually, it is the fourth one because its disclosure threshold is high 0.75). Thus hiding failure = 0.25%. No non-sensitive itemsets is missing (missing costs = 0.0 %) and no artificial patterns (Artificial Patterns = 0.0 %). The difference between the original and sanitized database is 11.765 %.

3.3 Protective Sensitive Knowledge with Data Sanitization

While the previous algorithms do not consider the non-sensitive itemsets during sanitizations, these algorithms do. In these algorithms, for every sensitive transaction, a ratio between supporting sensitive and non-sensitive itemsets is calculated. When modifying a sensitive transaction, we take first the transaction with maximum sensitive/ non-sensitive ratio to sanitize. This ensures hiding maximum number of sensitive itemsets with the minimum effect on the non-sensitive ones [2].

3.3.1 Aggregate Approach

In this algorithm, sensitive and non-sensitive frequent itemsets are defined. Database is scanned to find sensitive transactions. Then, while there are sensitive frequent itemsets, and for each sensitive transaction found in the previous step, find the number of sensitive and non-sensitive frequent itemsets supported by this transaction and the sensitive / non-sensitive ratio. Take the transaction that has the maximum ratio, and remove it utterly from the database. We then reduce the support of every sensitive and non-sensitive itemset supported by that transaction by one. If their support is less than the minimum support, we remove them from the list of the frequent sensitive/ non-sensitive itemsets. These steps continue until no frequent sensitive itemset is there. A pseudo-code of this algorithm is shown in Fig (3.10)

```
Algorithm 8:Aggregate [2]
Step 1: Initialization:
    • The sanitized database $\mathbf{D'}$ equals the original database $\mathbf{D}$.
    • Determine the set $\boldsymbol{D^C}$ of all sensitive transactions that support the sensitive itemsets.
    • Determine the support $S_j$ of every sensitive and nonsensitive itemset j ϵ F = S U N (Recall
      that F is the set of all frequent itemsets, S is the set of sensitive itemsets, and N is the set of
      non-sensitive itemsets, and that initially $S_j \geq$ S ∀ j ϵ F).
    • The set $\overline{\boldsymbol{S}}$ of sensitive itemsets in $\mathbf{D'}$ with support ≥ S equals the set of all sensitive itemsets S
      and the set $\overline{N}$of non-sensitive itemsets in $\mathbf{D'}$ with support ≥ S equals the set of all non-
      sensitive itemsets N.
Step 2: While there is still a sensitive itemset with support ≥ S (i.e., $\overline{\boldsymbol{S}} \neq$ Ø) do the following:

1. for each transaction k in $\boldsymbol{D^C}$ do
    • Determine the number ($\boldsymbol{b_k}$) of sensitive itemsets in(i.e., with support $\overline{\boldsymbol{S}}$) that are supported by
      k
    • Determine the number ($\boldsymbol{a_k}$) of non-sensitive itemsets in $\overline{\boldsymbol{N}}$ (i.e., with support ≥ S) that are
      supported by k.
    • Determine the ratio $\boldsymbol{f_k}\ \frac{\boldsymbol{b_k}}{\boldsymbol{a_k}}$ if $\boldsymbol{a_k}$ > 0 else $f_k$= $\boldsymbol{b_k}$.

2. Select the transaction $\boldsymbol{K^*}$ ϵ $\boldsymbol{D^C}$ such that $\boldsymbol{f_k^*}$ = max { $\boldsymbol{f_k}$: k ϵ $\boldsymbol{D^C}$}.
3. Update:
    • Remove the transaction $K^*$ from $D^C$ and from $\mathbf{D'}$ .
    • Reduce the support of every itemset jϵ $\bar{S}$ U $\bar{N}$ that is supported by $K^*$ by 1; i.e., $S_j$ = $S_j$ – 1.
    • Remove every sensitive itemset from $\bar{S}$ if its support < S.
    • Remove every non-sensitive itemset from $\bar{N}$ if its support < S.
```

Figure (3.10) a sketch of Algorithm Aggregate.

When applying Aggregate approach to same chosen sensitive itemsets of example (7), this results in removing transactions 9, 2 , 5 from the database. The result is concealing all sensitive itemsets (Hiding Failure 0 %) and no missing non-sensitives (Missing Costs 0 %). No artificial patterns are generated and the difference between source and sanitized databases is 29.41%.

3.3.2 Disaggregate Approach

This approach is similar in structure to the Aggregate algorithm, but instead of removing the entire transaction, we just remove the item in the sensitive transaction whose removal results in reducing support of the maximum number of sensitive itemsets and minimum number of non-sensitive itemsets by one. We take the union of all sensitive transactions and find the victim item inside the sensitive transaction. After deleting this item from the sensitive transaction, we reduce the supporting sensitive and non-sensitive itemsets by one and delete itemsets whose support goes below minimum threshold. The algorithm stops when there are no sensitive frequent itemsets. A pseudo-code of this approach is in Fig (3.11)

Algorithm 9: Disaggregate [2]

Step 1: Initialization:

- The sanitized database $\mathbf{D'}$ equals the original database **D**.
- Determine the set $\boldsymbol{D^C}$ of all sensitive transactions that support the sensitive itemsets.
- For every item *i* in every sensitive transaction k, set $\bar{t}_{ik} = t_{ik}$(this is used to keep track of the items still included in each sensitive transaction during the sanitization process, where the parameter
  $$t_{ik} = \begin{cases} 1 & if\ item\ i\ \in I\ is\ included\ in\ transaction\ k\varepsilon D \\ 0 & otherwise \end{cases})$$
- Determine the support S_j of every sensitive and nonsensitive itemset j ϵ F = S U N (Recall that F is the set of all frequent itemsets, S is the set of sensitive itemsets, and N is the set of non-sensitive itemsets, and that initially $S_j \geq$ S ∀ j ϵ F).
- The set $\boldsymbol{\bar{S}}$ of sensitive itemsets in $\mathbf{D'}$ with support ≥ S equals the set of all sensitive itemsets S and the set $\bar{N}$of non-sensitive itemsets in $\mathbf{D'}$ with support ≥ S equals the set of all non-sensitive itemsets N.

Step 2: While there is still a sensitive itemset with support ≥ S (i.e., $\boldsymbol{\bar{S}} \neq \emptyset$) do the following:

1. For each transaction k in $\boldsymbol{D^C}$ (i.e., $\bar{t}_{ik}$ = 1) do

- Determine the number ($\boldsymbol{b_{ik}}$) of sensitive itemsets $\boldsymbol{\bar{S}}$ in whose support will decrease by 1 if item *i* is removed from transaction k.
- Determine the number ($\boldsymbol{a_{ik}}$) of non-sensitive itemsets in $\boldsymbol{\bar{N}}$ whose support will decrease by 1 if item *i* is removed from transaction k.
- Determine the ratio $\boldsymbol{f_{ik}}$ $\frac{\boldsymbol{b_{ik}}}{\boldsymbol{a_{ik}}}$ if $\boldsymbol{a_{ik}}$ > 0 else $f_{ik} = \boldsymbol{b_{ik}}$.

2. Select the item $\boldsymbol{i^*}$ in the transaction $\boldsymbol{K^*}$ ϵ $\boldsymbol{D^C}$ such that $\boldsymbol{f_i^*}$ $\boldsymbol{k^*}$ = max{ $\boldsymbol{f_{ik}}$: i∈ I , k ∈ $\boldsymbol{D^C}$ such that $\bar{t}_{ik}$ = 1**}.**

3. Update:

- Remove item i^* from the transactionK^***.**
- Reduce the support of every itemset j ∈ $\bar{S}$ U $\bar{N}$ that contains item i^* and is supported by K^* by 1; i.e., $S_j = S_j - 1$.
- Remove every sensitive itemset from $\bar{S}$ if its support < S.
- Remove every non-sensitive itemset from $\bar{N}$ if its support < S.

