FAULT-TOLERANT

FREQUENT PATTERN MINING

WITH CLASSFIERS

BY

BEI XU

Bachelor of Science

Northeastern State University

Tahlequah, Oklahoma

2003

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2006

FAULT-TOLERANT

FREQUENT PATTERN MINING

WITH CLASSFIERS

Thesis Approved:

_____Dr. George E Hedrick_____
Thesis Advisor

_____Dr. Meg Kletke_____

_____Dr. Debao Chen_____

_____Dr. A. Gordon Emslie_____
Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. G. E. Hedrick for his intelligent supervision, constructive guidance, inspiration and friendship. My sincere appreciation extends to my other committee members Dr. Meg. Kletke and Dr. Debao Chen, whose guidance, assistance, encouragement prodding me to stay on schedule, kept me going.

Moreover, I would like to express my sincere appreciation to those who provided suggestions and assistance for this study: Dr. John P Chandler, Dr. H. K. Dai.

I would like to give my special thanks to my parents Mr. Deben Xu, Mrs. Ruqin Fu, my sister Hui Fu and my uncle Shuihao Fu, for their support and encouragement.

Finally, I would like to thank the Department of Computer Science and Department of Management of Science and Information System for their support during these years of study.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

NOMENCLATURE

CFT-FPM	Fault-tolerant Frequent Pattern Mining with Classifiers
FT-FPM	Fault-tolerant Frequent Pattern Mining
min_sup	Minimum Support Threshold
min_sup^{FT}	Minimum Fault-tolerant Support Threshold
min_sup^{item}	Minimum Item Support Threshold

CHAPTER 1

INTRODUCTION

1.1. BACKGROUND

Databases today tend to be large. Data mining[1] (sometimes also called knowledge discovery) finds hidden patterns and relationships in data both to save storage and to predict future events. These relationships include classification[2], clustering[3], association[4] and sequential analysis[5]

Frequent pattern mining (FPM) is a data mining technique to find frequent patterns (FP) in a dataset. FPM is very important in many data mining tasks, such as mining association rules[6] [AS94], sequential patterns[7] [AS95], multi-dimensional patterns[8] [KHC97], correlations[9] [BMS97], causality[10] [SBMU98], episodes[11] [MTV97], max-patterns[12][B98], partial periodicity[1] [HDY99], and emerging patterns[2] [DL99]. According to Pei [PTH01], while mining frequent patterns, if the support variables[3] are

[1] Data mining – The definition is found in http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm
[2] Classification - Store data into predetermined group based on data's properties.
[3] Clustering - Data are grouped according to logical relationships or customers' preferences.
[4] Association - Define a relationship such as X=>Y where X, Y are different items or itemsets.
[5] Sequential analysis - Data are analyzed to predict trends.
[6] Association rule mining - The process of finding all the association rules (ARs) in a given dataset.
[7] Sequential patterns - Patterns that contain items which appear in a timely order.
[8] Multi-dimensional patterns - Patterns are mined based on the constraints.
[9] Correlations - A statistical technique which can show whether and how strongly pairs of variables are related.
[10] Causality - The principle of or relationship between cause and effect.
[11] Episodes - Patterns that contain series of related items.
[12] Max-patterns - Pattern with maximum length and no supersets of them are patterns.

set too high, few patterns can be found. This problem is solved by Fault-tolerant Frequent Pattern Mining.

The concept of Fault-tolerant Frequent Pattern Mining (FT-FPM) was introduced by Pei [PTH01]. According to Pei, with FT-FPM, one may look for approximate and more general fault-tolerant frequent patterns instead of finding exact frequent patterns in data. Thus, longer, more general patterns with higher support[4] can be discovered, rather than specific, harder to find, patterns [PTH01]. However, in FT-FPM, the same fault tolerance is evenly applied to all data despite of the different properties among them. Consequently, the result is not quite realistic.

This paper proposes to mine fault-tolerant patterns with classifiers (CFT-FPM). With CFT-FPM: 1) one or more classifiers is (are) picked to be fixed at a set of specific values or specific ranges; 2) then, FT-FPM is used to mine patterns based on the corresponding fault-tolerance. Since the result is driven from classified data and proper fault-tolerance, patterns that are closer to reality are discovered.

In the following example[5], this paper illustrates frequent pattern mining (FPM), fault-tolerant frequent pattern mining (FT-FPM) and fault-tolerant frequent pattern mining with classifiers (CFT-FPM).

