Incremental Association Rule Mining Based on Intermediate Complete Itemset

By

Iyad Ahmad Izzideen Aqra

Supervisor

Dr. Fadi Thabtah

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies

Philadelphia University

January 2013

جامعة فيلادلفيا
نموذج تفويض

أنا إياد أحمد عزالدين أقرع، أفوض جامعة فيلادلفيا بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

التوقيع : إياد أقرع
التاريخ : 2013/1/24

Philadelphia University
Authorization Form

I am, Iyad Ahmad Izzideen Aqra, authorize Philadelphia University to supply copies of my thesis to libraries or establishments or individuals upon request.

Signature: Iyad Aqra
Date: 24/1/2013

Incremental Association Rule Mining Based on Intermediate Complete Itemset

By

Iyad Ahmad Izzideen Aqra

Supervisor

Dr. Fadi Thabtah

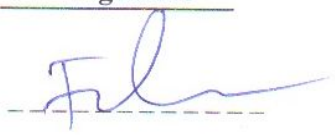
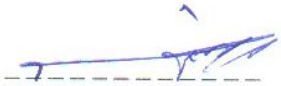
This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies

Philadelphia University

January 2013

Successfully defended and approved on 31/12/2012

Examination Committee Signature		Signature
Dr. Fadi Thabtah Academic Rank: Associate Professor	, Chairman.	
Dr. Rashid Al-Zubaidy Academic Rank: Associate Professor	, Member.	
Dr. Arafat Awajan Academic Rank: Associate Professor (Princess Sumaya University for Technology)	, External Member.	

Dedication

I fully dedicate this thesis:

To my family: father, mother, wife, brother and sisters and their families,,

To my professors and all my friends,,

To all Palestinians, specifically the people of my town Qabalan,,

To Palestine, and AL-Quds,,

To all who provide moral support, and supported me,,

To all humanity,,

To Myself,,

Iyad Aqra

Acknowledgment

Thanks ALLAH firstly, and before everything and after everything for giving me the knowledge and ability to complete this work in this final form.

I would like to thank our university to offer scientific nutrition necessary to complete our study and our research. I would like to express my sincere thanks to professor's staff in the college who provide a warm and lively environment to encourage and help graduate students to grow in their graduate study. Especially, Prof. Saeed AL-Goul for his support of the educational process and increase quality.

I would like to extend my regards and sincere gratitude to Dr. Fadi Fayez, who has guided me through my work from the beginning and supported me. To all of these who has helped and encouraged me.

I would like to thank my family, for their unconditional love and support, for encouraging me to pursue graduate studies and for always being there to share and to help. Their love, patience and support to me during my graduate study never ceased, and always be there whenever I need help.

I thank you for your love and support, and dedicate this work to you.

Iyad Aqra

Table of Content

List of Figureures

List of Tables

List of Abbreviations	
ARM	Association Rule Mining
INAP	Incremental Apriori
DB	Database
db+	Modified database
KDD	Knowledge Discovery in Database
Conf	confidence
MinConf	Minimum Confidence
supp	Support
MinSupp	Minimum Support
C_i	Candidate set length i
L_i	frequent itemset length i
Tid-List	Transaction identifier list
FUP	Fast Update Algorithm
NFUP	New Fast Update Algorithm
IMSC	Incremental Maintenance of association rules under Support threshold Change algorithm
MAAP	Maintenance Association rule with Apriori Property
KD	Last Knowledge Date extraction
LTA	Add Transaction List
LTU	Update Transaction List
LTD	Delete Transaction List

Abstract

Association rule mining (ARM) is one of the most important tasks in data mining that has attracted a lot of attention in the research community. The Apriori algorithm provides a creative and an intelligent way to find association rule on large database scale. Apriori is one of the most important algorithms, which aims to explore association rules. The main problem associated with Apriori is the multi scan database requires to find the rules when data gets updated every time. This problem increases complexity when databases grow over time. The discovered results from the original data are needed when mining the modified data set verifying knowledge obtained earlier.

Researchers have proposed many algorithms to deal with the incremental problem. Especially in applications were changing databases constantly like banking application. These algorithms created solution to the problem in an intelligent way, such as FUP, IMSC, MAAP algorithms. When the existing incremental learning approaches are reviewed, we found some defects such as:

(1) They do not take all data manipulation operations, specifically the update operation.
(2) These algorithms rescan database many times.
(3) Some of these algorithms discover knowledge without Frequency rate.

The proposed algorithm in this thesis is called Incremental Apriori (INAP), and it deals with the problems described above. It is an incremental ARM that doesn't need to rescan old database when gets update. The algorithm takes all data manipulation operation including the modifying, deleting and adding transactions into account when mine the data set and without going back to iterate over the original dataset. INAP algorithm allows us to extract knowledge with different thresholds (rule strength rate) every time without the needs to iterate over the original database meaning it solve the incremental problem in association rule.

Chapter 1

Introduction

2.5. Motivation

Nowadays, data are collected in large quantities up to several megabytes or gigabytes, which made the task of generating useful knowledge complicated especially in the decision making related to business. This drew the attention of several researchers in the field of database on the production of knowledge extracted from there large dataset that could help decision makers. So they can make decision automatically (Bao H. T., 1998).

The (Agrawal and Srikant, 1994) made the introduction of Association Rule Mining (ARM) in 1993, to generate knowledge from large transactional data in supermarkets. The proposed method by Agrawal (Apriori) had nontrivial serial iteration. Association rule mining is an important task in data mining that finds correlations between (Knowledge) items in a database, in the form "IF-THEN" rule. The classic application for association rule mining is market basket analysis (Han, et al., 2000; Rajak , and Gupta,2012) in which business experts aim to discover the shopping behavior of customers. Finding association rules help the decision maker to discover rules which work for shelving of products in supermarket, in order to increase sale and help customers locate them. For instance, in a supermarket, if a customer buys grapes, what is the probability that s/he buys a pack of juice as well? Using such Knowledge, marketing experts can develop strategic decisions concerning shelving, and planning. ARM has been widely used in various industries beside supermarkets such as mail order (Li, 2005), telemarketing (Xue, & Zhu, 2009), banking and visa card (Rajak , and Gupta,2012;Kobsa, et al., 2006), customer relationship management

(Rajak , and Gupta,2012), and e-commerce (Zhang, 2011), medical diagnosis, protein sequence (Rajak, and Gupta,2012). Extraction of knowledge is closely linked correlation with the presence of data, whenever data is found; extraction knowledge is also found (ARM).

In association, many algorithms and techniques have appeared to extraction knowledge but mostly rely on three main things: Candidate set, support and confidence threshold. When any researcher to provide new improvements to the algorithms or to provide a new algorithm, we must put in the same matters of importance, including reducing the time required for the extraction of knowledge, as well as providing I/O time, also to be a high accuracy output. Reducing scanning time in incremental learning is the first motivation of this thesis. When a new transaction arrives and/or an old transaction leaves most classic approaches ignore the delete, and update operations. So, when knowledge is generated from the updated database (db+) it must rescan the input database to reflect the changes, unless the algorithm kept the old results obtained from the original DB. On the other hand, incremental learning associated with allowing the user to get the most recent updated frequent itemset, without rescanning the entire updated database (old, and new). This saves not only the computation time, but more importantly, the I/O time to load and write data from database to memory. In addition, this approach can handle the insertion, deletion and updateing operations independently. Furthermore, in my approach the user can generate different frequent itemset, with support for each one, without the need to rescan original database.

2.6. **Knowledge Discovery in Databases (KDD)**

(Fayyad, et al, 1996) has defined knowledge discovery in database to be" the non trivial extraction of impact, previously unknown and potentially useful information in data". Knowledge discovery as a process is depicted in Figure (**Figure** 1.1), and consists of an iterative sequence of the following steps (Han, and Kamber, 2000), where data mining is one of its primary phase.

Hereunder is brief description of the KDD main processes:-

Data cleaning: to remove noise or irrelevant data, incorrect or incomplete records like invalid values, or n null can be fixed by end users, or void those recode, during the data cleansing step.

- Data integration: where multiple data from different sources may be combined.
- Data selection: where data relevant to the analysis task are retrieved from the database.
- Data transformation: where data are transformed or consolidated into forms appropriate or mining by performing summary or aggregation operations, for instance.
- Data mining: an essential process where intelligent methods are applied in order to extract data patterns.

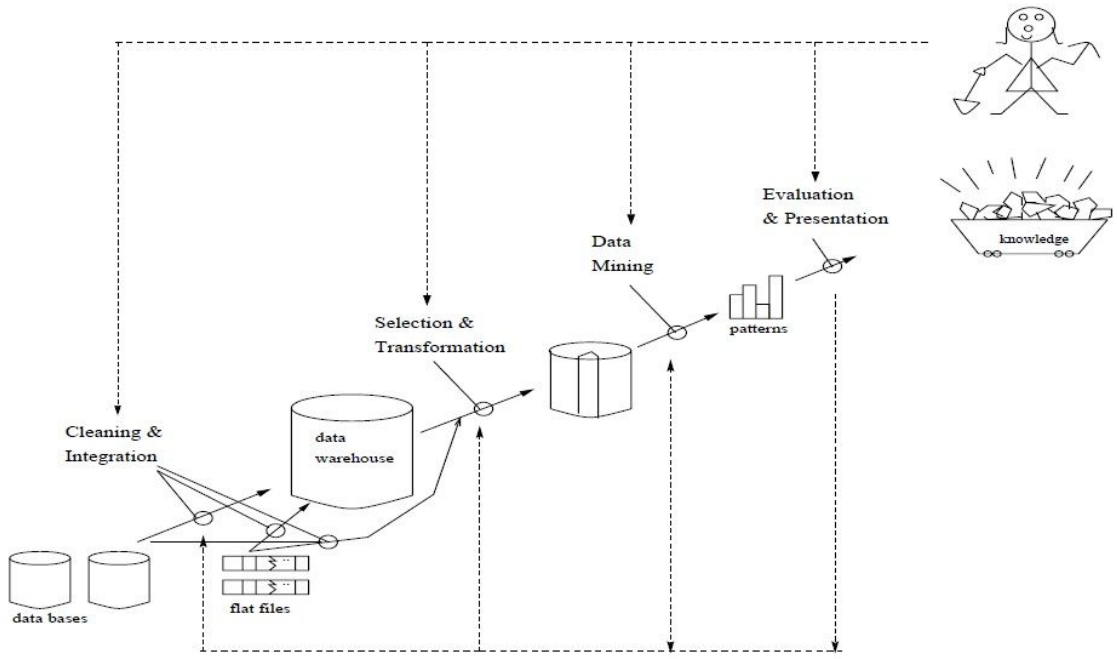


Figure 1.1: Data mining as a process of knowledge discovery (Han J. , Kamber M., 2000)

- Pattern evaluation: to identify the truly interesting patterns representing knowledge based on some interestingness measures.
- Knowledge presentation: where visualization and knowledge representation techniques are used to present knowledge.

2.6.1. Data Mining Main Concept

Since 1980s, many organizations have generated a large amount of machine-readable data in the form of files and databases. To process this data, we have the database technology available to us that supports query languages like SQL. The problem with SQL is that it is a structured language that assumes the user is aware of the database schema (Bao H. T., 1998). SQL supports operations of relational algebra that allow a user to select from tables (rows and columns of data) or join related information from tables based on common fields.

Thus, we need techniques, to simplify data, to become readable and understandable to the user. Data mining refers to the discovery or extracting of new information (knowledge) in terms of frequent itemset from large amounts of data. Practically useful, data mining must be carried out efficiently on large files and databases. “To date, it is not well-integrated with database management systems”. (Elmasri, and Navathe, 2001).

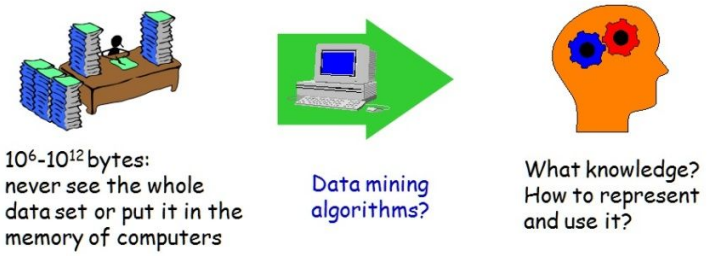


Figure 1.2: discovery Knowledge (Bao H. T., 1998).

Many researchers treat data mining as one of the main phases in Knowledge Discovery in Databases (KDD) (Fayyad, et al., 1998). Alternatively, others view data mining as simply an essential step in the process of knowledge discovery in databases.

2.6.2. Types of Knowledge Discovered during Data Mining

It is only after such preprocessing that data mining techniques are used to mine data to produce rules and patterns. For example, the results of the mining phase may be one of the following:

- Association rules- What items customers are likely to buy together. e.g., whenever a customer buys video equipment, then he or she also buys another electronic gadget.
- Sequential patterns- e.g., suppose a customer buys a camera, and within three months he or she buys photographic supplies, and within six months an accessory item. A customer who buys more than twice in the lean periods may be likely to buy at least once during feast day.
- Classification (Labeled data)- e.g., customers may be classified by frequency of visits, by types of financing used, by amount of purchase, or by affinity for types of items, and some revealing statistics may be generated for such classes.
- Clustering- When objects to be divided into groups, clustering applies. It is the process of grouping objects with certain similarities, where the similarities between the resulting clusters are minimized.

2.7. Association Rule Mining (ARM)

ARM is one of the most important and well researched techniques of data mining. ARM discovers interesting relationships among items in a given dataset, showing attribute-value conditions that occur frequently together in a given set of data. The database is regarded as a collection of transactions, each involving a set of items. Association analysis is widely used for market basket or transaction data analysis. This process analyzes the market basket corresponds to what a consumer buys in a supermarket during one visit. Consider four transactions in below table (Table 1.1):

Table 1.1 : Data sample in transactions database

Id	Items-Brought
101	milk, bread, juice
792	milk, juice
1130	milk, eggs
1735	bread, cookies, coffee

An association rule is of the form,

$$X \Rightarrow Y \quad \text{……………………………………..…………. (1.2)}$$

$X \subset I$
$X \subset I$
$I \quad Items$

Where X, and Y, are sets of items. The intersection between X, Y must be empty ($X \cap Y = \emptyset$). This association states that if a customer buys X, he is also likely to buy Y. In general, any association rule has the form LHS (left-hand side) named antecedent, RHS (right-hand side) named consequence, where LHS and RHS are sets of items (Agrawal and Srikant, 1994). Every itemset has support (frequency), where the rule has confidence (strength), and support. The support for the itemset is the percentage of transactions that hold all of the items. So that, if the support of itemset is low, it implies that there is no overwhelming evidence that items occurring together, because it only happens in a small fraction of transactions.

For support and confidence there are two values, called minimum support, minimum confidence inputted by end user. If the itemset have support greater than Minimum support then the itemset called frequent itemset, otherwise, the itemset will be avoided. When generating the rule calculate the rule confidence is considered, if it is greater than minimum confidence then the rule called strong rule and will be derived. Otherwise it is deleted. The confidence measure means the strength of a rule and it corresponds to statistical significance (Agrawal and Srikant, 1994).

Confidence value refer to ratio between the ($X \cup Y$).support and A.support

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} \quad \text{……. (1.3)}$$

$$Supp\ (X) = \frac{Number\ of\ transaction\ contain\ (X)\ item}{Total\ transactions\ numbers} \quad \ldots\ldots\ldots (1.4)$$

For example if the rule X ⇨ Y has confidence = 66.7% (meaning that, two from three transactions in which X occurs, contain Y item). According to (Agrawal and Srikant, 1994), the problem of discovering all association rules from a database can be decomposed into two sub-problems according to:

- Step 1. The generation of all itemsets with support greater than the *minimum support* threshold. These itemsets are called frequent itemsets. All other itemsets are called infrequent and are discarded. Many researchers unanimously, consider step (1) the main important problem, since the generations frequent itemset need to iterative steps such as creating candidate itemset and scan database for each itemset (multi scan) to ascertain the itemset is frequent or no.
- Step 2. For each frequent itemset generated in Step1, produce all rules that pass the *minconf* threshold. For example if item XYZ is frequent, then we might evaluate the confidence of rules $XY \rightarrow Z$, $XZ \rightarrow Y$ and $YZ \rightarrow X$.