Figure (3.11) a sketch of Algorithm Disaggregate.

Application of Disaggregate algorithm to the same example (7), results in concealing all sensitive itemsets (Hiding Failure 0 %) and no non-sensitives missing (Missing Costs 0 %). No artificial patterns generated and the difference between source and sanitized databases is 8.82%.

3.3.3 Hybrid Approach

This approach is a combination of the previous two approaches. First, we use the aggregate approach to find the sensitive transactions. From the result, we choose the sensitive transaction that affects the maximum number of sensitive itemsets and minimum number of non-sensitive itemsets. Then Disaggregate approach is used to find the victim item from the transaction found in the previous step.

The criterion of choosing victim items is the same as that of choosing the transaction.

Application of Hybrid algorithm to example (7), results in concealing all sensitive itemsets (Hiding Failure 0 %) and no non-sensitives missing (Missing Costs 0 %). No artificial patterns generated and the difference between source and sanitized databases is 11.77 %.

3.4 Sliding Window Algorithm (SWA)

In this algorithm, hiding is done on sensitive association rules level i.e. we are going to hide sensitive association rules instead of sensitive itemsets. The main idea behind SWA is similar to IGA. In this algorithm, for each K transactions in the database (K is the window size) we find sensitive transactions and sort them in ascending order of size. Then for each sensitive transaction, we count the frequency of each item that belongs to the sensitive rules. Items inside each transaction are then sorted in a descending order of their frequencies [26].

In the next step, for each restrictive association rule, we select the victim item from each supporting sensitive transaction. Victim item here is the item that has the maximum frequency since it is shared by more than one sensitive rule and thus its removal guarantees the minimum effect on the database. According to these results, for each sensitive rule, we take the first Num-Trans transactions and delete the victim items from them.

These steps ensure that the support of sensitive rules will be reduced to a value less than the minimum support threshold.

A sketch of Sliding Window Algorithm is in Fig (3.12).

Algorithm 10: **Sliding-Window-Algorithm** [26]

```
Input: D, $M_p$ , K
Output: $\acute{D}$
For each K transactıons ın D do {
Step 1. For each transaction t ϵ K do {
    1. Sort the items in t in ascending order
    2. For each association rule $rr_i$ ϵ $M_p$ do {
       If ∃ $rr_i$ such that ∀ j $item_j$ ϵ $rr_i$ and $item_j$ ϵ t then
       2.1 T[$rr_i$] ← T[$rr_i$] U t // t is sensitive
       2.2 Freq($item_j$) ← Freq($item_j$) + 1
       }
}
Step 2.If t is sensitive then
    1. Sort_Vector(Freq) // in descending order
    2. For each association rule rr$_i$ ϵ $M_p$ do
       2.1 Select $item_v$ such that $item_v$ ϵ $rr_i$ and ∀ $item_k$ ϵ $rr_i$ ,freq[$item_v$] ≥ freq[$item_k$]
       2.2 If freq[$item_v$] > 1 then $victim_{rr_i}$ ← $item_v$
           Else $victim_{rr_i}$ ← randomly selected $item_k$ such that $item_k$ ϵ $rr_i$
Step 3. For each association rule $rr_i$ ϵ $M_p$ do
    1. $NumTrans_{rr_i}$ ← |T[$rr_i$]| × (1- $\Psi_i$) // |T[$rr_i$]| Number of sensitive transactions for $rr_i$ in K
Step 4. For each association rule $rr_i$ ϵ $M_p$ do
    1. Sort_Transactions(T[$rr_i$]).// in ascending order of size
Step 5. $\acute{D}$ ← D
For each association rule $rr_i$ ϵ $M_p$ do {
    1. TransToSanitize ← Select first $NumTrans_{rr_i}$ transactions from T[$rr_i$]
    2. In $\acute{D}$ foreach transactiont ϵ transToSanitize do
       2.1 t ← (t - $victim_{rr_i}$) // transaction is santized
       2.2 if $\Psi_i$ = 0 then do look_ahead($rr_i$ , $victim_{rr_i}$ , t, $M_p$ )
}
```

Figure (3.12) a sketch of Algorithm Sliding-Window-Algorithm.

3.4.1 Look Ahead Procedure

In SWA, every restrictive rule has a disclosure threshold. A procedure called, **Look-Ahead procedure** is used when the disclosure threshold of the rule at hand is 0%. This procedure is used to see if we need to sanitize a sensitive transaction more than once. In this case, we look for the remaining sensitive rules starting from the current one to see if the transaction at hand has been selected to be sensitive by any other sensitive rule. If so, and the victim item is also part of the other restrictive rule, we remove the current transaction from the supporting transactions list of the other rule since it has already been sanitized and thus the support of the other rule has already reduced .

The rationale behind this procedure is to reduce the missing costs of the algorithm. As a concrete example, let T1 = [ABCD] be a transaction supporting two restrictive rules R_{r1} = AB $\rightarrow$ C, R_{r2} = AB $\rightarrow$ D and the disclosure threshold for R_{r1} = 0% and R_{r2} = 50%, if A is the victim item for R_{r1} in transaction T1. After removing A from transaction T1, we remove T1 from supporting transaction list of R_{r2} since the support of R_{r2} has already reduced by removing A from the same transaction.

Example 8: In this example, we use the same dataset in Table 1. We also choose the same rules in example (1) to be sensitive.

Table (3.12) rules, transactions, support, confidence, and victim items.

ID	Sensitive Rule	Support	Confidence (%)	Disclosure Threshold (Ψ)	Fully Supporting Transaction indices	N-iterations Required	Victim Item in transactions Respectively
R_{r1}	8 $\rightarrow$ 2	2	100	50	[6 , 7]	1	2
R_{r2}	2 , 3 $\rightarrow$ 1	3	60	0	[1 , 4 , 0]	3	[2^*,1^* ,1]
R_{r3}	3 , 4 $\rightarrow$ 1	2	66.7	50	[3]	1	1
R_{r4}	6 $\rightarrow$ 2 , 3	2	100	75	[6 , 8]	1	2

Supporting transactions are sorted in ascending order of size. The reason is, as mentioned before, to reduce the impact of sanitization on the non-sensitive data. We see the victim item is the first element that has the maximum frequency. If the sensitive transaction supports only one sensitive rule, we choose the element at random , as we see in the R_{r2} case (the first victim elements are chosen at random).

In this example, we can see the effect of look-ahead procedure. When we sanitize R_{r2} , we remove items[2 , 1,1] from transactions [1,4,0] respectively. Disclosure threshold of this rule is 0%, so we apply look-ahead procedure in the next set of rules in which any [1, 4, 0] transactions support. Since sensitive rule R_{r3} is supported by the sensitive transaction {0} and includes the chosen item to remove {1}, so we delete transaction {0} from R_{r3} supporting trnasactions list.

For evaluation, no hiding failures (Hiding failure = 0.0%), a number of non-sensitive rules are lost (missing cost = 33.33%), one new rule is generated (Artificial rules = 4.35), difference between source and modified databases = 11.77

CHAPTER 4: EXPERIMENTAL RESULTS

4.1 Software Description

All the algorithms were coded in Visual Studio 2013 with C#. For Frequent itemset generation, we used FPgrowth algorithm. To generate interesting association rules, we used Agrawal94. Both algorithms are implemented in java using SPMF library [39]. We also used IKVM.openJDK.Core library to link java codes in .Net environment.

We run the algorithms on an Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz (4 CPUs), ~2.6GHz with 8084MB RAM available memory. Operating system is Windows 8.1 Single Language 64-bit (6.3, Build 9600) (9600.winblue_r7.150109-2022).