Example 1: VHS/DVD movie rental

After doing business for years, the manager of a VHS/DVD rental store wants to find frequent patterns in the movie rental database to increase future sales. In order to

---

[1] Partial periodicity - Patterns that contains items which repeat in a timely order.
[2] Emerging patterns - Association of features whose frequency increases significantly from one class to another.
[3] Support variables - Certain parameters that need to be set to define certain frequent patterns.
[4] Support - Number of records in the database contains the pattern.
[5] All examples in this paper are made by the auther.

find frequent patterns in the database, first he must define what a frequent pattern is. In other words, he must set the support threshold (minimum value of support) for a frequent pattern. In this case, he set the support threshold to 50%. Thus, for a movie rental pattern to be considered frequent and valuable, it should have no less than 50% pattern support. Mining results show four old movies: a, b, c, d and one new movie: e form an interesting pattern.

Frequent pattern (FP):

Pattern P0: 85% of customers who rented *four out of four* old movies also rented the new movie.

$$85\%: \quad (a) \,\&\, (b) \,\&\, (c) \,\&\, (d) \rightarrow (e)$$

Since the support of 85% is greater than the support threshold of 50%, the pattern is considered a frequent pattern in the database. What if the support is less than 50%? Let's look at pattern P1:

Pattern P1: 35% of customers who rented *four out of four* old movies also rented the new movie.

$$35\%: \quad (a) \,\&\, (b) \,\&\, (c) \,\&\, (d) \rightarrow (e)$$

Since the pattern support is only 35%, P1 is not considered frequent. If every pattern discovered has less than 50% pattern support, then data mining in the database will be meaningless and no pattern can be used for future prediction. Thus, in order to extract meaningful frequent patterns from the database, the manager either have to decrease the desired support (make it less than 50%), or increase the pattern support by relaxing some constraints. Fault-tolerant frequent pattern mining is such a technique to increase pattern support by relaxing some constraints.

Fault-tolerant frequent pattern (FT-FP):

The movie rental manager also discovered that some customers, who rented only three out of four old movies, also rented the new movie. He decided to add these customers to the customers in pattern P1. Thus, he discovered a new pattern with higher support. Let's look at pattern P2:

Pattern P2: *60%* of customers who rented *at least three out of four* old movies also rented the new movie.

$$60\%: \quad (abc) \mid (bcd) \mid (abd) \mid (acd) \mid (abcd) \rightarrow (e)$$

P2 achieves higher support of 60%. Since pattern support of 60% is higher than the minimum support (50%), P2 is valuable for future prediction. This method produces more general results than P1 because every customer who satisfies P1 also satisfies P2. We call P2 a fault-tolerant frequent pattern. Fault-tolerant frequent pattern mining (FT-FPM) is a mining method to discover all the fault-tolerant frequent patterns (FT-FP) from the database. It is used to increase the pattern support by relaxing some constraints: "three out of four" instead of "four out of four".

However, in reality, FT-FPM is not recommended to be applied evenly in every case. According to Fischoff, etc [FAL97], "Those in the Older age range show the least appreciation for films in the HORROR genre". Thus, young people might be more interested in scary movies than older people. In other words, chance for a young person to watch a new scary movie without a scary movie rental history is greater than that of an older person. CFT-FPM suggests that an age variable in the dataset should be used to

classify the data. All of the data is included, assuming no missing values for the age variable.

Let's look at how CFT-FPM might produce a pattern when classified by age.

Mine Fault tolerant frequent pattern with classifiers (CFT-FPM):

Pattern P3: We discover different rules based on customer age range:

1) *65%* of customers who are under age 20 and watched *at least two out of four* old movies also rented the new movie "e".
2) *75%* of customers whose age is between 20-50 and watched *at least three out of four* old movies also rented the new movie "e".
3) *80%* of customers who are above 50 years old and watched *all four* old movies also rented the new movie "e".