Frequent itemset mining (Step 1) is a crucial step of the process, and its computational efficiency strongly impacts the overall performance of mining association rules (Agrawal and Srikant, 1994). Generating rules by using all frequent itemsets and their supports is relatively straightforward. Discovering all frequent itemsets together with the value for their support is a crucial and a major problem (Elmasri, and Navathe, 2001). So in this study focuses on how to discover the large frequent itemset and generate the rules incrementally.

2.8. **Incremental Association Rule Mining problem**

Most algorithms for mining association rules run in the so called batch mode, i.e. they are intended to analyze the whole available transaction database. When new transactions are added to the database, the rule discovery process needs to be restarted from start point or scratch by applying any ARM algorithms such as Apriori algorithm, Eclat algorithm

(Zaki, and Gouda, 2003), and FP-Growth Pattern (Ha, et al, 2000). There are many incremental ARM methods such as: FUP (Cheung, and et at, 1996), NFUP (Chang, and et. At, 2005), IMSC (Bachtobji, & Gouider, 2006), MAAP (Zhou and Ezeife, 2001), aims to enable an incremental discovery process, which is composed of possibly many subsequent runs. In each run mining covers a portion of transactions, which have been accumulated so far or modified transactions. When association rules are found in the current data, they are added to the knowledge base, using an incremental algorithm, which ensures, that the resulting set of rules is highly similar to the one, which would be obtained if the whole transaction database was analyzed and extracted from scratch. These algorithms don't go over (scan) the database again to update the rule when the original database is updated.

The knowledge discovery process using incremental technique in three main steps (Dudek , Zgrzywa,2005):

1. Generating all rules, for the existing portion of transactions which is updated by any ARM algorithm, without rescan the old data under any case.
2. When new transactions are added, modified or deleted, new frequent itemsets or update existing set are needed to be generated or removed, from changes done over the source database incrementally.
3. The analyzed facts are disposed. We assume that input data - transaction portions are in an acceptable format for a mining algorithm. The final step is to represent knowledge in rules form.

The most complicated step in the previous three steps is step2. For that, our approach streamlines the extraction of the frequent itemset, incrementally and handles the different types of operations occur on the database (insert, delete, and update).

Incremental association rule mining approach, which can keep the last results obtained after mining the source database and only consider data records that have been updated. A more efficient approach are can lead to a huge saving in computational time.

To precisely explain the incremental problem in association rule, consider a database T, the following operations may occur on T:

- The source database T can be incremented by T+ records (adding).
- T- records can be removed from the source database T (deleting).

- T+ records can be added to T and T- records can be removed from T (updating).

The result of any of the operations described above is an updated database T'. It should be noted that the update operation is more difficult than insert/delete operations since it combines them both. The question is how the outcome (rule/knowledge) of the original database T can be updated to reflect the changes done on T without having to perform extensive computations. We suggest that this problem can be divided into sub-problems according to the possible itemsets (attribute value) contained in T after performing data operations such as insert, update or edit. For example, itemsets in T can be divided into the following groups after inserting new records (T+):

- itemsets that are frequent in T and T+
- itemsets that are frequent in T and not frequent in T+
- itemsets that are frequent in T+ and not frequent in T
- itemsets that are neither frequent in T+ nor T

The itemsets in groups 1 and 2 can be identified in a straightforward manner. For instance, if itemsets Y is frequent in T, then it's support count in the updated training data (T'), Y'count = Ycount + Y+count, where Ycount is known and Y+count can be obtained after scanning T+. The challenging problem is to find frequent itemsets that are not frequent in T but frequent in T+ since these itemsets are not determined after scanning T or T+-. Once all above frequent itemsets are determined, the generation of the incremental rules is easy because no database scan is involved.

2.9. **Study Objectives**

Since the introduction of ARM (Agrawal and Srikant, 1994), it has continued to be an active research area in the data mining and machine learning communities. Association rule mining is an important task in data mining that finds correlations between items in a database. The classic application for association rule mining is market basket analysis (Agrawal and Srikant, 1994) (Han, et al., 2000), in which business experts aim to investigate the shopping behavior of customers in an attempt to discover regularities. In finding association rules, one tries to find groups of items that are frequently sold together

in order to infer certain items from the presence of other items in the customer's shopping cart.

Most of the existing ARM algorithms including FP-Growth (Han, et al., 2000), Apriori (Agrawal and Srikant, 1994), Horizontal Format Data Mining With Extended Bitmaps (Alwis, el at,2012), Partitioning (Brin, et al., 1997) and others mine the input transactional database as whole in order to find frequent itemsets and produce the knowledge. When data manipulating operations (adding, deleting and editing) occur on the source database, current algorithms have to scan the complete transactional database one more time in order to reflect changes done. Further, since data are collected in most application domains such as retail supermarkets, banking, etc, on a daily, weekly or monthly basis, the source database can rapidly grow. As a result of that, the cost of the repetitive database scan each time the source database gets modified in order to update the set of knowledge is costly with regards to I/O and processing times.

There has been some research works on incremental association rule mining algorithms, i.e. (Zhou and Ezeife, 2001; Cheung et al., 1996). The proposed algorithm in (Cheung et al., 1996) can be considered as a starting point for incremental association rule mining. For example, an incremental association rule mining algorithm called Maintenance Association Rule with Apriori Property (MAAP) (Zhou and Ezeife, 2001), has been presented in 2001. This algorithm efficiently generates incremental rules from an updated database. MAAP computes high level frequent n-itemsets and then starts producing all lower level n-1, n-2, …, 1 frequent itemsets. This approach decreases the processing overhead for generating some of the low-level frequent itemsets that have no chance of being frequent in the updated database. Another incremental association rule mining algorithm, which extends the Fast-Update (FUP) algorithm to handle editing and deleting operations on transactional database, was proposed by (Tsai, et al., 1999). The FUP incremental algorithm deals only with dynamic insertion of records into the database and uses Apriori approach for discovering frequent itemsets. The proposed algorithm by (Tsai, et al., 1999) improves upon FUP with reference to processing time by storing not only frequent itemsets discovered from the original database but also itemsets that are not frequent, but may become frequent after updating the original database. These potential

frequent itemsets may reduce the search time for candidate itemsets in the updated database.

The main challenges for most of the current incremental association rule mining algorithms can be summarized in the following folds:

1) Most of them don't deal with all data manipulation operations (insert, update, delete), like FUP and MAAP, and NFUP considers only either insert and delete. Therefore, there is a need for a generic incremental association rule for all data manipulation operations.
2) Most of the existing incremental algorithms require scanning the source database multiple times in order to find frequent itemsets and produce the knowledge. We aim here to reduce the number of source database scan to just once.
3) We believe that the incremental association rule mining is a challenging problem in data mining, which has not carefully studied. Further, the key to success in solving this problem is to determine the frequent rule items that overlap between the source database set and the records, which have been updated regardless whether the operation is insert, delete or edit. Finally, hereunder are the main aims of this research project.
4) Reducing the number of database scans for the source database to one, considering all data manipulating operations in the source database in the design and the implementation of our algorithm. Extending new existing traditional association rule mining to handle the incremental problem.

2.10. Thesis Contributions

There are several achievements in this thesis. The main contributions are shown and addressed below:

2.10.1. Extending Apriori Algorithm to Handle Incremental Learning

An Apriori algorithm is considered an important algorithm in association rule discovery. It has high efficiency for discovering strong rules. Apriori builds rules, by full scanning the database multi times. The proposed algorithm **In**cremental **Ap**riori (INAP)

extended Apriori to discover rules incrementally with high efficiency without going to rescan the database.

2.10.2. Reducing Database Scans

The main problem of ARM after generating candidate set items, is the need to scan the database to calculate the actual support generated itemset many time. This consumes a great time. For instance, this time will increase largely in many cases. The most important case is when the database is distributed or large. However, in INAP the data will be represent in vertical format (Transactions Identifier List, TID List). INAP needs to scan (db+) new database with the modified old database and transfers it to TID list. After this step the calculation of support for itemset is performed by intersection there lists, without the necessity to rescan database.

2.10.3. Handling All Data Manipulating Operations

INAP algorithm is different from most incremental association rule mining algorithms. Since it takes into consideration all data manipulation operations (add, update, and delete). Most incremental algorithms deals with only the add operations.

2.10.4. Extraction of Knowledge

Naturally, each algorithm of ARM algorithms (algorithms generally) have specific inputs and outputs. Inputs such as: thresholds (Minsupp, Minconf), database. Outputs as: frequent itemset and rules. The output of the ARM algorithms is knowledge based on the inputs thresholds. When we need to extract new knowledge based on the different thresholds, algorithm may need full scan database, and build knowledge from scratch. In INAP the user can extract much knowledge with different threshold without need to rescan original database. The algorithms separate frequent itemset from intermediate complete list based on minimum support that entered each times.

The rest of this thesis is arranged as follows. Chapter 2 reviews the related work. This chapter is divided into two major sections that correspond to the background materials. Chapter 3 presents our approach proposed to discovering association rules incrementally. Chapter 4 concludes and future works the thesis.

Chapter Two

Overview Association rule Mining (Related Work)

3.1. Introduction

Since Agrawal has discussed the problem of association rule mining, it has received a lot of attention from researchers. This attention is motivated by several application domain such as market basket analysis, www analysis, telecommunications analysis, mail order (Li, 2005), telemarketing (Xue, & Zhu, 2009), banking (Kobsa, et al., 2006), and e-commerce (Zhang, 2011). Several association rule mining algorithms have been constructed to solve this problem (Agrawal and Srikant, 1994; Han et al, 2000). They compute association rules that describe a data set. Nevertheless, the sources database is continuously updated. It follows that discovered knowledge describe a data set at the mining moment. Changes of data imply invalidation of knowledge and necessity to update it.

Wherefore, many researchers try to solve the problem, and several algorithms of incremental maintenance have been proposed (Cheung et al, 1996), (Tsai et al, 1999), (Zhang et al, 1997), (Bachtobji, & Gouider, 2006). Most of these algorithms are based on Apriori (Agrawal and Srikant, 1994) that is oriented sparse databases such as transactional databases. These approaches have been developed and are mostly focused on minimizing the number of database rescanning.

However, in this chapter, we will review some of related work, it was done or presented by many of researchers and we will make comparison in the last chapter. After presenting proposed approach in this thesis, I hope that present enhancements in incremental association rule discovery make proceeding in the line of discovery knowledge incrementally. This chapter provides the relevant background for the discussions in next Chapters (chapter 3).

3.2. Common Association Rules Approach

In this section, we will review a common association rule approaches. There are a lot of approaches proposed, but we will focus in most common algorithm Apriori, because we would to extend Apriori to deal with incremental problem.

3.2.1. Apriori Algorithm

Apriori is an influential and classic algorithm for mining frequent itemsets for Boolean association rule learning. The algorithm uses a breath-first search strategy. The algorithm is developed to work on static data, where database is transactional. It finds the frequent itemsets, iteratively by repeating the following steps through multiple scans of the database. At iteration k (step k), it finds frequent itemsets with cardinality from 1 to k. The set of all frequent k-itemsets is denoted by L_k. Then the candidate k+1 frequent itemsets, denoted by C_{k+1}, are generated by combining all combinations of itemsets in L_k. Finally, in the prune phase, any k-itemset that is not frequent and cannot be included in L_{k+1} is removed from C_{k+1}.

First, the set of frequent 1-itemsets is found. This set is denoted L_1. L_1 is used to find L_2, the frequent 2-itemsets, which is used to find L_3, etc., until no more frequent k-itemsets can be found. The finding of each item in L_k requires one full scan of the database. By definition, if an itemset (I) does not satisfy the minimum support threshold, s, then (I) is not frequent, i.e., *I.support* < *s*. The Algorithm Pseudocode for Apriori can you see in Figure.2.4.

Itemset property. All non-empty subsets of a frequent itemset must also be frequent.

By definition, if an item A is added to the itemset I, then the resulting itemset (i.e., I∪A) cannot occur more frequently than I. Therefore, I ∪ A is not frequent either, i.e., (I∪A).support < s. what meaning? All supersets that contain non frequent subset item, the supersets also non frequent itemset.

How to generate candidate set, taking above property? To generate the L_k from L_{k-1}, there are a two step process is followed, consisting of join and prune actions.

1. **The join step**: To find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself $L \bowtie L$. This set of candidates is denoted Ck.
2. **The prune step**: C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k. A scan of the database to determine the support of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k-1)-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.

Whenever, discovery of large frequent itemset is finished, go to generate rule. As we define the association rule previously $A \rightarrow B$, also $A \subseteq I$, $B \subseteq I$, and $A \cap B = \emptyset$, and this association rule $A \rightarrow B$ is strong rule, if have confidence more than minimum confidence (cheung et. at. , 1996; Agrawal and Srikant, 1994).

Where Apriori is one of best algorithms to discovery knowledge from large databases, there are many algorithms was proposed based on Apriori, for handle it to solved specifics problem such as incremental problem. FUP (Cheung, and et at, 1996), NFUP (Chang, and et. At, 2005), IMSC (Bachtobji, & Gouider, 2006), MAAP (Zhou and Ezeife, 2001), Horizontal Format Data Mining With Extend Bitmaps (Alwis, et al, 2012).

- ## Example

Consider the transaction database*DB* presented in (Table2.1) with a minimum support requirement is s=*50%*. As we see in (Table 2.1), the first column “TranID” represents a unique identifier of each transaction, and the“Items” column lists the set of items of each transaction. The original database includes 12 transactions. Accordingly, the supports of the frequent itemsets are at least 6.

Table 2.1: a transaction database

TranID	Items
001	ABC
002	ABD
003	ABCD
004	ACDE
005	ABCD
006	BCD
007	CDE
008	ACDE
009	ABDE
010	BCD
011	BCD
012	ACDF

According to the learning philosophy of Apriori algorithm, and before starting the first pass to find frequent items, we prepare candidate set (*C1*) size of 1 items, that contain all items in the ***DB***. The first step is to make a pass over DB, to find first frequent itemset (1-itemset), that contain one items and his actual support. If an item support is greater than ***Min_Support***, as we show in **Figure**2.1. The item E, F it will be discarded.