4.2 Description of the Real Datasets

The first dataset we used was Mushroom dataset from UCI Machine Learning repository [42]. Number of transactions in this dataset is 8124 with 119 items and 23.0 average transaction lengths. In these experiments, we used two minimum supports; 0.25 (absolute support =2031) and 0.32 (absolute support = 2600). The first minimum support gives us 5510 non-singleton frequent itemsets and the second one gives us 1920 non-singleton frequent itemsets.

The second dataset is Retail Market Basket Data from an anonymous Belgian retail store [40]. This dataset consists of 88162 transactions and 16470 different items with average transaction length of 11. We used two minimum supports: 0.0008 (absolute support = 123.43) and 0.0014 (absolute support = 70.53). The first minimum support gives us 7703 non-singleton frequent itemsets and the second one gives us 3231 non-singleton frequent itemsets.

Third dataset is Chess Dataset from UCI repository [41]. It consists of 3196 transactions and 75 different items with 37 average transaction lengths. Here also we used two minimum supports: 0.8 (absolute support = 2556.8) and 0.9 (absolute support = 2876.4). The first minimum support gives us 8208 non-singleton frequent itemsets and the second one gives us 609 non-singleton frequent itemsets.

From each dataset, we chose 13 and 25 different random itemsets to be sensitive. However, this resulted in other itemsets to be sensitive since any superset of the sensitive itemset is also sensitive. Similarly, we chose 25 and 50 different rules to be sensitive from the results of significant rules in each dataset. Results are shown in table (4.2).

Table (4.1) Datasets Description

Dataset Name	# Transactions	# Items	Average Transaction Length
Mushroom	8124	119	23.0
Chess	3196	75	37
Retail	88162	16470	11

Table (4.2) Datasets with chosen parameters

Dataset Name	Relative Min Support (%)	Absolute Min-Support	Min Confidence	Non Singleton Frequent İtemsets	# strong rules	# Sensitive Rules (with supersets)	# Sensitive Itemsets (with supersets)
Mushroom	25	2031	66	5510	183,103	25(26152)	13 (803)
						50(36539)	25 (1208)
	32	2599.68		1926	33079	25(1327)	13 (217)
						50(2342)	25(480)
Chess	80	2556.8	95	8208	145035	25(12258)	13(1264)
						50(40100)	25(2774)
	90	2876.4		609	6855	25(746)	13(97)
						50(2057)	25(188)
Retail	0.08	70.53	30	7703	11176	25(116)	13 (28)
						50(907)	25 (66)
	0.14	123.43		3231	4528	25(72)	13 (15)
						50(208)	25 (37)

4.3 Measuring Effectiveness

We summarize the performance measures as follows [26, 27]:

- **Hiding Failure (HF):** measures the amount of restrictive association rules (or sensitive itemsets) that are disclosed after sanitization. Hiding failure is measured by $HF = \frac{\# R_R(D0)}{\# R_R(D)}$ where $\# R_R$ (X) denotes the number of restrictive association rules discovered from database X.
- **Misses Cost (MC):** measures the amount of legitimate association rules that are hidden by accident after sanitization. It is calculated as follows: MC = $\frac{\#\sim R_R(D) - \#\sim R_R(D')}{\#\sim R_R(D)}$ where $\# \sim R_R(X)$ denotes the number of non-restrictive association rules discovered from database X. We can define Data utility as:

  Data Utility (%) = 1 – Missing Costs (%).

- **Artificial Patterns** (AP): measure the artificial association rules created by the addition of noise in the data. Artificial patterns are measured as AP = $\frac{\#|R'|-|R \cap R'|}{|R'|}$ where |X| denotes the cardinality of X.
- **Difference between the original and sanitized databases**: the difference between the original (D) and the sanitized (D') databases, denoted by dif(D , D'), is given by:

Diff (D,D') =$\frac{1}{\sum_{i=1}^{n} f_D(i)} * \sum_{i=1}^{n}[f_D(i) - f_{D'}(i)]$ where $f_X(i)$ represents the frequency of the i_{th} item in the dataset X and n is the number of distinct items in the original dataset. This measurement is also called item-level accuracy [2] as opposed to transaction-level accuracy that is the number of transactions in the original dataset that remains intact after sanitization. We do not consider this later measurement in our study.

4.4 Evaluation

When evaluating these sanitization algorithms, we consider the following points:

- Sensitive itemsets and sensitive rules are chosen at random.
- Sensitive itemsets are subset of frequent itemsets whose size is greater than one (called non-singleton frequent itemsets).
- Supersets of any sensitive itemset are also sensitive and we should hide them. Similarly, association rules containing restrictive rules are also sensitive and we should hide them [6 , 7].
- All the algorithms hide sensitive itemsets except Alg.1a, Alg.1b and Alg.2.a; these algorithms hide sensitive rules. We also used two variations of SWA algorithms: one for hiding sensitive rules and the other for hiding sensitive itemsets.
- The higher the support/confidence of the sensitive itemset/rule, the greater the effect on the sanitized database.
- For those algorithms, that use disclosure threshold, we take the disclosure value at 0.0% since this is the worst case and all sensitive itemsets should be removed.
- Amiri algorithms [2] are the slowest among these heuristic approaches.
- We used the same sensitive itemsets for those algorithms that are based on sensitive itemset hiding and the same sensitive rules for those algorithms that are based on sensitive rule hiding.
- For all experiments, we set minimum confidence at 0.66, 0.95 and 0.30 for Mushroom, Chess and Retail datasets respectively with association rule hiding algorithms.

4.4.1 Association Rule Hiding Algorithms Evaluation

In the first experiment, we set minimum supports at 0.25, 0.80, and 0.08 for Mushroom, Chess and Retail datasets respectively. We randomly select 25 different rules from each dataset. Since any association rule containing sensitive rule is also sensitive ,the choice results in 26152 sensitive rules from Mushroom, 12258 sensitive rules from Chess and 116 sensitive rules from Retail. Effects of sanitizing these rules using association rule hiding algorithms are shown in Fig (4.1)

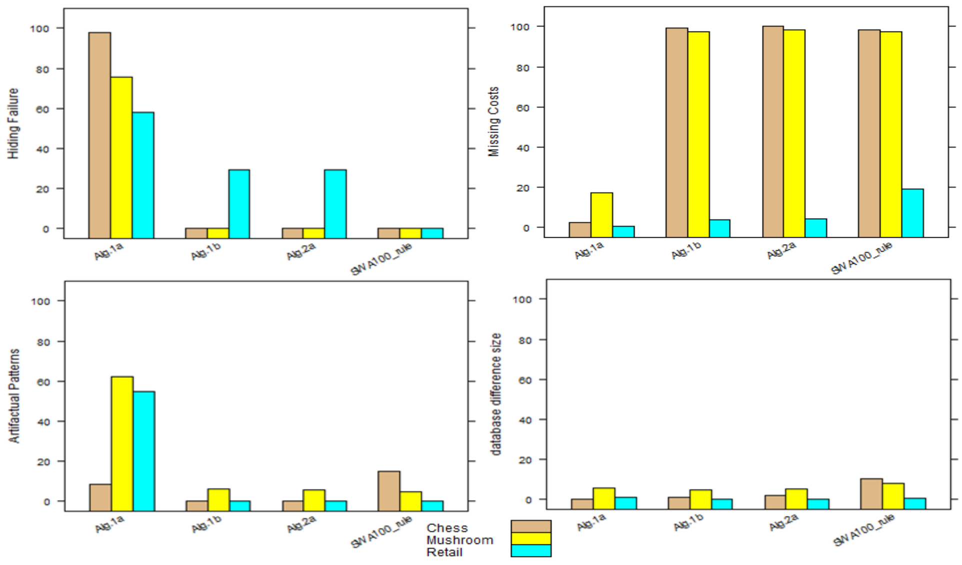


Figure (4.1) Effect of sanitization when # sensitives rules = 25

In another experiment, we increased sensitive rules to 50. These result in 36539, 40100, and 907 different rules to be sensitive from Mushroom, Chess, and Retail datasets respectively. Results are shown in Fig (4.2)