The following summarizes pattern P3:

if (age < 20)

65%: (ab) | (ac) | (ad) | (bc) | (bd) | (cd) | (abc) | (bcd) | (acd) | (abd) | (abcd) $\rightarrow$ (e)

else if (20 < age <50)

75%: (abc) | (bcd) | (acd) | (abd) | (abcd) $\rightarrow$ (e)

else if (age >50)

80%: (a) & (b) & (c) & (d) $\rightarrow$ (e)

In this case, CFT-FPM constructs three specific rules in P3; whereas, for the same data FP-FPM constructs one general rule in P2. In P3, we see that if a young person watched only 2 out of 4 old scary movies, most of them will watch the new scary movie. On the other hand, if an older person watched 4 out of 4 old scary movies, most of them

will watch the new scary movie, which makes more sense according to the reality. Compare P3 to P2, it can be seen that pattern P3 adds an age classifier based on the value of the age variable. P3 discovers a more detailed pattern that is closer to the reality expressed by the data. Based on the above pattern, the VHS/DVD store would recommend movie "e" to the customers who are under age 20 and watched two out of above four old movies, to the customers whose age is between 20-50 who watched three out of four old movies, and to the customers who are over 50 years old and watched all four old movies. Thus, customers' needs are satisfied better, and sales may be increased significantly.

We introduce CFT-FPM[1] to discover patterns such as the above in this paper.

1.2. The Problem and the Purpose

Frequent pattern mining is very important in many data mining tasks and has broad applications. Fault-tolerant frequent pattern mining (FT-FPM) was introduced to "achieve higher pattern support to discover longer patterns and more general rules from the database" [PTH01]. However, FT-FPM treats data as a whole and does not differentiate it by its properties. Thus, unrealistic patterns might be generated since FT-FPM is not based on the data's properties. In order to extract patterns that are closer to reality, new idea should be introduced.

[1] CFT-FPM - A mining technique we proposed that mines fault tolerant frequent patterns based on one or a set of classifier(s).

1.3. Objectives

This paper proposes to mine fault-tolerant frequent pattern mining with classifiers (CFT-FPM). This proposal adapts the FT-FPM approach to mine fault-tolerant frequent patterns based on one or a set of specific classifier(s) value. In this way, it generates rules that are tailored to the data itself. We also implement and test the CFT-FPM over a real-world dataset. The objectives of this paper are:

1. To analyze CFT-FPM.
2. To implement CFT-FPM.
3. To test CFT-FPM over a real-world dataset.

1.4. Outline of the Thesis:

The remainder of the thesis is structured as follows:

Chapter 2: The frequent pattern mining and an overview of related work

Chapter 3: The concept and implementation of CFT-FPM

Chapter 4: The results and findings of CFT-FPM

Chapter 5: The conclusions and suggest future work

CHAPTER 2

REVIEW OF THE LITERATURE

In this chapter, we review the history of:

1. Frequent Pattern (FP)
2. Frequent Pattern Mining (FPM)
3. Association Rule Mining (ARM)
4. Apriori[1] Heuristic
5. Apriori Algorithm
6. Fault-tolerant Frequent Pattern Mining (FT-FPM)
7. Classification

2.1. Frequent Pattern (FP):

Example 2: Customer Transactions.

Table "Customer Transactions (version 1)"[2] contains some sales data from a supermarket sales database.

Customer ID	Purchased Items
1	1, 2, 3, 4, 5, 6, 9
2	1, 2, 4, 7, 8
3	2, 3
4	1, 3, 5, 11
5	1, 3, 4, 5, 12, 24

Table 1: Customer Transactions (version 1)

[1] Apriori – This is the correct spelling of the name of the algorithm, The dictionary spelling would be a priori.

[2] Customer Transactions (all versions) – derived from Transaction database in Pei[P02

There are five transactions (records) in the table. Each transaction stores information about a customer along with the items purchased. The table contains two column attributes and they are the "Customer ID" (primary key) and the itemsets "Purchased Items".

According to Agrawal, given a user specified support threshold min_sup, a minimum length threshold min_length, itemset X is considered a frequent pattern in the table iff sup(X) ≥ min_sup and length(X) ≥ min_length [AIS93]. Given min_sup = 3, min_length = 3, X = {1, 3, 5}. Scan through the "Purchased Items" column which contains itemsets, we see that X is contained in transactions 1, 4, 5. With sup(X) = 3 and length(X) = 3, itemset X is considered a frequent pattern in this table since sup(X) ≥ min_sup(3) and length(X) ≥ min_length(3). However, if we set the minimum support threshold to 4; E.g.,., min_sup = 4, then X is not a frequent pattern in this table since sup(X) < min_sup(4). If we set min_length to 4, X is not a frequent pattern since length(X) < min_length(4).

This means that 60% of the customers bought items {1}, {3}, {5} together. In other words, itemset {1, 3, 5} is a popular set. One might consider placing these three items together to increase sale.