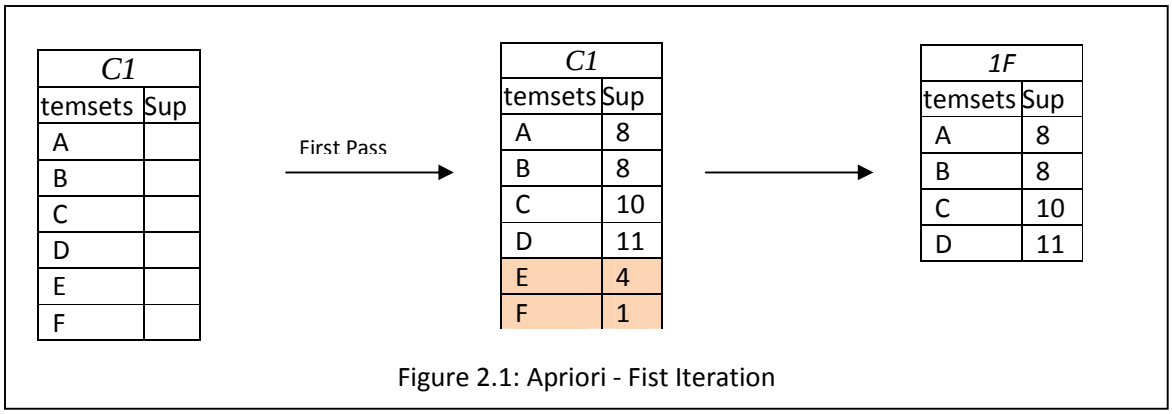


Figure 2.1: Apriori - Fist Iteration

When we reach to frequent 1-itemset (*F1)* , we start the second iteration by passing *F1* over DB to generate candidate 2- itemset (C2), and there actual supports and as show in **Figure**2.2, And discard AB because it had 5 as actual support, and this less than *Min Support.*

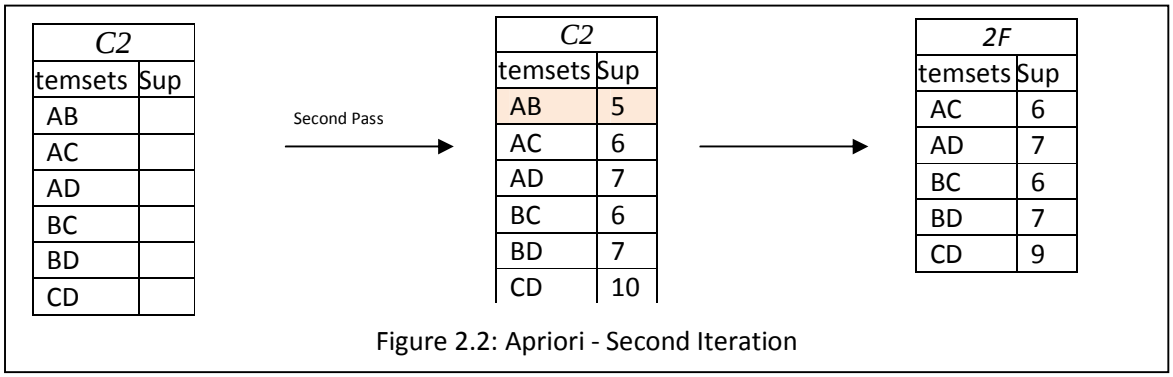


Figure 2.2: Apriori - Second Iteration

Moreover, now we have the 2F item set ,we can go to next iteration to get C3, but in this step we will go deeply into instruction to show important point may by don't take it right previously. However, I well write all possible item set may be exist into output in this iteration from generate candidate item set (C3):{{ ABC},{ABD},{ACD},{BCD}}, but if you take sight to output candidate set C3, we don't see ABC, ABD, in this function generate candidate, the item set is put in output list, if and only if all sub set of item set is

frequent item in input frequent item set, from previous sake the two item set ABC,ABD, discard from output candidate set.

However, now we have C3 candidate set, and can pass to DB, to find actual represent item set in C3 as we show after scan *DB*, {ACD}.support = 5, {BCD}.support= 5, and both is less than Min support, yonder this, two item list can't pass to frequent item set, after complete the frequent item set is empty, for this should be stop.

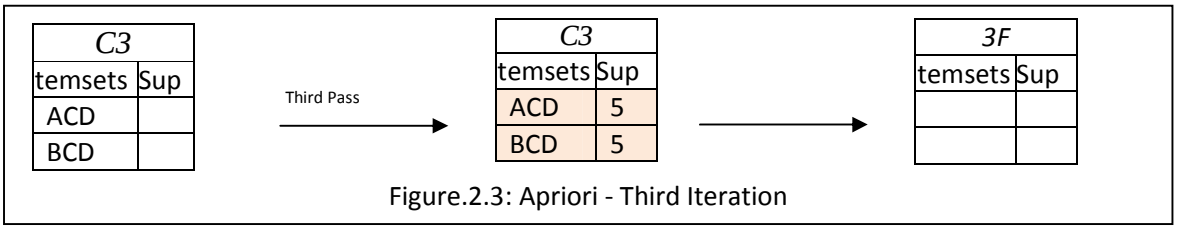


Figure.2.3: Apriori - Third Iteration

Finally we get large frequent item set, and it's contain equal to 2F, if there are item set in 3F, will be add to large frequent item set. And when stop generate function you can go to generate strong rule.

Assume, the *Minimum confidence c= 80%*, find possible association rules?

Since large frequent item set = {(AC,6),(AD,7),(BC,6),(BD,7),(CD,9)}, from this list we can go to generate possible rule as confidence formula we will now Calculate the foreseeable rule:

A→ C =sup(A→ C)/ sup(A) = 6 / 8 = 75% , un exceeds the min confidence .

C→ A =sup(C→ A)/ sup(C) = 6 / 10 = 60% , un exceeds the min confidence.

A→ D =sup(A→ D)/ sup(A) = 7 / 8 = 88% , exceeds the min confidence .

D→ A =sup(D→ A)/ sup(D) = 7 / 11 = 64% , un exceeds the min confidence.

B→ C =sup(B→ C)/ sup(B) = 6 / 8 = 75% , un exceeds the min confidence.

C→ B =sup(C→ B)/ sup(C) = 6 / 10 = 60% , un exceeds the min confidence.

B→ D =sup(B→ D)/ sup(B) = 7 / 8= 88% , exceeds the min confidence.

D→ B =sup(D→ B)/ sup(D) = 7 / 11 = 64% , un exceeds the min confidence.

C→ D =sup(C→ D)/ sup(C) = 9/ 10 = 90% , exceeds the min confidence.

D→ C =sup(D→ C)/ sup(D) = 9 / 11 = 82% , exceeds the min confidence.

From previous calculation, there are four association rule, (Rule, confidence) = {(A→ D, 88%),(B→ D, 88%),(C→ D, 90%),(D→ C, 82%)}.

```
Input: Database, D, of transactions; minimum support threshold, min sup.
Output: L, frequent itemsets in D.
Method:
1) L1 = find frequent 1-itemsets(D);
2) for (k = 2;Lk-1 = □;k++) {
3)    Ck = apriori_gen(Lk-1, min_sup);
4)    for each transaction t ∈ D { // scan D for counts
5)      Ct = subset(Ck , t); // get the subsets of t that are candidates
6)     for each candidate c∈ Ct
7)       c.count++;
8)                    }
9) Lk = {c ∈ Ck | c:count ≥ min sup}
10) }
11) return L = ∪kLk;

Procedure apriori gen(Lk-1:frequent (k-1)-itemsets; min_sup: minimum support)
1) for each itemset l1 ∈ Lk-1
2)   for each itemset l2 ∈ Lk-1
3)     if (l1[1] = l2[1]) ^ (l1[2] = l2[2]) ^ ::: ^ (l1[k-2] = l2[k-2]) ^ (l1[k-1] < l2[k-1]) then {
4)        c = l1⋈ l2; // join step: generate candidates
5)        if has_infrequent_subset(c,Lk-1) then
6)             delete c; // prune step: remove unfruitful candidate
7)        else add c to Ck;
8) }
9) return Ck;

procedure has_infrequent_subset(c: candidate k-itemset; Lk-1: frequent (k-1)-itemsets); // use prior knowledge
1) for each (k-1)-subset s of c
2)     if s ∉ Lk-1 then
3)        return TRUE;
4) return FALSE;
```

Figure.2.4: Pseudocode for Apriori Algorithm (Agrawal and Srikant, 1994)

3.2.2. Eclat (Tid-list Intersection)

The Eclat algorithm has been presented in (Zaki, et al., 1997), to reduce the number of passes over the database. Addressing the question of whether all frequent itemsets can be derived in a single pass for each item in 1-itemset. Eclat uses a vertical database transaction layout, where frequent itemsets are obtained by applying simple tid-lists intersections, without the need for complex data structures.

From **Figure**.2.6 in next page, shows how a typical vertical mining process would proceed from one k-itemset to k+1-itemset using intersections of tidsets of frequent items, without need to rescan database. For example, assume minimum support =33% , the tidsets of T (t(T) = 1356) and of W (t(W) = 12345) can be intersected to get the tidset for TW (t(TW) = 135) which is frequent.

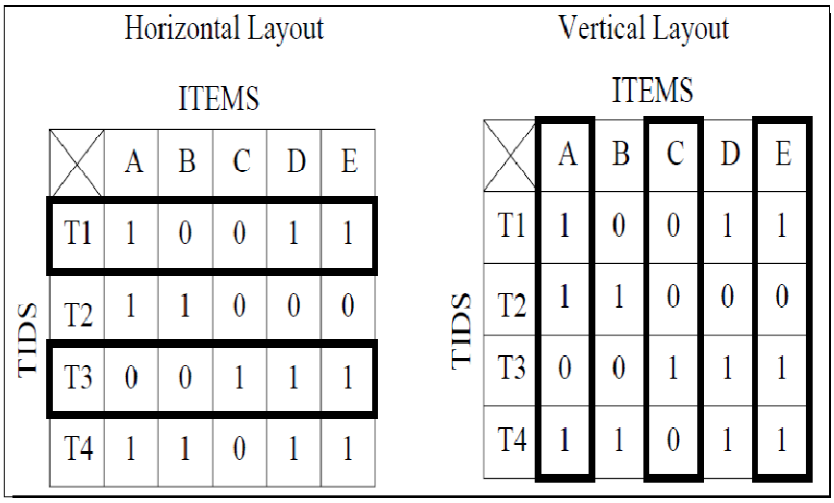


Figure.2.5: Horizontal and Vertical Database Layout

A recent variation of the Eclat algorithm, called dEclat, has been proposed in (Zaki and Gouda, 2003). The dEclat algorithm uses a new vertical layout representation approach called a diffset, which only stores the differences in the transactions identifiers (tids) of a candidate itemset from its generating frequent itemsets. Author claimed diffset is better than tidlist. Yes, it's good while the item set is small and the transaction it contains almost item, but if the set of item is large it's become same.

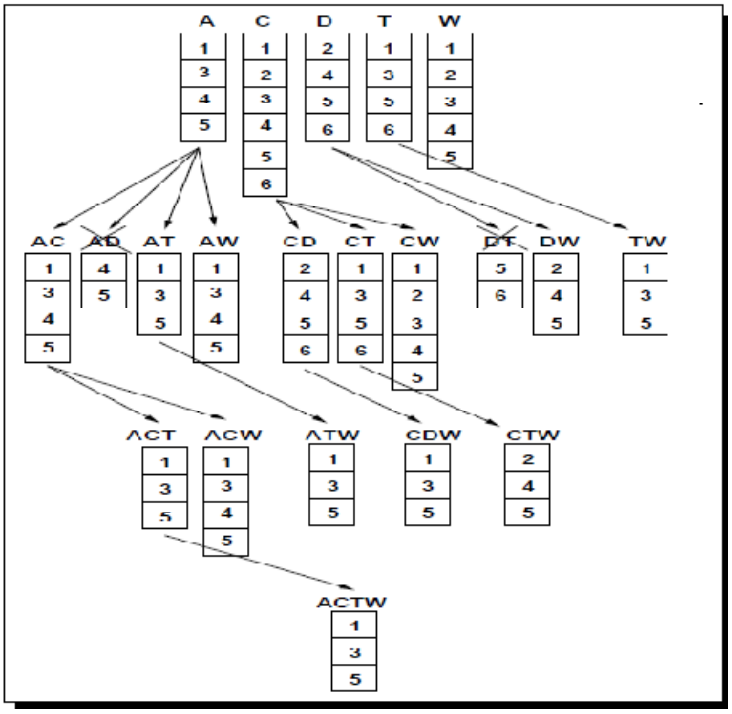


Figure.2.6: Tidsets for Pattern Counting (Zaki Moh., and Gouda K., 2003)

3.2.3. FP-Growth Pattern

FP-growth algorithm (Han, et al., 2000), generates a highly condensed frequent pattern tree (FP-tree) representation of the transactional database. The algorithm does not subscribe to generate and test paradigm of Apriori. Instead, it represents the data set using a compact data structure called an FP-tree, and extract frequent itemsets directly from this structure. Each path in FP-tree is a frequent item. This approach, shrink the scanning source database, where the algorithm read transaction by transaction and represent it in tree, and after reading all transaction, there is no need to rescan database.

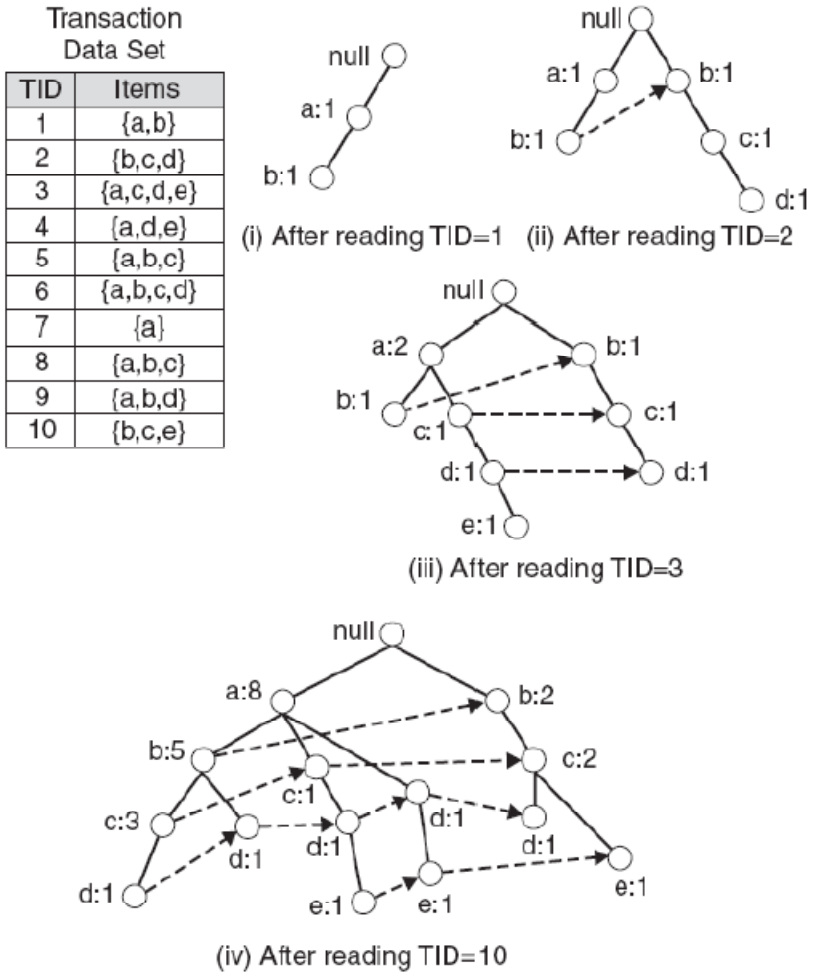


Figure.2.7: FP-tree Pattern to discover frequent itemset

Performance comparison between FP-growth and Apriori on two 10000 record data sets (Han, et al., 2000) indicates that FP-growth is at least an order of magnitude faster than

Apriori since the candidate sets that Apriori must maintain become extremely large. Also the searching process through the database transactions to update candidate itemsets support counts at any level becomes very expensive for Apriori, especially when the support threshold is set to a small value. As the number of transactions grows, the processing time difference between the two techniques becomes still larger.

3.3. Common Incremental Association Rule Approaches

In this section, we will review a common incremental association rule approaches. As we show the most proposed algorithms was extend for Apriori:

3.3.1. Fast Update Algorithm (FUP)

This algorithm deeming the first attempt, to deal with the main problem in this theses incremental association rule discovery, this algorithm was presented by Cheung, et at, and the algorithm was published in paper badge (Cheung, and et at, 1996), the authors have proposed an algorithm called Fast Update algorithm (FUP) to efficiently generate associations in the updated database, the efficiency of this algorithm comes from its first attempt to deal with incremental association mining.

Authors in The FUP algorithm relies on Apriori and reused it after presented on (Agrawal and Srikant, 1994), and considers only these newly added transactions. And the FUP it's a good attempt to deal with increment Database, Let *DB* is original Database, and the Transaction added on called *db*, an X item set from k-itemset, is either frequent or infrequent in *DB* or *db*.