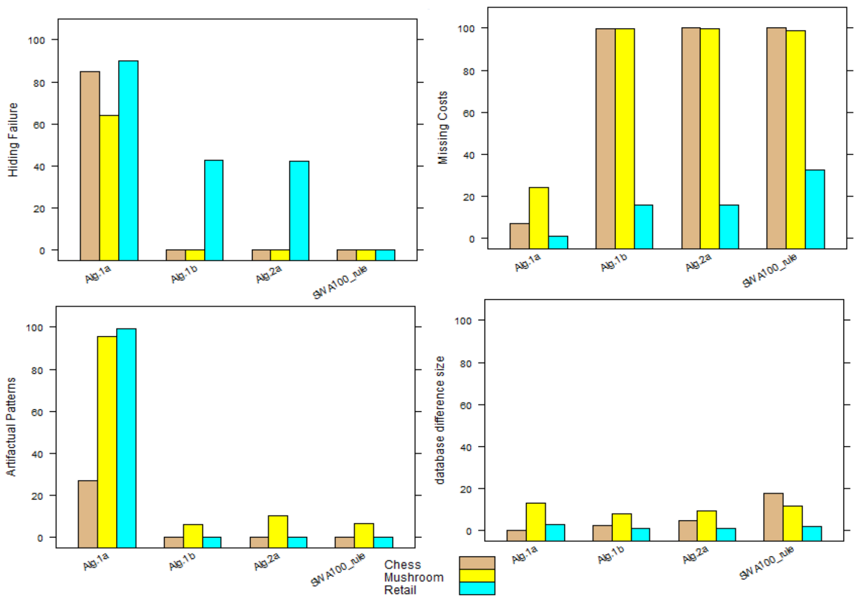


Figure (4.2) Effect of sanitization when # sensitives rules = 50

From Fig.1 and Fig.2, we see that algorithm 1.a has the worst hiding failure and artificial patterns performance. At the same time, this algorithm has the best missing costs performance. The main reason is that this algorithm is based on adding elements to partial supporting transactions as we mentioned in chapter (2). From these figures also , we see that there is a trade-off between artificial patterns and missing costs , the larger the missing costs , the smaller the artificial patterns and vice versa.

In another experiment, we increased minimum supports by setting them at 0.32, 0.90, and 0.14 for Mushroom, Chess, and Retail, respectively. When choosing 25 different sensitive rules, this makes 1327, 746 and 72 rules to be sensitive for each dataset, respectively. Results are shown in Fig (4.3).

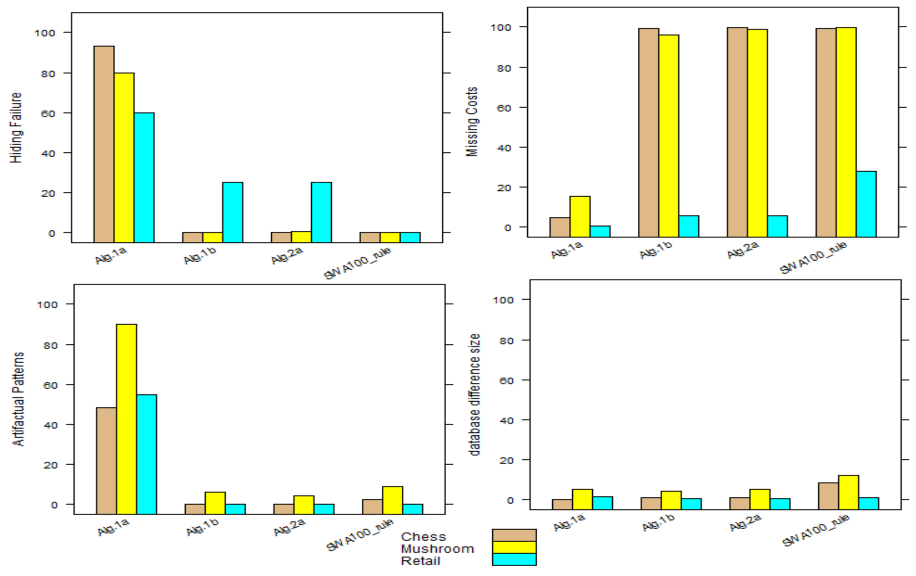


Figure (4.3) Effect of sanitization when # sensitives rules = 25

Under the same experiment, we increased number of sensitive rules to 50 to see effects of sanitizing algorithms on the chosen datasets. This resulted in 2342, 2057, and 208 different rules to be sensitive from Mushroom, Chess and Retail datasets, respectively. Results are show in Fig. (4.4).

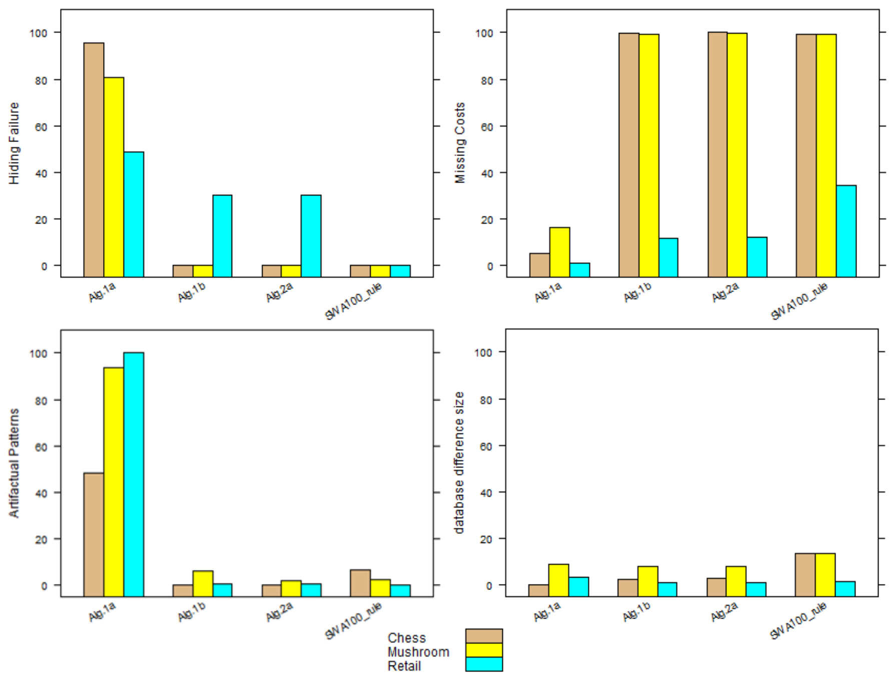


Figure (4.4) Effect of sanitization when # sensitives rules = 50

4.4.2 Itemset Hiding Algorithms Evaluation

To evaluate itemset-hiding algorithms, we used the same approach and the same measurements. We set minimum supports at 0.25, 0.80 and 0.08 for Mushroom, Chess and Retail, respectively. Then from the generated frequent itemsets, we chose 13 different itemsets to be sensitive. This choice results in 803, 1264, and 28 itemsets to be sensitive since any superset of a sensitive itemset is also sensitive as we stated before. Fig (4.5) shows the results.