2.2. Frequent Pattern Mining (FPM)

The **concept of frequent pattern mining** was first introduced by Agrawal [AIS93] to mine frequent patterns between sets of items. The definition of frequent pattern mining follows.

According to Agrawal, $I = \{i_{1,\ldots,}i_n\}$ is a set of **items**. An **itemset** X is a non-empty subset of I. E.g.,., $X \subseteq I$. $X = i_{j1}, \ldots i_{jk}$. An itemset containing K items is called a

k-itemset. A transaction is denoted as T = (tid, X), where tid is a transaction-id and X is an itemset. T is said to contain itemset Y iff $Y \subseteq X$. A **transaction database** TDB contains a set of transactions (records). The number of transactions in transaction database TDB containing itemset X is called the **support** of X, denoted as **sup(X)**. Given a transaction database TDB and a **support threshold** min_sup > 0, an itemset X is a **frequent pattern** iff $sup(X) \geq min\_sup$. The **problem of frequent pattern mining** is to find all the frequent patterns in database based on a given min_sup [AIS93]. Another mining technique, association rule mining, can be derived from frequent pattern mining.

2.3. Association rule Mining (ARM)

According to Agrawal, association rule mining (ARM) is the process of finding all the association rules (ARs) in a given data set. An association rule (AR) defines a relationship such as X =>Y, read as" if X is true, Y is also likely to be true" (X and Y are **itemsets** and $X \cap Y = \varnothing$). The rule X => Y has **support s** in a transaction database TDB if: **sup ($X \cup Y$) = s** (where support s is a scalar, $\cup$ is union operation). The rule X =>Y holds in the transaction database TDB with **confidence c**, where: **c = [sup ($X \cup Y$)]/sup (X)** (where confident c is a scalar). Given a transaction database TDB, a **support threshold** min_sup and **confidence threshold** min_conf, the **problem of association rule mining** is to find all the association rules that satisfy the min_sup and min_conf [AS94].

For instance, given min_sup = 3, min_conf = 70%, let X = 1, Y= (3, 5), itemset {1, 3, 5} is a frequent pattern in table 1 since {1, 3, 5} is contained in transaction 1, 4, 5, which satisfies the min_sup(3). This implies the three customers who purchased the item "1" also purchased the items "3, 5". Thus, s = sup ($1 \cup (3, 5)$) = 3. Since item 1 is

contained in database four times, which means, sup (1) = 4. Thus, c = [sup (1 ∪ (3, 5))]/sup (1) = 3/4 =75%. Since s ≥ min_sup(3), c ≥ min_conf (70%), the relationship between itemset{1}, {3, 5} is an association rule discovered from the database. From this rule, one can conclude that {1}, {3, 5} associate with one another in majority (3/5=60%) of the transactions. Follow the same logic, we can also discover the association rule between itemset {3} and {1, 5}, and the association rule between {5}, {1, 3}. Thus, there is a relationship (association rule) among three items {1}, {3} and {5}. Supermarket might consider placing all three items together to increase the chance for customers to buy all three items at the same time. Thus, sales might significantly increase since more items are sold at the same time.

Mining frequent patterns could be very time consuming, many methods were proposed to mine frequent patterns efficiently.

2.4. The Apriori[1] Heuristic

The Apriori heuristic was introduced by Agrawal, et al to mine frequent patterns efficiently. According to Agrawal, Apriori heuristic highlights an anti-monotonic[2] property: any superset of an infrequent itemset cannot be frequent. In other words, if any length-k pattern is not frequent in the database, its length-(k + 1) superset patterns can never be frequent [AS94]. E.g., length-3 itemset {1, 3, 5} can not be frequent if one of its length-2 subset, {3, 5}, is not frequent. This becomes an important rule for limiting pattern searches, as will be seen in later examples.

---

[1] Apriori - is an efficient association rule mining algorithm, developed by Agrawal et al. (wikipedia). It is different from A priori, which is a Latin phrase meaning "from before, from the beginning" or less literally "before experience"(wikipedia).