Let db be a set of new transactions and DB be the updated database (including all transactions of DB and db). An itemset x is either frequent or infrequent in DB or db.

Therefore, x may has one case from four possibilities, as shown below in **Figure**2.8.

Case1: x is frequent in *DB*& in frequent in *db*.
Case2: x is frequent in *DB*& infrequent in *db*.
Case3: x is infrequent in *DB*& frequent in *db*.
Case4: x is infrequent in *DB*& infrequent in *db*.

Figure 2.8: Four Scenarios Associated with an Itemset in DB

As we the FUP in an enhancement over Apriori algorithm, and as apriori in the first pass, FUP scans db to obtain the occurrence count of each 1-itemset. Since the occurrence counts of Fk in DB are known in advance.

As it's shown in **Figure** 2.8, if x is frequent in db, there are two possible cases the first one ***Case1***, in this state no problem and this is the best chance, but the other one ***Case3***, in this case unfortunately we need to scan the original DB to find support for this itemset. Similarly, the next pass scans db to count the candidate 2-itemsets of db. If necessary, DB is rescanned. The process is reiterated until all frequent itemsets have been found. In the worst case, FUP does not reduce the number of the original database must be scanned.

3.3.2. New Fast Update Algorithm (NFUP)

After the FUP, Many of researchers (Chang, and et. At, 2005) proposed the algorithm. This algorithm proposed to deal with the main problem in this thesis incremental association rule discovery. The algorithm was published in paper named **New Fast Update Algorithm NFUP**. This algorithm forced in generate itemset from database incrementally with period time, as most of algorithms that work in itemset mining stand on apriori algorithm, and its threshold. The same function is used to generate candidate set. This attempt, as we say to deal with increment association rule from growth Database. However, this approach needs to keep all data with frequent item set in last update of database periodically for each increment in database, and there are final item set.

To picture this approach assume there Transaction Database called *DB*, and generate knowledge for this *DB*, and after a period of time, the decision maker new to last version of knowledge, the algorithm is generating knowledge for the new database, the transaction inserted to database after generate last knowledge from original database, and packet the new database in *db1*, and the next period put database into: *db2,db3,db4,db5,……*

This algorithm new in some cases rescan database but there is a difference between this algorithm and previous algorithm. In the FUP in Case three the algorithm needs to

make complete scan for database to find support to some of frequent item set, but NFUP may need to rescan some of packet of database.

In this algorithm the author suggest three categories for item set in frequent item set:

1) α set : frequent itemset in DB+.
2) β set : frequent itemset in $db^{m,n}$($m \leq n$), but infrequent in $db^{m-1,n}$.
3) γ set: frequent itemset in $db^{m,m}$, but infrequent in $db^{m+1,n}$.

This imply that α needs less rescan the original database, because any item set labeled as α is frequent over all database. Also, the problem still exists, that needs to rescan the original, or some parts in database, and needs to check the β set and γ set, to determining itemset labeled β, or γ is frequent or not.

3.3.3. IMSC Algorithm

While Association rule Mining is hot search field, and increment association is important problem, many researcher go on to solve this problem, (Bachtobji, & Gouider, 2006), propose a new approach called IMSC (Incremental Maintenance of association rules under Support threshold Change) that is based on FUP (Fast UPdate) algorithm (Cheung et al, 1996). FUP maintains a rule base incrementally under the same support threshold. When s=s'(where S: original support, and S': new support), IMSC and FUP are practically identical.

In IMSC modify the FUP algorithm to solve practically problem that change the Support in next time to generate knowledge, when S<>S'. In this approach, there is a good trial to solve important part of incremental association mining that change of support, but there are many parts of problem that searchers a ovoid them. The manipulation operation not solved the researchers solved only insert operation but delete and update not take it and its effect in original data, and in some cases we need to rescan original database.

3.3.4. Maintenance Association Rule with Apriori Property (MAAP)

An incremental association rule mining algorithm called Maintenance Association Rule with Apriori Property (MAAP) (Zhou and Ezeife, 2001). This algorithm efficiently generates incremental rules from an updated database. MAAP computes high level frequent n-itemsets and then starts producing all lower level n-1, n-2,…, 1 frequent itemsets. This approach decreases the processing overhead for generating some of the low-level frequent itemsets that have no chance of being frequent in the updated database.

In the real world where large amounts of data grow steadily, some old association rules can become stale, and new databases may give rise to some implicitly valid patterns or rules.

Hence, updating rules or patterns is also important. A simple method for solving the updating problem is to reapply the mining algorithm to the entire database, but this approach is time-consuming. The algorithm in this paper reuses information from old frequent itemsets to improve its performance. Several other approaches to incremental mining have been proposed.

Although many mining techniques for discovering frequent itemsets and associations have been presented, the process of updating frequent itemsets remains trouble for incremental databases. The mining of incremental databases is more complicated than the mining of static transaction databases, and may lead to some severe problems, such as the combination of frequent itemsets occurrence counts in the original database with the new transaction database, or the rescanning of the original database to check whether the itemsets remain frequent while new transactions are added.

This work proposes an algorithm for incremental mining, which can discover the latest rules and doesn't need to rescan the original database.

3.4. Chapter Summary

This chapter is split into three main sections. First section, we present small overview for normal or traditional approach association rule mining, since the core of this approach association rule mining. Second section, will review some of related work which was done or presented by many of researchers also a comparison will made in the last chapter, after presenting my approach. The two previous sections show the main on challenges in association rule mining. I hope that the enhancements that are made in incremental association rule discovery make proceeding in the line of discovery knowledge incrementally. This chapter also provides the relevant background for the discussions in next Chapters.

Chapter Three

The Proposed Incremental Algorithm

3.1. Introduction

In the previous chapters, we surveyed existing approaches that deal with mining frequent itemset incrementally and statically and their defect. We showed that most of incremental algorithms, have been ignoring some manipulation operations especially data update. It has been argued that it is important to take all manipulating operations by the algorithm into account, especially for real time application such as banking.

In this chapter, we aim to provide an algorithm that improves the efficiency of the incremental learning of ARM techniques by dealing directly with insert, update and delete data operations. There are many problems that will be solved in this chapter, not limited to the reduction of time, and database scans to complete the calculation supports itemsets. One of these problems, to be considered in developing ARM by changing the thresholds values of (Minsupp, Minconf) on the fly. These results, in a new ARM algorithm, called Incremental Apriori Algorithm (INAP).

To speed up the process of finding frequent itemset, the proposed algorithm uses a vertical data format from association rule discovery (Margahny, and hosam, 2010), where each item is followed by a TID-list that represents its occurrences in data. In addition, INAP employs an efficient intersection method that reduces source database rescan, instead of using the Apriori multi scan approach.

This chapter deals with the following:

a) Incremental learning in ARM.
b) Efficiency of rule learning.
c) Using multi input thresholds.

INAP is presented in Section 3.2 where details about itemset discovery are discussed. An overview of the proposed algorithm and abstract methodology were given.

3.2. **Solution Scheme for Proposed Algorithm**

The process of Association rule mining can be divided into two sub-tasks (Agrawal and Srikant, 1994):

(1) The discovery of all frequent itemsets (The itemset have support is equal or greater than the minimum support threshold). This task is the basic foundation in the process.

(2) Generate rule:

X⇨Y ---------- Rule Confidence (*RConf*)

Where X, Y, X∪Y are frequent itemsets, where has *Rconf* above the minimum confidence threshold. The support of an itemset in association rule mining is defined as the proportion of transactions in the database that contain that itemset and the confidence of a rule X⇨Y, is defined as support(X∪Y)/support(Y).

Despite the simplicity and style of the solution strategy used in association rule mining, the first sub-task of discovering frequent itemsets is a computationally expensive problem and has been studied extensively in the literature, e.g. (Li , et al., 2005;Han, et al., 2000; Zaki, et al., 1997).

However, there are many attempts to discover the Knowledge and frequent itemset incrementally to reduce of expensive of process (Anour, et al., 2010; Xu, and Wang, 2006; Chin-Chen, et al.,2005). This research presented good approaches and solutions, as shown in next sections of this chapter. As we know there are manipulate data base periodically, and there are three operations any user in data base application system can manipulate data by using those operations. Add, Update, Delete, and there are two operations, almost all of approaches where disregarded, it Update, Delete, and we mustn't forget the importance of these operations, may be deuce importantly or greater than Insert operation.

The research, take incremental discovery knowledge, as incremental in database, that it's good thing, but the original database as we know it periodically manipulate, whether by insertion, or editing, and may be deletion from Data.

Assume, the database refer to marketing center:

- Insertion: sell new billing, contain many of items.
- Editing or update: may by the customer need to add new items to last buy bills, or backing some of item, and in real world, it's done or need to modification of existing bills more that insertion. e.g. in bank system, the updated database greater than more new account inserted.
- Deletion: may be customer discard or undo buying some of bills.

To picture the previous, and become clearly if there quantum of transaction assume $1x10^6$ transaction in data base named *Distributor_DB*, that refers to a large company that distributes dietary items. The manager of the company had been generate model of knowledge (association rule), as we show in **Figure 3.1**.

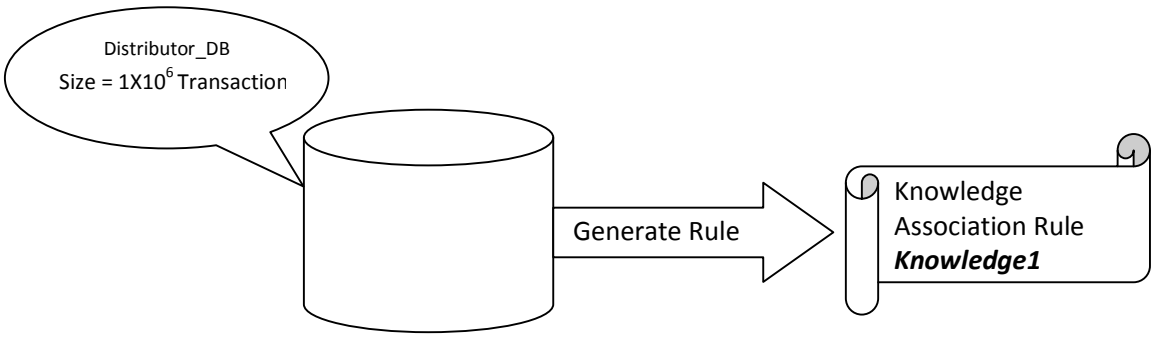


Figure 3.1: Knowledge Discovered from Databases

After amount of time or months, the manager would to generate knowledge model for current state of database, as we show in the Figure 3.2. The current state of database contain four part of data: the first one the new transaction was inserted to database named *New_Distributor_DB* that contains new data added to original data, on the other side there are too many changes happened in original, it is right that nearly 60% of data there are no changes effect thereon, as we can see in Figure 3.2. A bout 25% there are changes and modification in transaction, may be new items added to transaction in this part of data. 15% from original data was removed from database for a reason or another.

Most of approaches assume that the old database always keeps the same status, and there are no changes will occur in data that stored in database. Really, the original data may change periodically, either editing some of transaction or deleting amount of data. Thus, any approach allegation that the original data will always stay on the same status, this is far from truth.

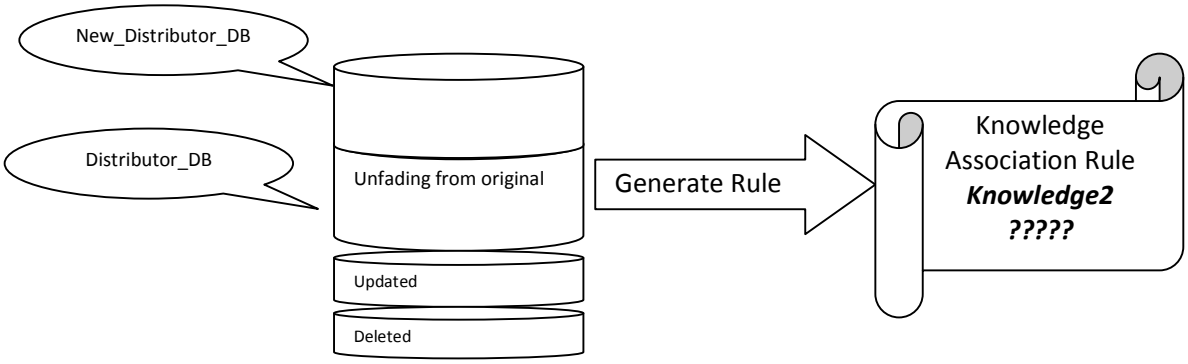


Figure 3.2: Knowledge Discovered from New Manipulate Databases

We build present approach, on this step:

1. From log database file to classify the Transactions in *New_Distributor_DB* in three groups, Added Transactions, updated transactions, deleted transactions.
2. Using tid-list structure, generate intermediate List (*L*): the item set define as

   Where *L*= {i1, i2, i3, …, in},

   in have three attribute in= {i, *S*, *List*<*TID*>}. i meaning itemset, and *S*: Actual support for itemset (i), and List<TID>, contain the transactions id that the itemset (i) exist in it record transaction.

3. When we need to update Knowledge, from step one start from Delete, update, Classes and for all itemset in *L*, which contain transaction ID, remove the transactions ID from List<TID>, and discount the *S*. For add, update, classes applied search method for itemset and update the intermediate list *L*.
4. After complete search for itemset support for added and modify transactions, we separate the itemsets, which have support greater than minimum support.

3.3. Incremental Apriori Algorithm (INAP)

3.3.1. Related Definitions to INAP

In this section we write many definitions that need in this chapter:

- Original database (*DB):* or source database is scanned first time to generate the knowledge, in the form of If-Then rules.
- Modified database *(db+):* after insertion, deletion or updating.
- Log file (*LogF*): controlled by the DBMS, log file saves all changes it is formed that occured on the database. We need to handle all events that occur in (transactions), so the db can be emended all time. A log file has been used to keep track of these database changes. The log file contains three columns which are the *transactionID, ActionID* and finally *ActionDate*. Whenever, a transaction is performed by the user, a new record is added to the log file. This helps identifying the newly changes or updates in the incremental databases by avoiding repeatedly scanning database to locate newly updates.
- Knowledge Date (*KD)*. It is xml file to store the date of the last time the algorithm has executed and generated the knowledge, to ensure that on the next time the algorithm should be executed from the last time it has been found.
- Add transactions list (*LTA*): the list of transactions ID. That has been added to the database after the *KD* date.
- Update transactions list (*LTU*): the list of transactions ID. That got modified in the database after the *KD* date.
- Delete transactions list (*LTD*): the list of transactions ID. That has been deleted from database after the *KD* date.
- First list (*L1*), in this list we save the itemset (that length =1) and its support, and the data structure of this list and the most of the list, Where: $L1 = \{i_1, i_2, i_3, \ldots, i_n\}$,

The three attributes *{i, S, List<TID>}*, where *i* corresponded to an itemset, and *S*: Actual support for itemset (*i*), and *List<TID>:* contain the transactions iD that the itemset (*i*) exists in.

- List of all itemset (*Ln*), in this list we keep the complete itemsets.
- Candidate set (*CL*): intermediate list between *Ln* and *Ln+1*.
- Frequent itemset (*Lf*): the subset of (Ln) with survived support. Meaning all frequent itemsets above *MinSupp* threshold after iteration.
- *Rule*: the association rule generated.
- *MinSupp:* The user minimum support threshold, to derive frequent itemsets. As show in Figure 3.3 below.
- *MinConf:* the minimum confidence threshold the user can define this threshold value, to create strong rule (*rule*). With different threshold values different strong rule list as show in Figure 3.3.