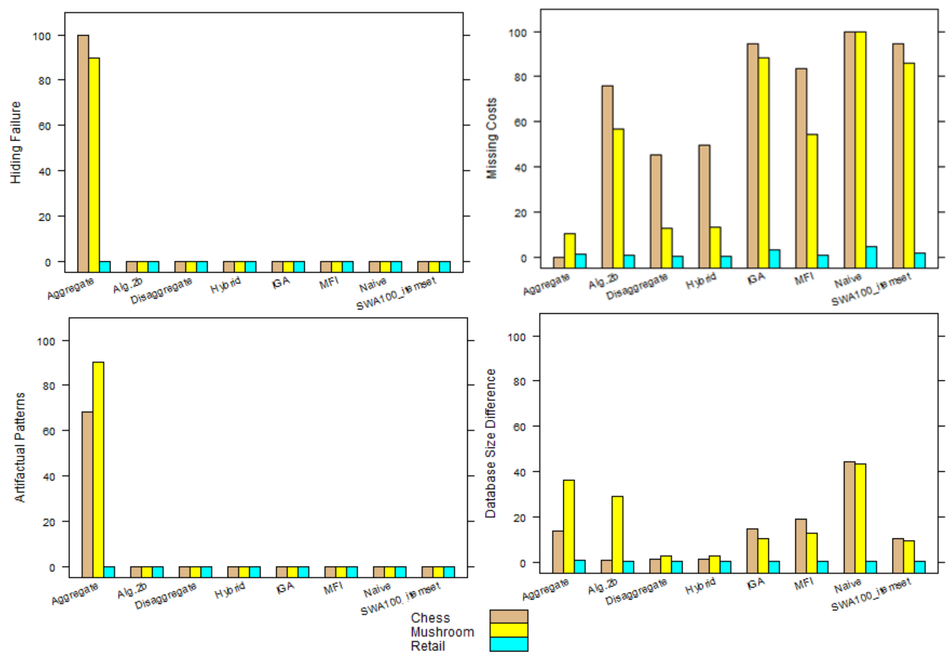


Figure (4.5) Results of hiding sensitive itemsets on datasets

We also increased number of sensitive itemsets to 25 using the same support thresholds. These result in 1208, 2774, and 66 different itemsets to be sensitive from Mushroom, Chess and Retail, respectively. Results are shown in Fig.4.6.

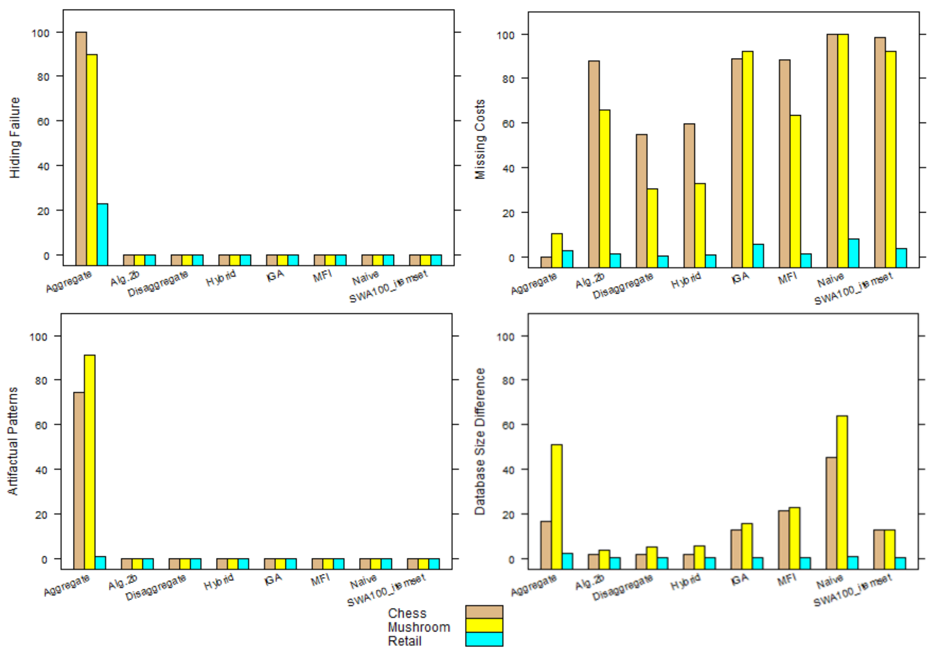


Figure (4.6) Results of hiding sensitive itemsets on datasets.

We made two other experiments. First, we set minimum support thresholds at 0.32, 0.90, and 0.14 for Mushroom, Chess and Retail, respectively. In the first experiment, we set number of sensitive itemsets at 13. This results in 217, 97, and 15 different itemsets to be sensitive from each Mushroom, Chess, and Retail respectively. Results are shown in Fig (4.7)

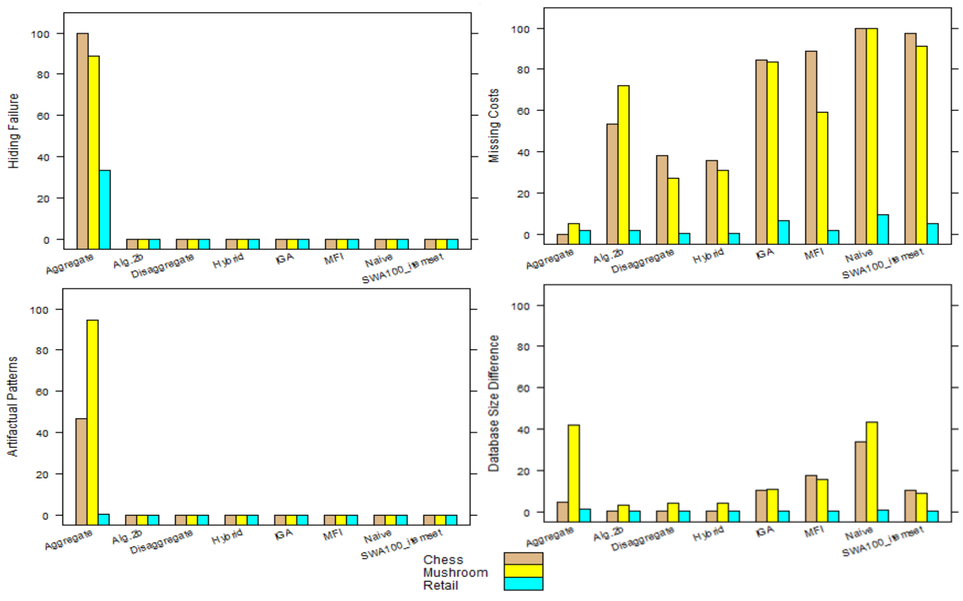


Figure (4.7) results of hiding sensitive itemsets on datasets

In the second experiment, we increased number of sensitive itemsets to 25 using the same support threshold. The consequence is 480, 188, and 37 different itemsets to be sensitive. Measurements results are shown in Fig (4.8)

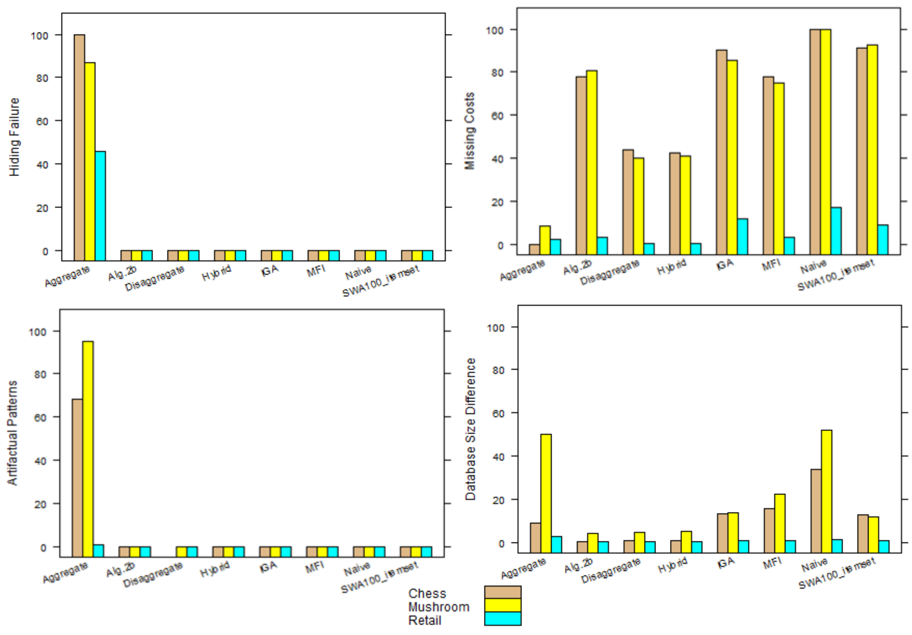


Figure (4.8) results of hiding sensitive itemsets on datasets

From these experiments, we see that Aggregate algorithm has the worst hiding failure and artificial pattern performances. In contrast, it has the best missing costs performance. The main reason is that, Aggregate approach hides sensitive itemsets by deleting transactions that have the maximum sensitive / non-sensitive ratio as we stated before. Consequently, Transaction-deletion changes database size. This may make non-frequent itemsets to become frequent and hidden sensitive ones to haunt back. To explain this point, let us take the following example. Suppose a dataset D whose size |D| = 50. Suppose minimum support σ = 20% (absolute support = 10 transactions). If a frequent itemset X with support s(X) = 30% (absolute support = 15) is sensitive. Using Aggregate approach, we should delete at least six transactions from D in order to hide X. When we consider again the support of X with the new dataset size i.e. |D| = 44. Support of X, S(X) = 20.46% .That means it becomes frequent again. From this point, we may conclude that size of datasets should not be changed as support and confidence thresholds are constant.

Disaggregate and Hybrid algorithms have the best performances in almost all measures for the three selected datasets. The reason is that Disaggregate approach successively deletes the item whose removal reduces the maximum number of sensitive itemsets and the minimum number of non-sensitive ones. However, this approach requires many calculations. The problem is worse with denser datasets where number of sensitive transactions approximates database size and number of sensitive/ non-sensitive itemsets is very large. This makes Disaggregate approach the slowest despite its good performance. The same thing also applies for Hybrid approach, which uses Aggregate and Disaggregate combinations.

Missing costs seems to be high for approximately all algorithms especially for Mushroom and Chess datasets. The reason is the high correlation between elements in such datasets. Application of our discussed algorithms shows that these algorithms do well on sparser datasets (like Retail dataset) and they have side effects on denser ones (dataset density is measured by average transaction length divided by number of items) [2].

For those algorithms that use disclosure threshold i.e. Naive, MFI, IGA and SWA, we used disclosure threshold at 0.0% in order to remove sensitive itemsets. That is why, in all these algorithms the hiding failure is zero for all cases. Artificial patterns are also zero because they are based on items deletion. However, these deletions become unnecessary when support of sensitive itemset becomes lower than minimum support threshold. Consequently, the missing costs rates are very high.

To explain this point further, let us take the following example using the simplest algorithm of these; Naive algorithm. Suppose a dataset ***D*** with size |***D***| = 50 with minimum support 20% (absolute support = 10). A sensitive itemset, ***X*** with support S(X) = 30% (absolute support = 15). Using Naïve algorithm with disclosure threshold = 0.0% and from equation (5) in chapter 2, required number of iteration is 15. At every iteration all elements in itemset X are removed from each transaction supporting X. These deletions affect non-sensitive itemsets. The final support of X is zero while only six transactions are needed to make this itemset hidden. We elaborate this remark further in the following section

4.4.3 Why Disclosure Threshold Should Not Be Zero?

In Oliveria-Zaiane algorithms, they used disclosure threshold to measure how private the sensitive itemset should be. When disclosure threshold of a sensitive itemset is 0.0% that means we should remove it from resultant dataset and when 100%, that means there is no restriction on revealing that sensitive itemset. However, during evaluation, we saw that when disclosure threshold value was lower than minimum support threshold value, there were unnecessary deletions because the corresponding itemset has already become non-frequent. Consequently, this may greatly affect non-sensitive itemsets and database dissimilarity measurements. Minimum support threshold is already a pruning value for any non-frequent itemset and should be used as a lower bound of disclosure threshold value that is sufficient to hide sensitive itemsets.

Fig.(4.9) shows effects of disclosure threshold (φ) on hiding failure , misses costs and dissimilarity measurements at 0.25 , 0.80 and 0.08 support thresholds for Mushroom, Chess and Retail, respectively using SWA algorithm with K=100. Numbers of sensitive itemsets are 13 as in Table 2. We see that hiding failure is approximately zero for up to 50% of the three datasets and then started increasing up to 100%. That means 50% of disclosure threshold was sufficient to hide selected sensitive itemsets. Lower values of disclosure thresholds severely affect non-sensitive itemsets and dissimilarity measurement especially with denser datasets like Mushroom and Retail.

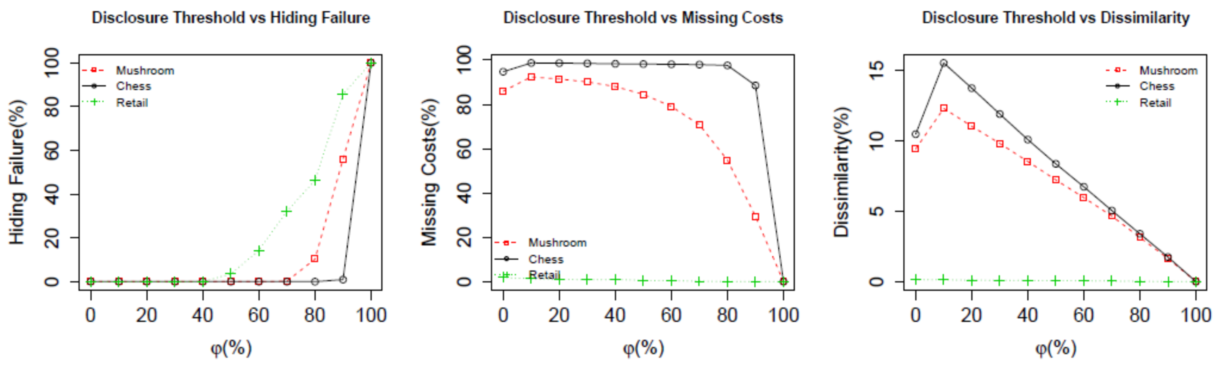


Figure (4.9) Effects of φ on hiding failure, Misses Costs and Dissimilarity using SWA at k=100

We do not mention artificial patterns measurement because its value is zero in all cases with the three datasets. Note also that values of missing costs and dissimilarities at 0.0% are lower than following values. This is mainly due to look -ahead procedure in SWA which is used only at 0.0% disclosure threshold .This procedure alleviates missing costs, and dissimilarity at 0.0% as mentioned in chapter (3).