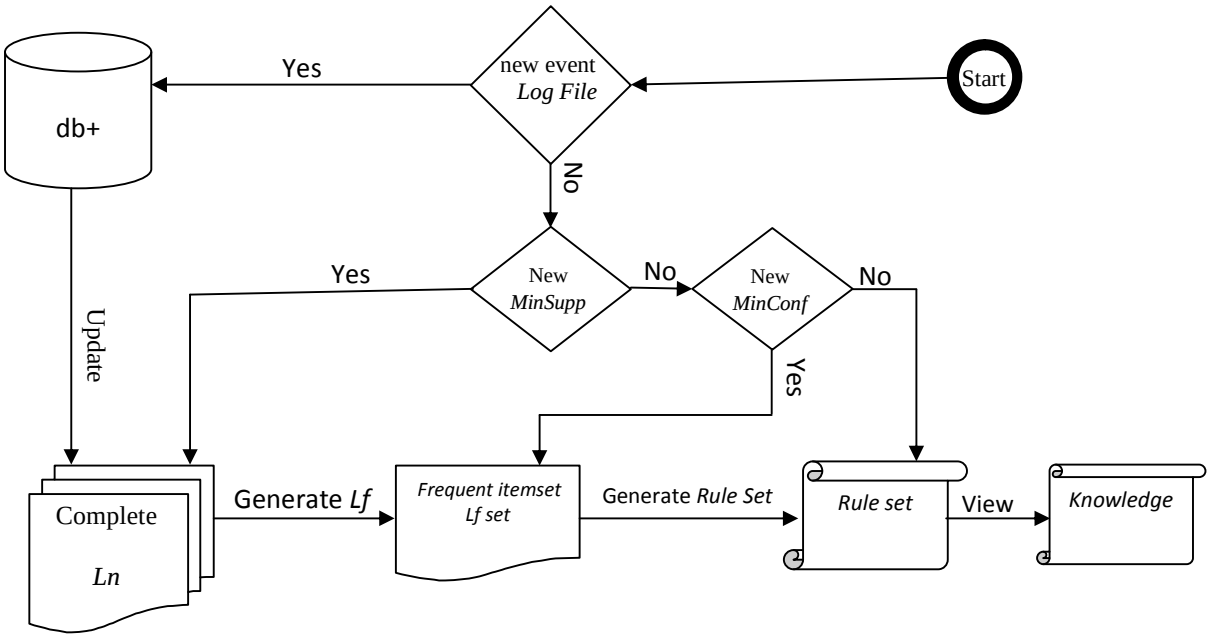


Figure 3.3: The flowchart of the proposed Algorithm

3.3.2. The Algorithm

The learning algorithm proposed in this chapter consists of two main functions to generate association rules. The first one aims to generate complete itemset list with the support for each item (*Ln*). The second one is used to separate the frequent itemset (*Lf*) from large itemset (*Ln*), according to minimum support entered by user (see Figure.3.3), after separation go to generate strong rule, with confidence greater than minimum confidence entered by user. There are many of secondary operation such as: extracting of frequent itemset, generating strong rules, saving output, reading old output.

When execute INAP algorithm need some of inputs to get out knowledge incrementally: the modified database (db+) (Where DB not needed, because we don't need rescan data scanned previously), the last output itemsets, the date, when last output generated. If there is no last output exists in input stream, then the algorithm builds new knowledge from zero.

```
1. Ln ← read(Ln);
2. KD ← read(KD);
3. PreperTransactionList(out LTA, out LTU, out LTD, in LogF, in KD);
4. DiscardTransaction(in LTU, in LTD);
5. L1 ← GetL1Items();
6. do {
7.       CL ← GenerateCandidates(Ln);
8.       Ln ← GetItemset(CL)
9.    } while (CL!Empty)
10. Lf ← GetFrequentItemset(MinSupp,Ln);
11. rule ← GetStrongRules(MinConf);
```

Figure3.4: INAP learning algorithm Pseudocode – Main body

3.3.3. Preparing the Data

3.3.3.1. Data Format

Many presented works in ARM have used the horizontal format presented in the Apriori approach (Agrawal, and Srikant, 1994; Han, et al., 2000)(Alwis, el at,2012). We can see the horizontal format in (Table 2.1, Figure2.5, and Figure2.7). There are many researchers (Zaki and Gouda, 2003; Zhou and Ezeife, 2001; Zaki, et al., 1997) stated thout

approaches for ARM discovery, in vertical format is faster than horizontal format. A database in horizontal format consists many columns (attribute), and there are rows each row name transaction has values for database attribute (for example in supermarket contain purchases items), where each one has a unique identifier. A major drawback resulting from using horizontal data representation is the multiple data scans when searching for itemsets at each level leading to high computation cost.

A database in vertical format comprises a collection of items, where each item is followed by a list of Transaction identifier (tids), which contain that item, this list is called a tid-list (Zaki and Gouda, 2003) show that the vertical layout is a more efficient way of representing the data because candidate generation and support counting are facilitated, since an items support is calculated by fast intersections between tid-lists. The vertical format reduces the number of I/O operations. In spite of the advantage of vertical data representation, when the cardinality of the tid-list becomes very large, intersection time gets larger as well. This happens particularly for large and correlated transactional databases (Zaki and Gouda, 2003).

We have Combining between two ways, because of the prevalence of horizontal format (transactional database), and the desired efficiency from vertical format. Through the adoption of horizontal format in database, and when scan database (only one) to find support for (L1), represent the data into vertical format. When go to find (L2) don't return to rescan database in order to calculate support for 2-itemset. We can see the horizontal format and how convert to vertical in Figure2.5.

According to the sequence of the steps in Section1.2, for the extraction of knowledge, before applying data mining algorithms, we need to prepare the data. In this section, we will discuss preparing Data issue. This process needs to execute sub tasks such as, preparing the data to a mining form, to update the existing knowledge. Surely, we get the initially in horizontal format in the database similar for the data that appear in (Table 3.1). In order, to take advantage of the characteristics of the vertical format (Table 3.2 – Table 3.4), we transfer the data into vertical format to perform mining later such as (Table 3.4).

Table 3.1: Horizontal database format	
TID	Items
1	a,b,d,e
2	b,c,e
3	a,b,d
4	c,d,e
5	a,b,c
6	b,c,d,e
7	a,c,e
8	b,c,d
9	a,b,c
10	a,b,c,d

Table 3.2: Vertical Binary database format , for data (Table 3.1)					
TID	Items				
	a	b	c	d	e
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	0
4	0	0	1	1	1
5	1	1	1	0	0
6	0	1	1	1	1
7	1	0	1	0	1
8	0	1	1	1	0
9	1	1	1	0	0
10	1	1	1	1	0

Table 3.3: Vertical format for database, data show in Table 3.1					
	Items				
	a	b	c	d	e
TID	1	1	2	1	1
	3	2	4	3	2
	5	3	5	4	4
	7	5	6	6	6
	9	6	7	8	7
	10	8	8	10	
		9	9		
		10	10		

Table 3.4: Vertical format (TID List): the data representation need in INAP to allow intersection between lists	
Item	TID
a	1,3,5,7,9,10
b	1,2,3,5,6,8,9,10
c	2,4,5,6,7,8,9,10
d	1,3,4,6,8,10
e	1,2,4,6,7

A database in vertical format comprises a collection of items, where each item is followed by a list of row identifiers (tids), which contain that item, this list is called a TID-list (Zaki, et al., 1997). The study (Zaki and Gouda, 2003) show that the vertical layout is a more efficient way of representing the data because candidate generation and support counting are facilitated, since an items support is calculated by fast intersections between TID-lists.

```
GetL1Items ()
1.    {
2.      Itemlist ← Database.Read("select ID from Item");
3.      Foreach(item in ItemList)
4.       {
5.         Item.Transaction ← Database.Read("select TID from Transaction
             where item_ID= item.ID");
6.         Item.support ← item.transaction.count;
7.         L1.merge(item);
8.       }
9.    }
```

Figure3.5: INAP learning algorithm – GetL1Items Function

Preparing the data we use the `GetL1Items()` function (Figure3.5), this function collects the first itemset from database. For example in supermarket the first-itemset (1-itemset) {bread, milk, yogurt, chocolate, biscuits, sugar, salt, etc….}, the data structure of this list can be shown in the end of Section 3.3.1 (Definition Section), it contains three parts (itemset name, actual support for this item, list of transactions contain this itemset). For each item_ID we find set of transaction id for each item, and get actual support for first itemset. In short, the function GetL1Items represents the data from the horizontal format (Table 3.1) to the vertical format TID-List (Table 3.4).

3.3.4. Incremental Frequent Itemset Discovery

The pseudocode of the INAP algorithm for frequent itemsets discovery is shown in Figures (Figure3.6-3.9). This includes three functions, the first one is candidate set generation. That finds the complete itemset, and frequent itemset. This is accomplished by applying the `GenerateCandidates` function, by passing *Ln* and making intersection to discover *Ln+1*, show Figure3.6.

There is a high similarity between generate candidate set in proposed algorithm INAP, and normal Apriori for generate Candidate set. The deferent between them pruning itemset that have actual support = 0, but in normal Apriori pruning done by cutting the itemset have actual support less than *MinSupp*.

When generating new itemsets, and calculating its support, the `GenerateCandidates` function is used by utilizing intersection among TID List, and by this way we save resources and more importantly one rescan database is used. The way the normal Apriori generates frequent itemset, that needs to satisfy the support by scan database many time for each itemset level.

```
GenerateCandidates (Ln)
1. {
2.   for( all item in Ln)
3.    {
4.     Candidateitem.itemset = Ln[Location].itemset ∪
       Ln[Location+1].itemset;
5.     Candidateitem.TList = Ln[Location].TList ∩ Ln[Location+1].TList;
6.     Candidateitem.support = Candidateitem.TList.count;
7.     CL.add(Candidateitem);
8.    }
9. }
```

Figure3.6: INAP learning algorithm – GenerateCandidates Method

In Apriori there are pruning itemset have actual support less than *MinSupp*, this best step as attempt to reducing rescanning database, but in another way the discovered frequent itemset (knowledge generally), that have support greater than *MinSupp*. The users can't filter or generate other frequent itemset with different *MinSup* threshold, but in our approach users can generate much knowledge with different threshold without need to rescan original database.

```
      GetItemset(CL)
1.   {
2.      Foreach(item in CL)
3.       If(Candidateitem.support>0)
4.        Ln+1.merge(Candidateitem);
5.}
```

Figure3.7: INAP learning algorithm – GetItemset Method

Function GetItemset Figure3.8 has small process, which gets itemset from candidate set (CL), and put the itemsets we have support greater than zero in the complete itemset *Ln* .

`GetItemset(CL)` method will be called in line 8, consider this method functionality which is very easy, but in the same time it is very important. Also, it is articulated point that promotes the proposed algorithm. Which are summarized function as searching for generate itemset that generate with previous method, in large itemset (*Ln*) and if there exists will be merge two transaction list, and addition two support together. Otherwise add generated new item to (*Ln*).

Finally we make intersections, and we get the complete itemset (*Ln*), and (`CL`) become empty set. It is possible to generate new itemset, then algorithm complete the first and important step on algorithm, that calculate actual support for all generated itemset set. After this we will go to generate or separate the frequent itemset (*Lf*), as definition the all itemset that have actual support that greater than *MinSupp*, entered by user. This process will allow us to generate many frequent itemset with deferent *MinSupp*, and this is not available in a lot of ARM algorithms.

```
      GetFrequentItemset(MinSupp,Ln)
6.   {
7.    foreach( item in Ln)
8.      if(item.support>= MinSupp)
9.        LF.add(item);
10.   }
```

Figure3.8: INAP learning algorithm – GetFrequentItemset Method

To this point of algorithm, we have come to the end of the first and most important stage in the algorithm, to find complete itemset with it support. Now we will go to next stage to separate frequent itemset (*LF*), and generate strong rule. Algorithm get (*LF*) by calling the `GetFrequentItemset(MinSupp,Ln)` Method. The main process of this method is to check the support for every itemset in (*Ln*) list, if the itemset support greater than

MinSupp, add this itemset in (*LF*) to become frequent itemset. Of course, this method can be executed and get different (*LF*) with *MinSupp* for each, without the need to scan original database.

While the above method is the most important part of algorithm, we will present partial example for generate a candidate set.

Example 3.1: In table 3.5 appear first itemset (*L1*), now, GenerateCandidates Method will be applied to find *L2*, first step for all itemset in L1 join to other the output of this step is itemset appear in Table 3.2 (AB,AC,AD,BC,BD,CD).

Table 3.5: Example 3.1 L1 itemset

Itemset	Support	TID
A	4	{1,3,4,5}
B	5	{1,4,5,6,7}
C	4	{2,3,4,5}
D	3	{8,9,10}

The next step in GenerateCandidates method is to make intersection for each new itemset from original itemset. AB come from union A itemset, and B itemset then to find AB.TID, need to intersection between A.TID and B.TID.

A.TID ⋂ B.TID =

{1,3,4,5} ⋂ {1,4,5,6,7} = {1,4,5}, which contain three transaction id, this imply AB.Support =3. And so on for all itemset, C2 (Table 3.6).

Table 3.6: Example 3.1 C2 itemset

Itemset	Support	TID
AB	3	{1,4,5}
AC	3	{3,4,5}
AD	0	{}
BC	2	{4,5}
BD	0	{}
CD	0	{}

By applying `GetItemset` in Figure3.6 to the list appear in Table 3.2,Since AD,BD,CD support equal zero, then avoid from L2, As appear in Table 3.7.

Table 3.7: Example 3.1 L2 itemset

Itemset	Support	TID
AB	3	{1,4,5}
AC	3	{3,4,5}
BC	2	{4,5}

The importance in the algorithm it came to dealing with discovering frequent itemsets incrementally, and here we will review an important function that work helped to accomplish that. The first instruction is reading the n itemset *Ln* (last time discovered), in order to modify them and add new changes in the database. In Figure 3.7 function is considered one of the most important among the algorithm functions and instructions. The `PreperTransactionList`, this method aims to classify the manipulated transaction, either

added, updated or deleted transactions. Of course we classify transaction that manipulated after the last date for discovery large itemset (*Ln*). Functions need two in put parameter `LogF, KD`, by definition in previous section the `KD`, is the date of the last time execution. `LogF`, contain the history of tractions action (add, update, delete). There are three output `LTA, LTU, LTD`, lists of added, updated, deleted transactions respectively.

```
PreperTransactionList(out LTA, out LTU ,out LTD, in LogF, in KD)
1. {
2.  openFile(LogF);
3.  Foreach (row in LogF| row.date> KD)
4.  { if(row.ActionType =  Add)
5.     LTA.add(row.TID);
6.    Else if (row.ActionType =  update)
7.     LTU.add(row.TID);
8.    Else if (row.ActionType =  Del)
9.     LTD.add(row.TID);
10. }
11. }
```

Figure3.9: INAP learning algorithm – PreperTransactionList Function

After preparing the lists *LTA*, *LTU*, *LTD*, of course, these lists as it was full meaning in database have a reflection on the discovered Knowledge (*Ln*). We will be dealt with list in two ways. The first way for LTD, and LTU, need to discount the itemsets support, for each itemset have in TID one of the transaction ID in LTD, and LTU. we will see this reflection when we review `DiscardTransaction(in LTU, in LTD)` function. The second way for compilation of existing TID in LTA, and LTU, which will be considered are the source data to calculate the support for itemset.

Now will be trace internal function instruction. First Command `openFile(LogF),` and this allow the processor reading *LogF*. The second instruction `Foreach (row in LogF| row.date> KD),` this command fetch the all row in *LogF* that added to file after *KD* date. Next one is Condition for rows have Action Type is add, will TID (Transaction ID that row refer it) added to `LTA` List, when rows have Action Type is delete, will TID added to `LTD` List. When rows have Action Type equal to update, then TID will be added to `LTU` or added to `LTD`, `LTA`, because when action type is update need to remove old effect in knowledge, and add new effect for updated Transaction.