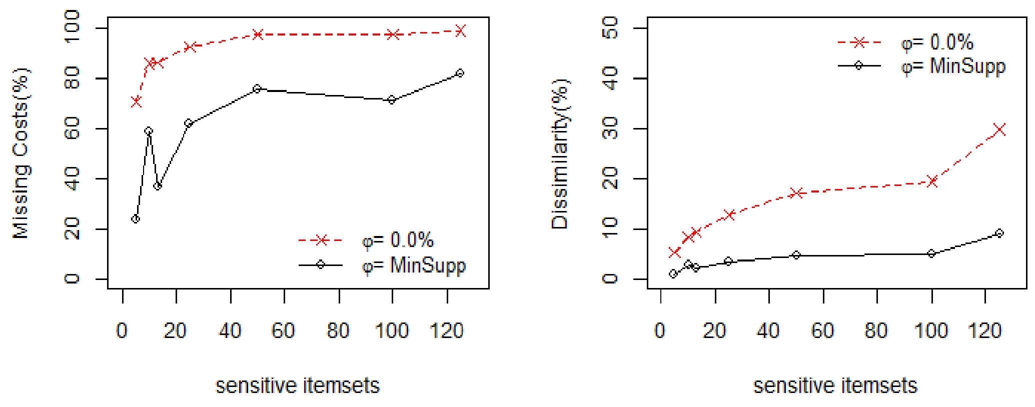


Figure (4.10) Effects of φ at 0.0% and min_supp in Mushroom Dataset using SWA at k=100

In Fig (4.10), we examined SWA at K=100 using different number of sensitive itemsets ranging from 5 up to 125. We set disclosure values once at 0.0% and once more at min-supp value. Dataset was Mushroom at 0.25 minimum supports. Results show that in both cases hiding failures and artificial patterns were zero. However, effects on missing costs and dissimilarity were better with using min-supp value than 0.0 as is clearly seen in fig (4, 10).

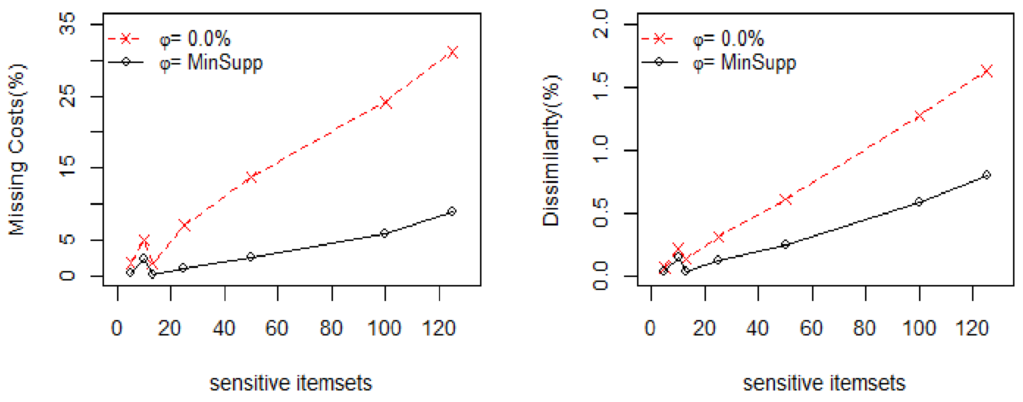


Figure (4.11) Effects of φ at 0.0% and min-supp in Retail Dataset using SWA at k=100

In Fig (4.11), we used the same experiment using Retail dataset at 0.08 minimum support thresholds. Here also hiding failure and artificial patterns were zero in all cases. However, in case of missing costs and dataset differences, disclosure threshold with min-supp value was better than that at 0.0 values.

CHAPTER 5: CONCLUSION AND FUTURE WORK

In this thesis, we addressed one important side effect of data mining which is privacy. In particular, we investigated Privacy Preserving Data Mining which is a new promising field that allows sharing data, mining them collaboratively and at the same time preserving privacies of database owners.

We discussed a number of approaches to solve association rule hiding problem and applied them on a number of publically published datasets. The first set of algorithms we discussed devised by Verykios et.al [12]. The merit of these algorithms is that they are the first to address the ARH problem by modifying support and confidence of the sensitive patterns. These algorithms propose two strategies to hide sensitive patterns. The first strategy is by decreasing confidence of the sensitive rule by either increasing the support of the rule antecedent or decreasing the support of the rule consequent. The second strategy depends on decreasing support of the sensitive rule by removing items from transactions that support rule both antecedent and consequent of the rule. A major drawback of these algorithms is their assumptions to hide sensitive knowledge.

Oliveira and Zaiane devised the second set of algorithms we discussed. The advantage of these algorithms is their relaxation of Verykios et.al assumptions, the introduction of disclosure threshold concept and the framework suggested to speed up the sanitization process [27]. These algorithms rely on four steps to hide sensitive knowledge. In the first step, sensitive transactions for each sensitive itemset are identified and sorted according to their degree of conflict. In the next step, for each sensitive pattern, itemsets to be deleted are defined and then number of sensitive transactions necessary for sanitization is calculated depending on the disclosure threshold. Finally, victim items found in step two are removed from the first number of transactions found in the previous step. Experimental results show that among this set of algorithm, IGA show the best performance in terms of missing costs and database dissimilarity. The reason is that this algorithm groups sensitive patterns sharing common items and delete these common items from the conflicting transactions starting with transactions with the largest degree of conflict. This ensures taking care of more than one sensitive pattern at a time.

The same authors suggest a fast efficient algorithm called Sliding Window Algorithm (SWA). The idea of SWA is similar to IGA. In this algorithm, for each sensitive transaction, the victim item is the item with the highest frequency. The frequency of item is the number of its occurrences in the sensitive patterns supported by sensitive transaction at hand. The contribution of this algorithm is the introduction of sliding window concept that reduces impact of sanitization on the original database. This parameter controls the number of transactions necessary for scanning. Another contribution is the look-ahead procedure. This procedure is used to alleviate impact of victim item removal on the non-sensitive knowledge [26].

The final set of algorithms are Aggregate, Disaggregate and Hybrid approaches. The major contribution of these algorithms is that they take the effect of sanitization on the non-sensitive knowledge into account. This is done by finding number of sensitive and non-sensitive patterns supported by each sensitive transaction [2]. Although these algorithms require a lot of computation, they model best the role of association rule hiding.

From the experiments, we made to validate these algorithms; we conclude that 1.a and Aggregate approaches have the worst hiding failure and artificial patterns performances. In contrast, they have the best missing costs performance. The reason is that algorithm 1.a is based on reducing sensitive rule's confidence by adding elements of rules antecedents to partial supporting transactions. Since significant rules in Mushroom and Chess have high confidence values, total number of partial transactions found is less than N-iteration necessary to reduce sensitive rule confidence to a value less than confidence threshold. Besides, addition results in generating new patterns. Aggregate approach, on the other hand, is based on transactions deletion that have maximum sensitive/non-sensitive ratio. Transaction deletion results in changing database size while min-sup is fixed. This causes removed sensitive itemsets to haunt back. We also see that Naive, MFI, IGA and SWA algorithms have the worst missing cost performance because of the unnecessary deletions. These algorithms use disclosure threshold as a measure of sensitivity of the frequent pattern. In our experiments, we used 0.0% disclosure value. This means, deleting the whole occurrences of the sensitive patterns in the original datasets even if they go below support threshold. This is the same reason why we got zero-hiding failure and zero-artificial patterns. We may conclude here that if the lower value of disclosure threshold be minimum support threshold, performance of these algorithms will be better.

Experimental results also showed that Disaggregate, Hybrid, and 2.b algorithms have the best performance using measures mentioned in chapter 3. However they require a lot of computations .The computational complexity of disaggregate and Hybrid is O ($|F|^2|I||D^C|$) and the computation complexity of the Aggregate approach is O ($|F|^2|D^C|$) where F denotes frequent itemsets, I denotes number of items and D^C sensitive transactions [2]. For those algorithms that use disclosure threshold, Naive, MFI, IGA and SWA, we suggested that minimum support threshold be used as a lower bound of disclosure threshold in order to avoid unnecessary modifications.

For our future work, we have already started studying border-based approaches namely BBA, MaxMin1 and MaxMin2 algorithms. As stated in chapter 2, these algorithms are based on the border theory. We implemented these algorithms and our results show that these algorithm are really better than pure heuristic approaches discussed in chapter 3. We have already written a paper “Towards Association Rule Hiding, Heuristics vs Border-based Approaches”. Our next intention is exact approaches as these algorithms show even better empirical results than heuristics and border-based approaches. In the near future, we are going to examine exact approaches and study their effects.