The next Method `DiscardTransaction`, is simple method, but has high importance. This has function to discard the itemset support for updated and deleted transaction. When the support of itemset depend to updated or deleted transition, and TID exist in itemset transactions ID list. I believe the method instruction is clear, and do not need to explain.

```
DiscardTransaction(in LTU, in LTD)
10.    {
11.    Foreach(item in LN)
12.      {
13.        Foreach(TID in (LTU ∪ LTD))
14.         {
15.          If(item.transaction.exist(TID)
16.           {
17.            Item.support--;
18.            Item.transaction.remove(TID);
19.        }
20.     }
21.    }
22.    }
```

Figure3.10: INAP learning algorithm – DiscardTransaction Function

3.4. Rule Generation

Similarly approximately to extract (*Lf*) we can extract *Rule*, or many *Rule* with different *MinConf*. The best we have to be honest with ourselves and say the `GetStrongRules` was Similar and almost completely to the normal Apriori function to generate strong rule from frequent itemset.

Now we come to the last Method that aims to generate strong rule from (*LF*) list. The main problem in incremental ARM, how to get frequent item set incrementally, and it state main problem in ARM (Xu, and Wang, 2006; Chang, Li, and Lee, 2005; Han, Pei, and Yin, 2000). We focused on our research project how to get frequent itemset. However, we took generate strong rule, as most completely similar to original Apriori generation rule function. Moreover, we can see the pseudocode function for generation rules in Figure 3.9.

INAP consists rule generation, INAP computes the complete set of rules in the form of *Rule*: A → B, where A, B is a frequent itemset, where B itemset length 1. We have built the algorithm to generate consequence contain only one item. Built as well as give

consequence length one item. Our work because most of other fields data mining or application needs to sequence length one item, for ease of use or make generalization to using algorithm in other fields. The *rule* has support and confidence, greater than given support and confidence thresholds.

In the GetStrongRules function as we show Figure 3.11, for each itemset in frequent itemset separate to two sides (antecedent, and consequence) to generate association rule, and calculate the confidence for rule and if the rule confidence greater then *MinConf*, then add rule to the strong rule list.

```
  GetStrongRules(MinConf)
1.  {
2.   Foreach( item in LF)
3.    {
4.     LHS = item.subList(0, length-1);
5.     RHS = item.sublist(length-1, length);
6.     Rule= LHS +”>” RHS ;
7.     Rule. Confidence = item.support/ RHS.support;
8.        if(Rule.confidence> MinConf)
9.       Rulelist.Add(Rule);
10.   }
11.   }
```

Figure 3.11: INAP learning algorithm – GetStrongRules Function

3.5. Comparison of INAP & other ARM Approaches

While our algorithm is enhancement of Apriori algorithm, in this section we will make comparison between INAP algorithm from side, and Apriori algorithm. On other hand, where the INAP consideration incremental learning, we will make comparison between our algorithm and other algorithms in incremental learning field.

➢ Apriori

The INAP algorithm is similar to Apriori algorithm and this normally, because INAP is improve for apriori. However, similar does not mean it is completely identical, although the results are completely similar, we can see the similarity on the result in Experimental Results section, of course the similarity in the output knowledge. We can identify the difference in two main points:

1- The INAP, scan database once time to find L1. After this to calculate the support for new itemset generated make intersection between itemsets compiled. In the Apriori for every itemset generated need to rescan original database (DB, db+).
2- The Apriori pruning in candidate set generate itemsets, which have support less than *MinSupp*, and put itemset generated in the frequent itemset. In the INAP pruning condition for itemsets have support zero, and put the generated itemsets in complete itemsets (*LN*), from (*LN*), the user can generate Multi frequent itemset (*LF*), according to the entered value. Generate rule similar to frequent itemsets, in Apriori the output one rule set that have *MinConf* predetermined, in the INAP the user can generate many rule set see Figure3.1. Of course and we see the comparison experiment result in experiment section.

➢ Incremental algorithms

There are good attempts to propose algorithm to deal with incremental learning for ARM (Chang, and et. at, 2005; Bachtobji, & Gouider, 2006; Zhou, and Ezeife, 2001). They went ahead in the field, but also still some of defect in other approaches such as:

- Don’t take all manipulation operation for database, as we said earlier, that most incremental don’t take the basic operation on database. Proposed algorithm mostly adapted to the incremental knowledge discovery linked to the add operation, on the basic that the old data (in database) is stable and did not get them any change, or deleted. This is contrary to the nature of life, system, and data.

  In proposed algorithm INAP, takes consideration all operations: insert, update, and delete. This means that extracted knowledge is always and ever reflects the reality of the data.

- Some of algorithm gets frequent Itemset without actual support (s) (FUP, NFUP), there algorithm need to rescan source database, to check if infrequent itemsets become frequent or no (MAAP). In algorithm presented INAP addressed these problems and solved it. Naturally, when avoid some

of manipulation operations, then lacking credibility of actual support that once. The second reason, assuming that there is an itemset (i) is infrequent over DB, and when extracts knowledge from db+ i become frequent. Until we achieve that i is frequent over all database (DB, db+), we must rescan all database.

In INAP, retain the value of support for itemsets, even if the itemset is infrequent, in complete itemsets, when apply algorithm over determined *MinSupp*, the user can get the frequent itemsets at these moment of time without need to rescan database.

There other reason to have frequent itemsets without the actual support, such as MAAP algorithm, where start discovery frequent itemset from level n, this imply getting the all subset from frequent itemset in level n as frequent itemset in next level (n-1). Without have actual support for frequent itemset in (n-1,n-2,…) level.

- Other defect, changing in thresholds by user. In IMSC approach there attempt to deal with important issue in the incremental learning, that changes in thresholds when generate, but the approach avoid main issue, where need to rescan database in some cases. In INAP algorithm, solve the changes in thresholds in a professional manner and precise.

We will summarize the main differences between incremental algorithms in next table:

Table 3.8: Comparison between INAP and incremental algorithms

Algorithm	Add	Update	Delete	Rescan solved	The support accuracy	Threshold changes
FUP	✓	×	×	×	×	×
NFUP	✓	×	×	×	×	
IMSC	✓	×	×	×	×	✓
MAAP	✓	×	×	×	×	×
INAP	✓	✓	✓	✓	✓	✓

3.6. Detailed Example

Example 3.2: Consider the database presented in Table 3.9 with a *MinSupp* requirement is 50% and there is no existing old result (*Ln* Is Empty). The database includes 4 transactions. Accordingly, the supports of the frequent itemsets are at least 2. The first column "TID" includes the unique identifier of each transaction, and the "Item" column lists the set of items of each transaction. Table 3.10 contain log file, documenting the operation that occur on database .

Table 3.9: An Example of a transaction database (Example3.2)

TID	Item
1	ab
2	bc
3	ac
4	abc

Table 3.10: Log File 1 (Example3.2)

TID	ActionID	Date
1	1	2012/05/18 12:30
2	1	2012/05/18 12:35
3	1	2012/05/18 12:40
4	1	2012/05/18 12:42

According to algorithm INAP (Figure3.2), firstly need to read last result, where the last result does not exist then the last result is empty set. Secondly, will prepare classification for TID in Log file based on the [ActionID] (preparing LTA, LTU, and LTD lists) LTD, and LTU is empty list, LTA is {1, 2, 3, 4}. While last result does not exist then *DiscardTransaction* function will not applied. Thirdly, the algorithm applying *GetL1Items* function, to scans the original database (specifically fetch the transaction, that TID exist in (LTA ∪ LTU)) and counts 1-itemsets (*L1*) List with their support and transaction ID List (<TID>), they located to make it easily finding such transactions, see table below (Table 3.11).

Table 3.11: 1-itemsets (L1) (Example3.2)

ListIndex	Itemset	Support	<TID>
1	a	3	1,3,4
2	b	3	1,2,4
3	c	3	2,3,4

After get (*L1*), passing it to GenerateCandidates function, to generate Candidate itemset of size 2, C2 and count their support by intersection the (L1) : ab.TID = a.TID ∩ b.TID = {1,3,4} ∩ {1,2,4} = {1,4} then support of ab is 2.

ac.TID = a.TID ∩ c.TID = {1,3,4} ∩ {2,3,4} = {3,4} then support of ac is 2.

bc.TID = b.TID ∩ c.TID = {1,2,4} ∩ {2,3,4} = {2,4} then support of bc is 2, and we have got the (L2) itemset that appear in (Table3.12).

Table 3.12: 2-itemsets (L2) (Example3.2)			
ListIndex	Itemset	Support	<TID>
1	ab	2	1,4
2	ac	2	3,4
3	bc	2	2,4

The next steps, generate C3, and make intersection between ab, ac or ac, bc TID List, the result intersection {4}. Then the abc.support =1. Since L3 is only one item, then no more candidate itemset can generate .The final complete itemsets Ln is discovered, and can we show in Table3.13. Now can find LF by take all itemset in Ln that have the support greater than or equal *MinSupp(50%= 2 transaction),* then the frequent itemset LF can see in Table 3.14.

Table 3.13: Complete itemset Ln (Example3.2)			
ListIndex	Itemset	Support	<TID>
1	a	3	1,3,4
2	b	3	1,2,4
3	c	3	2,3,4
4	ab	2	1,4
5	ac	2	3,4
6	bc	2	2,4
7	abc	1	4

Table 3.14: Frequent itemset Lf (Example3.2)			
ListIndex	Itemset	Support	<TID>
1	a	3	1,3,4
2	b	3	1,2,4
3	c	3	2,3,4
4	ab	2	1,4
5	ac	2	3,4
6	bc	2	2,4

And finally, the rules are generated as follows (Table3.15):

Table 3.15: Rule set (Example3.2)		
ListIndex	Itemset	confidence
1	a->b	66.6%
2	a->c	66.6%
3	b->a	66.6%
4	b->c	66.6%
5	c->a	66.6%
6	c->b	66.6%

To this, we got to the end of algorithm, and built knowledge of nothing. In order to become an incremental approach clear in the algorithm, we're going to see in the complementary example.

Example 3.3: (complementary example for example3.2) Consider the database presented in Table 3.16. With a *MinSupp* requirement is 50%, the last result exist in complete itemset list (*Ln*) in (Table 3.13). The database includes 4 transactions. Table 3.17 is log file.

Table 3.16: An Example of a transaction database (Example3.3)

TID	Item
. . .	. . .
4	ac
5	ab
6	abc

Table 3.17: Log File 1 (Example3.3)

TID	ActionID	Date
. . .	. . .	. . .
3	3 (delete)	2012/05/19 12:35
4	2 (update)	2012/05/19 12:40
5	1(add)	2012/05/19 12:42
6	1(add)	2012/05/20 12:42

- Last results exist in (Table 3.13).
- Will prepare classification for TID in Log file based on the [ActionID] (preparing LTA, LTU, and LTD lists), LTA is {5, 6}, LTU is {4}, LTD is {3}.
- Now will discount for all itemsets in *Ln* (Table 3.13) that contain any TID from (LTA ∪ LTU) in itemset <TID>. (LTA ∪ LTU) = {3, 4}, *Ln* after discount support and remove TID exist in (LTA ∪ LTU) from <TID>.

Table 3.18: Complete itemset Ln after Discount support and remover TID in (LTA ∪ LTU) (Example3.3)

ListIndex	Itemset	Support	<TID>
1	a	1	1
2	b	2	1,2
3	c	1	2
4	ab	1	1
5	ac	0	
6	bc	1	2
7	abc	0	

- Get L1 itemset, fetch the transaction, that TID exist in (LTA ∪ LTU) and counts 1-itemsets (*L1*) List with their support and transaction ID List (<TID>). (LTA ∪ LTU) = {4, 5, 6}.

Table 3.19: 1-itemsets (L1) (Example3.3)			
ListIndex	Itemset	Support	<TID>
1	a	3	4,5,6
2	b	2	5,6
3	c	2	4,6

- Merge 1-itemset (Table 3.19), with Ln (Table 3.18). merge result in Table 3.20

Table 3.20: Complete itemset Ln after merge with 1-itemsets (Example3.3)			
ListIndex	Itemset	Support	<TID>
1	a	4	1,4,5,6
2	b	4	1,2,5,6
3	c	3	2,4,6
4	ab	1	1
5	ac	0	
6	bc	1	2
7	abc	0	

- After get (*L1*), passing it to Generate Candidates C2, and count their support by intersection the (L1) :

  ab.TID = a.TID ∩ b.TID = {4, 5, 6} ∩ {5, 6} = {5, 6} then support of ab is 2.

  ac.TID = a.TID ∩ c.TID = {4, 5, 6} ∩ {4, 6} = {4, 6} then support of ac is 2.

  bc.TID = b.TID ∩ c.TID = {5, 6} ∩ {4, 6} = {6} then support of bc is 1.

Table 3.21: 2-itemsets (L2) (Example3.3)			
ListIndex	Itemset	Support	<TID>
1	ab	2	5,6
2	ac	2	4,6
3	bc	1	6

- Merge 2-itemset (Table 3.21), with Ln (Table 3.20). merge result in Table 3.22

Table 3.22: Complete itemset Ln after merge with 1-itemsets (Example3.3)			
ListIndex	Itemset	Support	<TID>
1	a	4	1,4,5,6
2	b	4	1,2,5,6
3	c	3	2,4,6
4	ab	3	1,5,6
5	ac	2	4,6
6	bc	2	2,6
7	abc	0	

- The next steps, generate C3, and make intersection between ab, ac or ac, bc TID List, the result intersection {6}. Then the abc.support =1, merge L3 (Table 3.23) with Ln (Table3.23). Since L3 is only one item, then no more candidate itemset can generate.

Table 3.23: 3-itemsets (L3) (Example3.3)			
ListIndex	Itemset	Support	<TID>
1	abc	1	6

- The final complete itemsets Ln is discovered, and can we show in Table3.24. Now can find LF by take all itemset in Ln that have the support greater than or equal *MinSupp(50%= 2.5(3) transaction),* then the frequent itemset LF can see in Table 3.25.

Table 3.24: Final complete itemset Ln (Example3.3)			
ListIndex	Itemset	Support	<TID>
1	a	4	1,4,5,6
2	b	4	1,2,5,6
3	c	3	2,4,6
4	ab	3	1,5,6
5	ac	2	4,6
6	bc	2	2,6
7	abc	1	6

Table 3.25: Frequent itemset Lf (Example3.3)			
ListIndex	Itemset	Support	<TID>
1	a	4	1,4,5,6
2	b	4	1,2,5,6
3	c	3	2,4,6
4	ab	3	1,5,6

And finally, the rules are generated as follows:

Table 3.26: Rule (Example3.3)		
ListIndex	Itemset	confidence
1	a->b	75%
3	b->a	75%

3.7. INAP Distinguished Features

The INAP algorithm Presented has characteristic and features that distinguish it from other algorithm ARM, specifically incremental algorithms, her we will state many feature:

- The first and feature important one, INAP takes into minding three manipulation operations for database in association rule mining: add, update, and delete. Most the algorithm does not take these operations, therefore when it comes to mining they require complete scan over the update database (old database DB and new data db+) to update the knowledge.
- Horizontal data format is a good thing in order to accelerate database operations. For that, INAP algorithm allow user to use horizontal format in the database, alternatively, and in the first scan to find L1, represent the output data into vertical layout (Tid List). A recursive philosophy based on intersection to discover itemset, which requires only one scan.
- The Data Mining approach and algorithms generally, and the ARM algorithm especially, to find knowledge must using thresholds: minimum support and minimum confidence. In fact, this is the bottleneck to work in the discovering Knowledge. So, if extracted knowledge on certain threshold (minimum support: *S%*; minimum confidence: *C%*). When you change the value of (*S%* or *C%*), you need to rescan database and to applying all algorithm process and task, to deal with the new changes. In our algorithm you can extract many knowledge with different threshold, without need to rescan database.

3.8. Experimental Results

In this section, we describe experimental study and results, to measure the relative performance of our proposed algorithm. The algorithm was implemented and tested on Intel compatible with processor i5 and 4 GB of main memory running the Microsoft Windows 7 operating system.

As said previously, the algorithm can read database formatted with horizontal layout. After fetching data, representing it to vertical layout. We present the database schema for two main tables and instance for each data. The first table refers to items defined in the system, and contains Item_ID, Item_Name for each item as Shown in Table 3.27. The second table for register sales items and it contains basically two attribute T_ID, Item_ID Table 3.28.

Table 3.27: Data scheme for items

Item_ID	Item_Name
1	Biscuit
2	Bread
3	Juice
4	Milk
5	yoghurt
6	Coffee

Table 3.28: Data scheme for Sales items

T_ID	Item_ID
1	1
1	2
1	3
1	4
1	5
1	6
2	1
2	2
2	4
3	4
3	5
3	6

The implemented algorithm was run against data with (100- 100000) transactions. We obtained the data from (Apriori Dataset). Also, the implemented algorithm allows to user insert, update, delete transactions, and the program generates automatically random transactions of any size. As we see in the Figure 3.12, user can enter number of transaction that would insertion in (Transaction Numbers) box appearing in the below figure. After inputting, and if the inserted number is large in size, the algorithm utilizes a damp function that inputs the new transactions to the database. The number of transactions the user inputs can be added, and the program then inserts the new transactions.

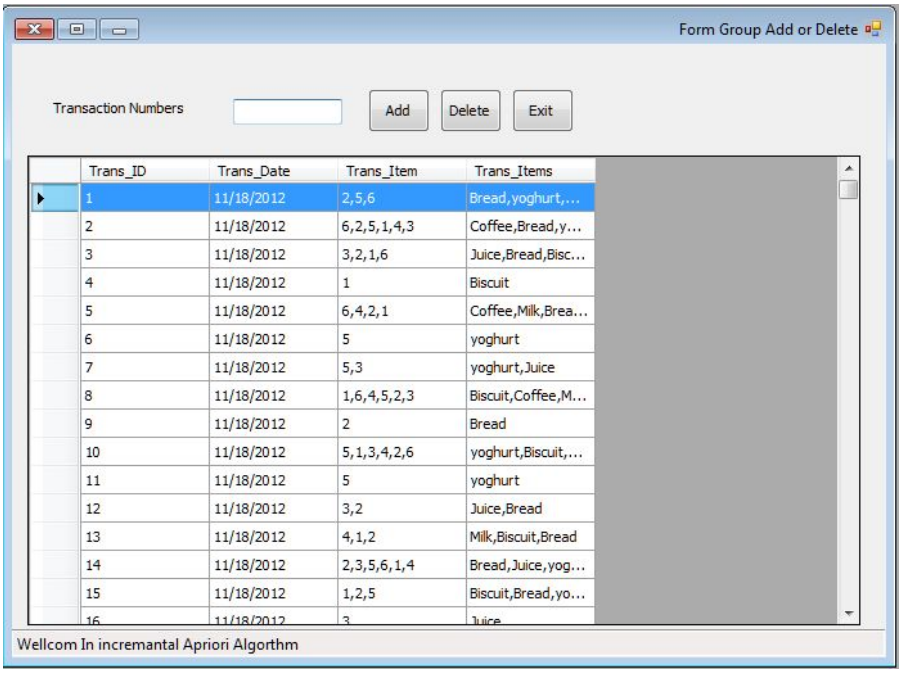


Figure 3.12: Implemented Algorithm- insert, delete group of transactions

(Figure 3.13 shows the “Transaction Form”, which the user can insert transactions one by one, delete transactions, and update existing transactions. Larger data size can be inputted or deleted using the damp function mentioned earlier.

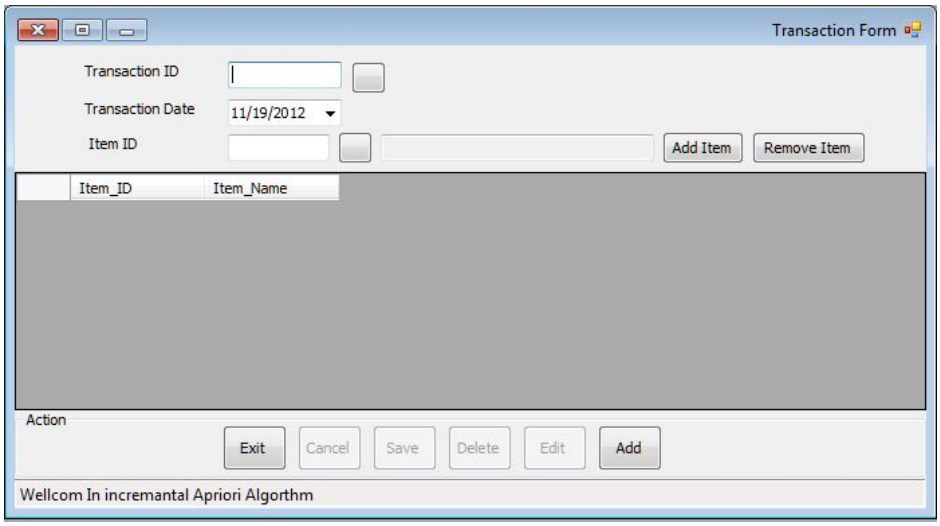


Figure 3.13: Implemented Algorithm- Transaction From

Here, in the (Figure 3.15), we show an important form in the program “incremental Apriori (INAP) form”. This form displays the classification transactions (add, update, and delete). Users can enter thresholds (*MinSupp, MinConf*) in this screen, and the result tab outputs frequent itemsets, and association rules.

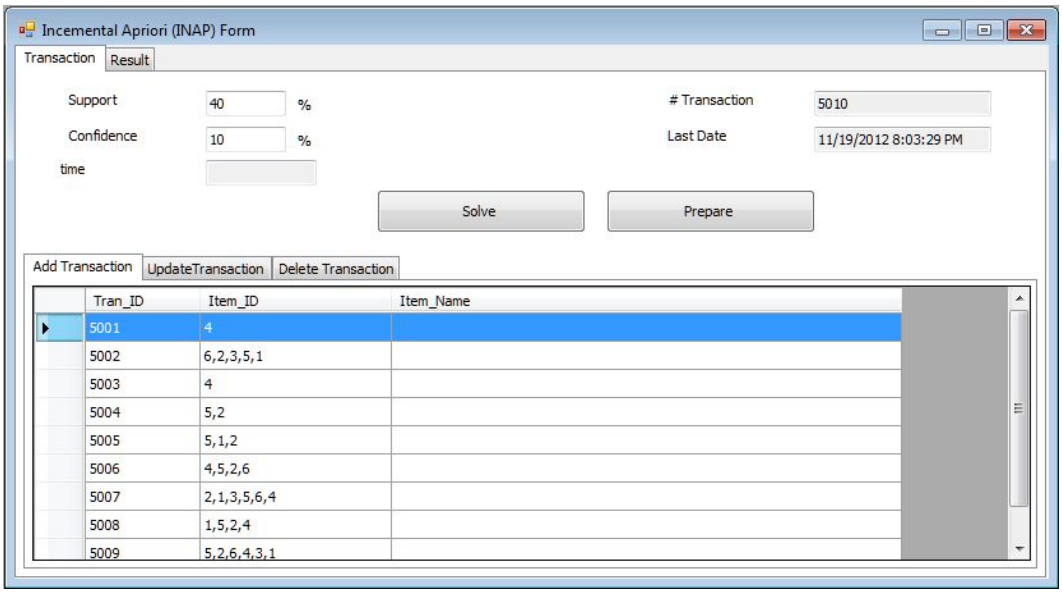


Figure 3.15: Implemented algorithm- Incremental Apriori (INAP) Form

Experiments have been contacted on different data sets based for supermarket databases from (Apriori Dataset). Moreover, we have experimented through our program, which described earlier. We compared our model with Apriori, because our algorithm is an incremental enhanced of Apriori. The comparisons have many criteria the: frequent itemset number, the support value, also rule set size, the rule set confidence. The basic criterion for comparison is the execution time.

Here, in this experiment, we compare the Apriori and INAP using some criteria, as shown in (Table 3.28, Figure 3.15). We apply two algorithms with deferent dataset sizes with the same thresholds (*MinSupp*=1%, and *MinConf* =10%). The size of frequent itemsets and the numbers of rules is similar for INAP and Apriori algorithms. Since both algorithms depend on Minsupp & Minconf parameters, in additions to the similarity in the number of result set, there was similarity in the value of support for frequent itemsets, and rule confidence for fair comparison. Table 3.29, and Figure 3.15, show the execution time, where time has been reduced an average of less than one-third. When you show the count of the frequent itemset, and rule number, will appear the same but realty the value of support and confidence change every time.

Table 3.29: Experimental results, for dataset with different each time increase 1000 transaction

# of Transaction	Time (millisecond)		Frequent itemset		Rule	
	INAP	Apriori	INAP	Apriori	INAP	Apriori
1000	270	950	63	63	186	186
2000	280	580	63	63	186	186
3000	200	400	63	63	186	186
4000	100	350	63	63	186	186
5000	150	560	63	63	186	186
6000	250	1000	63	63	186	186
7000	260	600	63	63	186	186
8000	240	900	63	63	186	186
9000	245	700	63	63	186	186
10000	240	600	63	63	186	186

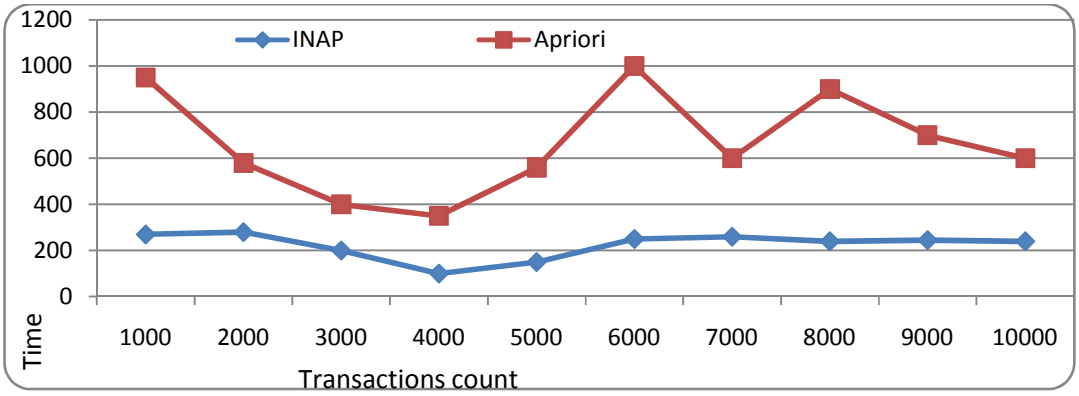


Figure3.15: Running time under different total number of transaction size in milliseconds

Here, in this experiment we compared the Apriori and INAP using some criteria, as shown in (Table 3.30, Figure 3.16). We apply two algorithms with different thresholds values, with the same dataset size (5000 Transaction). We show the output knowledge, the size of frequent itemset, and number of rules.

Table 3.30, Figure 3.16, display the reduction in the execution time, when the reproduction of knowledge base on the new threshold. Time has been reduced an average of less than 0.1 millisecond, which is an achievement. Meaning, one can extract knowledge with different thresholds in time less than one in ten of millisecond (0.1 millisecond).

Table 3.30: Experimental results, for dataset with different *MinSupp* threshold

MinSupp threshold	Time millisecond		Frequent itemset		Rule	
	INAP	Apriori	INAP	Apriori	INAP	Apriori
1%	100	600	63	63	186	186
5%	0.07	680	63	63	186	186
10%	0.07	690	63	63	186	186
15%	0.07	650	63	63	186	186
20%	0.07	530	56	56	150	150
25%	0.07	970	41	41	90	90
30%	0.07	900	21	21	30	30
35%	0.07	950	21	21	30	30
40%	0.07	130	6	6	0	0

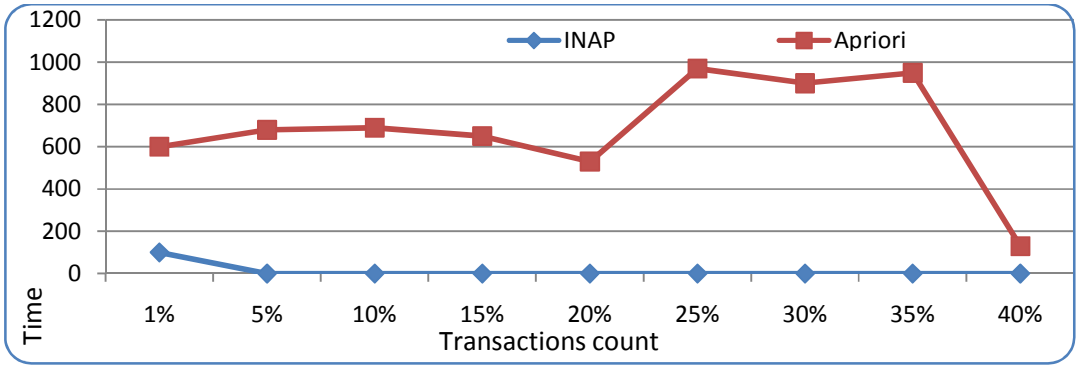


Figure3.16: Running time (milliseconds) under different thresholds for the same transaction size

The performance of the proposed algorithm has been evaluated in regards to the aspect of efficiency define. For the rule accuracy, the INAP generates similar results. In Figures 16-17, both rule set count, or frequent itemsets, and F1 in (Table3.31, Figure 3.17) correspond to the time needed to find frequent items of sizes 1. Denote the time needed to transform data from horizontal format to vertical format. In these experiments, we have used different *MinSupp* values where we have lowered the *MinSupp* in some cases to 3% and used different dataset size varying from 2 - 4 Million transactions. Figure 15 shows the time taken to transform the data to item-space when *MinSupp* equals 3% including the generation of 1-frequent items.

Table 3.31: Experimental results- for Find 1-frequent itemset, for different dataset size

# of Transaction	Time (millisecond)		Frequent itemset	
	INAP	Apriori	INAP	Apriori
1000	160	150	6	6
2000	250	500	6	6
3000	170	300	6	6
4000	85	350	6	6
5000	150	300	6	6
6000	200	500	6	6
7000	250	450	6	6
8000	240	550	6	6
9000	230	400	6	6
10000	235	450	6	6

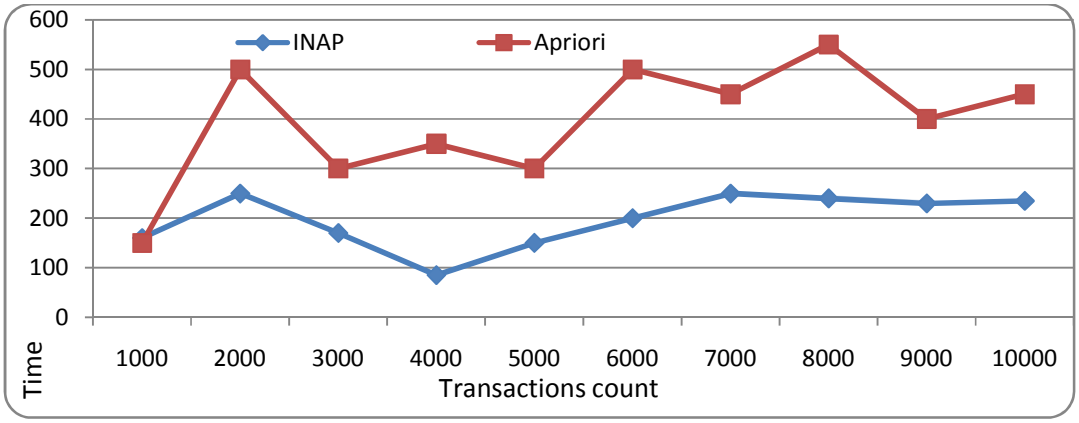


Figure3.17: Running time (milliseconds) to discover 1-itemset for different dataset size

3.9. Chapter Summary

In this chapter, the problem of using incremental ARM approaches in data mining has been investigated. The outcome is a new effective incremental ARM algorithm, INAP, that made enhancement to Apriori algorithm. INAP has got a number of new features over other existing techniques. The proposed algorithm is able to handle the changes that get in the thresholds. This gives the power of the algorithm, because, in fact, the changes in threshold are possible and frequent. In itemset discovery, INAP presents a fast intersection method that requires only one database scan, consuming less processing time than other multi scan database approaches.