CURRICULUM VITAE

Personal Information

First Name:	Afrah
Last Name:	Farea
Gender:	Female
Marital Status:	Married
Nationality:	Yemen

Contact Information

Permanent Address :	Al-luhum mahallesi, Al-salam istasyon arkasında, Najib apt, daire 2, Aden, Yemen
Telephone Number:	00967 770692304
Current Address :	Zafer mah. Sağlık cad.Rifatoğlu apt. daire 6 Battlegazi Malatya,Turkey
Telephone Number:	00905512558947
Telephone Number:	00905364494013
Email:	othello2009@hotmail.com; afrah.nacib@gmail.com

Scholarships:

Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı ,YTB

Publications :

[1] A.Farea, A.KARCI, Applications of Association Rules Hiding Heuristic Approaches, SİU-2015, Turkey

REFERENCES

[1] C.C.Aggarwal, Outlier Analysis,Springer,New York 2013

[2] A. Amiri. Dare to share: Protecting sensitive knowledge with data sanitization. Decision Support Systems, 43(1):181–191, 2007.

[3] O.Abul,M.Atzori,F.Bonchi,F.Giannotti,Hiding Sequences,IEEE,2007

[4] M.Ankerst, Ma.M. Breunig, H.P.Kriegel, J.Sander,OPTICS: Ordering Points To Identify the Clustering Structure,Proc. ACM SIGMOD'99 Int. Conf. on Management of Data, Philadelphia PA, 1999.

[5] C.C. Aggarwal, J.Han, Frequent Pattern Mining,Springer,2014

[6] R.Agrawal, R.Srikant,Fast Algorithms for Mining Association Rules,IBM Almaden Research Center, 1994

[7] R. Bayardo. Efficiently mining long patterns from databases. Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, 1998

[8] P.B. Cerrito, Introduction to Data Mining Using SAS Enterprise Miner ,SAS institute, 2006

[9] V.Chandola, A.Banerjee ,V.Kumar, Anomaly Detection: A Survey,ACM,2009

[10] C.Clifton,D.Marks, Security and Privacy Implications of Data Mining,ACM SIGMOD workshop on Data Mining and Knowledge Discovery,1996

[11] T.G.Dietterich,Ensemble Methods in Machine Learning,Oregon State University, Oregon, USA, 2006

[12] A.G.Divanis,V.S.Verykios.,Association Rule Hiding for Data Mining,vol 41,Springer,2010

[13] M.Ester,H.P.Kriegel,J.Sander,X.Xu, Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,KDD,1996

[14] L.K.Grover,R.Mehra,The Lure of Statistics in Data Mining,Journal of Statistics Education,vol 16, 2008.

[15] C.Glymourg,D.Madigan,D.Rregibon, P.Smyth,Statistical Themes and Lessons for data Mining,Data Mining and Knowledge Discovery 1, 11–28 (1997)

[16] A. Gkoulalas-Divanis , V. Verykios. An integer programming approach for frequent itemset hiding. In Proceedings of the 15th ACM International Conference on Information and Knowledge Management, CIKM, pages 748–757, 2006

[17] D. Hawkins. Identification of Outliers, Chapman and Hall, 1980

[18] D. Hand, H. Mannila,and P. Smyth, Principles of data mining, vol. 30, no. 7. ,MIT press, 2001.

[19] J.Han,M.Kamber,Data Mining: Concepts and Techniques,2nd Edition,Elsevier,2006

[20] J.Han,M.Kamber,J.Pei,Data Mining Concepts and Techniques,3d edition, Morgan Kaufmann Publishers,2012

[21] J.Han, J.Pei and Y. Yin. Mining Frequent Patterns without Candidate Generation, ACM,SIGMOD Conference, 2000.

[22] D.Kuonen, “Data mining and Statistics: What is the connection?”, The Data Administrative Newsletter, Switzerland.2004

[23] T.Kanungo,N.S.Netanyahu,A.Y.Wu,An Efficient k-Means Clustering Algorithm:Analysis and Implementation,IEEE transaction on pattern analysis and machine intelligence, VOL. 24, NO. 7, JULY 2002

[24] N.R.Mabroukeh,C.I.Ezeife,A Taxonomy of Sequential Pattern Mining Algorithms,ACM,2010

[25] S. Menon, S. Sarkar, and S. Mukherjee. Maximizing accuracy of shared databases when concealing sensitive patterns. Information Systems Research, 16(3):256–270, Sept. 2005.

[26] S. R. M. Oliveira , O. R. Za¨ıane. An Efficient One-Scan Sanitization For Improving. The Balance Between Privacy And Knowledge Discovery.University of Alberta, Canada, June 2003.

[27] S. Oliveira, O. Zaiane, Privacy preserving frequent itemset mining, Proceedings of the IEEE ICDM Workshop on Privacy,Security and Data Mining, pp. 43–54, Maebashi City, Japan , December 2002.

[28] S.Oliveira, O.Zaïane, Privacy preservation when sharing data for clustering,Proceedings of the Int. Workshop on Secure Data Management in a Connected World,Vol.1, pp. 67-82, Toronto, Canada,2004

[29] J.S.Park, M.Chen,P.Yu,"An Effective Hash-Based Algorithm for Mining Association Rules," ACM SIGMOD Record archive, 24(2), 175 – 186,1995

[30] J.Roberto, Jr.Bayardo,Efficiently Mining Long Patterns from Databases,ACM-SIGMOD Int’l Conf. on Management of Data, 85-93,1998

[31] A.Savasere, E.Omiecinsk,S.Navathe,An Efficient Algorithm for Mining Association Rules in Large Databases,Proceedings of the 21st VLDB Conference Zurich, Swizerland, 1995

[32] Y. Saygin, V. Verykios, and C. Clifton. Using unknowns to prevent discovery of association rules. ACM SIGMOD Record, 30(4):45–54, Dec. 2001.

[33] P.N.Tan, M.Steinbach, V.Kumar,Introduction to Data Mining,Pearson Addison Wesley,2005

[34] J.Vaidya,C.Clifton, Privacy-preserving data mining: why, how, and when,IEEE,2004

[35] V. S. Verykios, A. K. Emagarmid, E. Bertino, Y. Saygin, and E. Dasseni. Association rule hiding. IEEE Transactions on Knowledge and Data Engineering, 16(4):434–447, 2004.

[36] M.J.Zaki,K.Gouda,Fast Vertical Mining Using Diffsets,2001

[37] M.Zaki,W.M.Jr,Data Mining and Analysis Fundamental Concepts and Algorithms,Cambridge university press,May 2014

[38] T.Zhang, R.Ramakrishnan, M.Livny,BIRCH: An Efficient Data Clustering Method for Very Large Databases,SIGMOD ’96 6/96 Montreal, Canada,1996.

[39] http://www.philippe-fournier-viger.com/spmf/

[40] http://recsyswiki.com/wiki/Grocery_shopping_datasets

[41] https://archive.ics.uci.edu/ml/datasets/Chess (King-Rook+vs.+King-Pawn)

[42] https://archive.ics.uci.edu/ml/datasets/Mushroom

[43] http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/KDD3.htm

[44] X. Sun and P. S. Yu. Hiding sensitive frequent itemsets by a border–based approach. Computing science and engineering, 1(1):74–94, 2007.

[45] G. V. Moustakides and V. S. Verykios. A max–min approach for hiding frequent itemsets. In Workshops Proceedings of the 6th IEEE International Conference on Data Mining (ICDM), pages 502–506, 2006.

[46] G. V. Moustakides and V. S. Verykios. A maxmin approach for hiding frequent itemsets. Data and Knowledge Engineering, 65(1):75–89, 2008.