In the next chapter, we will provide conclusions, and weaknesses that could be existed in the proposed algorithm. This problem will be solved in the coming researches, by any researcher in this field.

Chapter Four

Conclusions and Future Work

4.3. Conclusions

In this thesis, an Apriori algorithm is developed to perform discover association rule and frequent itemset incrementally. Through Apriori, discovery of the knowledge, either frequent itemset, or itemset, is through a full scan of database and frequent rescan. The Apriori has many features; the most important of these features are: the knowledge produced through Apriori reflect the real current state of the data in the database, and in a professional manner. We all know that database unstable, that there are manipulations operations occurring continuously on database, whether to delete, update existing records, or add other new records. The volume of data in database is growing rapidly with time, according to the system and the environment. Apriori, despite its many features, becomes a problem, which requires knowledge discovery, a full scan of database, and iteration rescan.

There are many algorithms that where proposed to deal with the problem, in serious attempts to get the knowledge incrementally. Some of these algorithms FUP, NFUP, IMSC, MAAP, offered these algorithms, many creative ideas to solve the incremental learning. With knowing that algorithms (FUP, IMSC, MAAP), came essentially based on Apriori, NFUP came as enhancement for FUP. All previous algorithms acquired Apriori properties. However, there many defects still existed in current approaches such as:

a. The current algorithms don't take all manipulation operation for database, where adopted only add operation, for transactions are added to the database, and ignored the update and delete operations.
b. Get frequent Itemset without actual support (s), for two reasons: the first one suppose that X is itemset on DB, most of the algorithms consider that X will remain frequent forever (with ignored effect of update, delete operation), and this not correctly, it possible to delete or update those transactions that have X itemset

between it item. The second reason, where MAAP algorithm gets the highest level of itemset, and test is frequent or no, if it frequent split it to subset frequent itemset without scan database, and unknown actual support for subset.

c. The algorithms rescan all source databases, to check if infrequent itemsets become frequent or no.

The proposed algorithm INAP utilizes the incremental association mining efficiently, to maintain the frequent itemset, and rule set incrementally. As a direct consequence of this usage, INAP performs never rescan original database (Scand database), and scan only added or updated transactions, taking into consider the transactions deleted. By using INAP the user can create many instances for output knowledge, according to input thresholds.

Several experiments have been tried to make sure the performance of the algorithm compared with Apriori. We focused in experiments on the execution time of the algorithm, was a satisfactory result to a good extent.

4.4. Future Work

Much work can be done in future in this area of research. There are some suggestions which have been rejected by other researchers. Many people may wonder why this could happen. In all cases, I show entire appreciation and respect for all researchers:

- **Negative association rule**

  ARM is an important task in data mining. There are two main types for association rule. The type was studied in this thesis is the positive type; this type is typical association rule. Search for itemset, and rule in transactions in the database.

  For example $A \rightarrow Y$, where A, Y, and $A \cup Y$ is frequent itemset, and occur frequently in database, as we review in chapter two, there are a lots of algorithm dealing with this type.

The second type is called negative association rule. This type of rule negates the correlation. A confined negative association rule is one of the following:

$X \longrightarrow \neg Z$,

$\neg X \longrightarrow Z$,

or $\neg X \longrightarrow \neg Z$, where the entire antecedent or consequent must be conjunction of the negated attribute or a conjunction of non-negated attribute (Mani, 2012).

The meanings of the above rule, for example the first rule $X \longrightarrow \neg Z$, the transactions that contain X itemsets, don't contain Z itemsets. The common bottleneck in the approaches for association rule generation is "Support, Confidence", in the negative the calculated support is complicated (Mani, 2012). In this thesis, it easy to find the support of itemsets, either negated or non-negated itemsets, because there are lists contain all transaction ID for all itemset, we can find support of itemsets only by intersection lists.

The research in our field "ARM", can extend the proposed algorithms INAP, to deal with the negative association rule generation.

➢ Algorithm Idea Generalized

For most data mining task, we need discovery itemsets, and need to calculate the support for them. This common problem, how to generate the frequent itemsets with actual support, is the main obstacle for most data mining tasks (Mani, 2012). Incremental problem: an important problem that is faced in data mining process is continuously evolving new data, and modified existing data. It is necessary that existing approaches of classification, clustering, and other data mining task, tackle this in such a way that the classifier is tuned-in to accommodate it (Joshi, and Kulkarni, 2012).

It is important to make a comparison between the present approach in this thesis and approaches proposed in other data mining task. If the present algorithm has got results better than other algorithms, then generalization of our algorithm approach can be made for other tasks.

REFERENCE

Agarwal R. and Srikant R., “Fast algorithms for mining association rules,” in 20th VLDB Conference, Santiago, Chile, September 1994.

Alwis B., Malinga S., Pradeeban K., Weerasiri D., Perera S., “Horizontal Format Data Mining with Extended Bitmaps”, International Journal of Computer Information System and Industrial Management Applications Vol.4, ISSN 2150-7988, PP. 514-521, 2012.

Anour F.A. Dafa-Alla, Ho Sun Shon, Khalid E.K. Saeed, Minghao Piao, Un-il Yun, Kyung Joo Cheoi, and Keun Ho Ryu. “IMTAR: Incremental Mining of General Temporal Association Rules”. Journal of Information Processing Systems, Vol.6, No.2, June 2010. P.P163-176.

Apriori Dataset, https://wiki.csc.calpoly.edu/dataset/wiki/apriori.

Bachtobji Moh., Gouider Moh., “INCREMENTAL MAINTENANCE OF ASSOCIATION RULES UNDER SUPPORT THRESHOLD CHANGE”, IADIS International Conference Applied Computing 2006, ISBN: 972-8924-09-7, IADIS, 2006.

Bao H. T., “Introduction To Knowledge Discovery and Data mining”, institute of information technology, National Center for Natural Science and Technology, 1998.

Brin, S., Motwani, R., Ullman, J., and Tsur, S. Dynamic itemset counting and implication rules for market basket data. Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data,1997,(PP. 265-276). Tucson, Arizona, USA.

Chang C.C., Li Y. C. , and Lee J.,”An Efficient Algorithm for Incremental Mining of Association Rules”, in Proc. of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA’05). IEEE.2005.

Chen M., Han J., Yu P., “Data Mining: An Overview from Database Perspective”, IEEE Transactions on Knowledge and Data Engineering, vol.: 8, PP. 866-883, 1996.

Cheung D., Han J., Ng V., Wong C.,”Maintenance of discovered association rules in large databases: an incremental updating technique”, The 12th International Conference on Data Engineering Proceedings, New Orleans, Louisiana, USA, March 1996, (PP. 106-114),1996.

Cheung, DW; Lee, SD; Kao, B. “A general incremental technique for maintaining discovered association rules”. Proceedings of the 5th International Conference on Database Systems for Advanced Applications, Melbourne, Australia, 1-4 April 1997, p. 185-194. IEEE.1997

Dudek D., Zgrzywa A., “The Incremental Method for Discovery Association Rule”, Computer Recognition Systems, Spring Berlin, vol: 30, 2005, ISBN: 978-3-540-25054-8, (PP. 153-160).

Elmasri R., Navathe S., “Fundamentals of Database Systems”, Third Edition, Maite Suarez-Rivas, (2001).

Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.;”From Data Mining to Knowledge Discovery in Databases".1996.

Ferdous Ahmed Sohel, Chowdhury Mofizur Rahman. “Association Rule Mining in Dynamic Database Using the Concept of Border sets”. Proc. of the ICEECE December 22-24, Dhaka, Bangladesh

Han J. , Kamber M., “Data Mining: Concepts and Techniques”, Simon Fraser University, Morgan Kaufmann Publishers, 2000

Han J., Pei J., and Yin Y.,” Mining frequent patterns without candidate generation”. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, PP 1-12, 2000.

Honglei Z., Zhigang Xu. “A Novel Incremental Updating Algorithm for Maintaining Discovered Negative Association Rules”. 2009 International Conference on Research Challenges in Computer Science. IEEE, PP.164-167, 2009.

Hongsong Li, Houkuan Huang. “New Estimation Methods of Count-Min Sketch”. Proc. of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA’05), IEEE, 2005.

Jonker J, Pennink B. “The Essence of Research Methodology”. Springer: New York.2010.

Joshi P., Kulkarni P., “Incremental Learning: Areas and Methods A Survey”, International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.2, No.5, September 2012

Kobsa, A., Chellappa R., “Spiekermann S., “Privacy-Enhanced Personalization”. CHI 2006 • Workshop, ACM, PP. 1631-1634, 2006.

Kothari, C.R.,”Research Methodology, Methods & Techniques.” New Delhi: New Age International, Pvt. Ltd. “Foreign Investment In India”, (2009), NSE INDIA, PP. 186-211. Handbook of NSE, 2010

Li h., Suh-Yin Lee, Man-Kwan Shan. “Online Mining (Recently) Maximal Frequent Itemsets over Data Streams”. Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA’05). IEEE.2005.

Mani T.," Mining Negative Association Rules", IOSR Journal of Computer Engineering, Vol. 3 Issue 6, PP. 43-47, 2012.

Margahny Moh., Hosam R., "Hori-Vertical Distributed Frequent Itemsets Mining Algorithm on Hetrogeneous Distributed Shared Memory System", IJCSNS International Journal of Computer Science and Network, vol. =10, PP. 56- 62, 2010.

Rajak K., Gupta M. K., "Association Rule Mining: Application in Various Areas", International Conference on Data Management, PP. 3-7, Ghaziabad , 2012.

Sharma K. N., Nagwani N.K., "Study and Analysis of Incremental Apriori Algorithm", High Performance Architecture and Grid Computing – international Conference, HPAGC 2011, Vol.: 169, Chandigarh, India, Proceeding 2011, (PP. 470-473).

Tanbeer S., Ahmed C., Jeong B., Lee Y. "Efficient single-pass frequent pattern mining using a prefix-tree". Elsevier Inc. October 2008.

Tarek F. Gharib, Hamed Nassar, Mohamed Taha, Ajith Abraham. "An efficient algorithm for incremental mining of temporal association rules". Elsevier B.V. March 2010. P.800-815.

Thivakaran.T.K, Rajesh.N, Yamuna.P, Prem Kumar.G. "PROBABLE SEQUENCE DETERMINATION USING INCREMENTAL ASSOCIATION RULE MINING AND TRANSACTION CLUSTERING". 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies. IEEE.2009. p.37-41.

Thomas S., Bodagala S., Alsabti K., Ranka S. "An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases". KDD-97 Proc, PP.263-266, 1997.

Xu C., and Wang J. , "An Efficient Incremental Algorithm for Frequent Itemsets Mining in Distorted Databases with Granular Computing". In Proc. of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06). 2006.

Xue M., Zhu C., "The Application of Data Mining In the Decision of Supermarket Extension and Businesses Expansion Based on Evolutionary Computation". 2009 Pacific-Asia Conference on Circuits, Communications and System. IEEE. 2009. (PP.778-780).

Yinggang Zhao, Qinming He. "An Incremental Learning Algorithm Based on Support Vector Domain Classifier". Proc. 5th IEEE Int. Conf. on Cognitive Informatics (ICCI'06) Y.Y. Yao, Z.Z. Shi, Y. Wang, and W. Kinsner (Eds.). IEEE.2006.

Zaki M. J., Parthasarathy S., Ogihara M., Li W., "New Algorithms for Fast Discovery of Association Rules", NSF Research Initiation Award (CCR-9409120), ARPA Contract

(F19628-94-C-0057), American Association for Artificial Intelligence (www.aaai.org), 1997

Zaki Moh., Gouda K., ”Fast Vertical Mining Using Diffsets”, SIGKDD ’03, 2003 ACM , Washington, USA, 2003

Zhang, “Research of Personalization Services in E-commerce Site based on Web Data Mining”. 2011 International Conference on Computational and Information Sciences. IEEE.2011.(PP. 438-441)

Zhou, Z., Ezeife, C. A low-scan incremental association rule maintenance method based on the Apriori property. Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence, 2001, (PP. 26-35).

الملخص

تعدين القواعد الجمعية يعتبر واحد من أهم الأعمال في تعدين البيانات، والذي جلب الكثير من الإنتباه من اللجان البحثية. خوارزمية ابرويروي تقوم على إيجاد قواعد الجمعية بطريقة إبداعية وذكية وعلى مقياس كبير لقواعد البيانات، وتعتبر واحدة من أهم الخوارزميات لإستكشاف قواعد الجمعية. المشكلة الرئيسية المرتبطة بأبرويروي هي المسح المتعدد لقواعد البيانات لإيجاد المعرفة (القواعد الجمعية) عند تحديث قواعد البيانات كل مرة. وتزداد صعوبة المشكلة عندما تكبر قواعد البيانات مع الوقت. النتائج المكتشفة من البيانات الأصلية تحتاج عند تعدين مجموعة من البيانات المعدلة تحتاج إلى تحقق من صحة المعرفة التي حصلت في وقت سابق.

الباحثون إقترحوا العديد من الخوارزميات للتعامل مع المشكلة التزايدية. خصوصا في التطبيقات التي تتغير في البيانات في قواعد البيانات بشكل مستمر مثل التطبيقات البنكية. قدمت الكثير من الخوازميات حلا للمشكلة على نحو ذكي، مثل الخوارزمياتFUP, IMSC, MAAP. وعندما قمنا بمراجعة نظرات التعلم التزايدي الحالية وجدنا بعض العيوب مثل:

1- لم تأخذ كل عمليات معالجة البيانات، وخصوصا عملية التعديل.

2- هذه الخوارزميات تستدعي إعادة مسح قواعد البيانات عدة مرات.

3- بعض هذه الخوازميات يستكشف المعرفة بدون نسبةِ ترددها في قواعد البيانات.

إنّ الخوارزمية المقترحة في هذه الأطروحة تدعى ابرويروي التنزايدية (**In**cremental **Ap**riori (INAP))، وهي تتعاملُ مع المشاكل المذكورة أعلاه. إنها تعدين لقواعد الجمعية وبشكل تزايدي ولا تحتاج إلى إعادة مسح قواعد البيانات القديمة عندما تجلب التحديثات. تأخذ الخوارزمية كل عمليات معالجة البيانات (الحذف، التعديل، والإضافة) في الحسبان عندما تعدن مجموعة جديدة من البيانات دون العودة لمسح البيانات الأصلية. تسمح خوارزمية (INAP) لنا لإستخراج المعرفة بعتبات مختلفة (نسبة تكرار في قواعد البيانات، ونسبة قوة القاعدة – الثقة) مرات عديدة ودونم الحاجة إلى الرجوع إلى قواعد البيانات مرة أخرى، وهذا يعني إنه تم حل مشكلة التزايد في القواعد الجمعية.

تعدين قواعد الجمعية التزايدي
مستندة على مجموعة العناصر الكاملة المتوسطة

قدمت من قبل

إياد أحمد عزالدين أقرع

بإشراف

د. فادي ثبتة

هذه الأطروحة قُدّمتْ لقسم علم الحاسوب

كمتطلب جزئي للحصول على شهادة الماجستير في علم الحاسوب

عمادة البحث العلمي والدراسات العليا

جامعة فيلادلفيا

يناير/كانون الثّاني 2013