[2] Support monotonicity- Given a transaction database TDB, Let X, Y be two itemsets in TDB. Then, (X ⊆ Y) => [support(Y) ≤ support(X)], Hence, if an itemset is frequent, then its superset might not be frequent. S*upport anti-monotonicity* - If an itemset is infrequent, all of its supersets must be infrequent

2.5. The Apriori Algorithm (to mine the complete set of frequent patterns in the database)

The Apriori algorithm was developed based on the Apriori heuristic. According to Agrawal, Apriori uses breadth-first search and bottom-up approach to generate frequent itemsets. The key observation behind Apriori is that all the subsets of a frequent itemset must be frequent. The essential idea of Apriori is to iteratively generate length-(k+1) candidates from the length-k frequent patterns (for $k > 0$); then check their corresponding occurrence frequencies in the database. The Apriori algorithm follows:

Apriori Algorithm

Input: transaction database *TDB* and minimum support threshold min_ sup

Output: the complete set of frequent patterns in *TDB* with respect to minimum support threshold min_ sup

Method:

"1. Scan TDB a first time to find $L1$[1](length-1 frequent itemsets);
2. for ($k = 2$; $L_{k-1} \neq \emptyset$; k + +)
{
(a) Generate C_k[2], the set of length-k candidates. A k-itemset X is in C_k iff every length-(k -1) subset of X is in L_{k-1};
(b) If $C_k = \emptyset$; then go to Step 3;
(c) Scan transaction database TDB once to count the support for every itemset in C_k;
(d) $L_k = \{X \mid (X \in C_k) \wedge (sup(X) \geq min\_sup)\}$;
}
3. Return all the frequent itemsets found."[P02].

Example 3: Customer Transactions (version 2)

Customer ID	Purchased Items	(Ordered) Frequent Items
1	1, 2, 3, 4, 5, 6, 9	1, 2, 3, 4, 5

[1] L_n - the complete set of length-n frequent patterns [P02].
[2] C_k – the complete set of length-n candidates [P02].

2	1, 2, 4, 7, 8	1, 2, 4
3	2, 3	2, 3
4	1, 3, 5, 11	1, 3, 5
5	1, 3, 4, 5, 12, 24	1, 3, 4, 5

Table 2: Customer Transactions (version 2)

Given min_length (3), the following steps[1] show how Apriori algorithm finds the complete set of frequent patterns.

Steps:

1. Scan table 2 once to find length-1 frequent itemsets that are contained in at least min_sup (3) transactions. They are {1, 2, 3, 4, 5}.
2. Generate length-2 candidates from length-1 frequent patterns. There are ${}_5C2$ (1, 2, 3, 4, 5) =10 length-2 candidates (combinations[2]) in total and they are {12, 13, 14, 15, 23, 24, 25, 34, 35, 45}.
3. Scan table 2 the second time to check length-2 candidates against min_sup(3). For instance, candidate {13} is contained in transactions 1, 4, 5 min_sup (3) times. Thus, {13}is a length-2 frequent pattern. There are four length-2 frequent patterns are discovered and they are {13, 14, 15, 35}. They are the length-2 frequent patterns. However, since they are unable to satisfy min_length (3), they are not in the output.
4. Generate the length-3 candidates. Only those length-3 itemsets in which every length-2 sub-itemset is a length-2 frequent pattern are qualified as candidates. E.g., itemset {135} is a length-3 candidate since its subsets 13, 15 and 35 are all

[1] Mining steps are derived from examples in [P02]

[2] Combination: number of ways of picking k *unordered* outcomes from n possibilities. (http://mathworld.wolfram.com/Combination.html) ${}_nC_k \equiv \binom{n}{k} \equiv \frac{n!}{k!\,(n-k)!}$,

length-2 frequent patterns. From the length-2 patterns, the only length-3 candidate generated is {135}.

5. Check length-3 candidate {135} against min_sup(3). It passes the test since it is contained in transactions 1, 4, 5. Since {135} also satisfies min_lengh(3), it will be in the mining result.
6. There is no length-4 candidate can be generated from only one length-3 frequent pattern. Moreover, the table only contains two transactions that have length-4 or longer itemset. Thus, mining process ends.

Thus, with a minimum support threshold min_sup (3), a minimum length support threshold min_length (3), there is only one frequent patterns discovered in the table. The frequent pattern {135} implies that 60% (3/5=60%) of customers who purchased two out of three items in the {135} also purchased the third item. Thus, the store should recommend the third item to the future customers who purchased two out of three items in the pattern. Sales might increase significantly since proper items are recommended to the proper customers.

According to Agrawal, Apriori heuristic achieves good performance gain by (possibly significantly) reducing the size of candidate sets. Otherwise, more candidates need to be generated and tested, which is a very time consuming process [AIS93]. For instance, without the Apriori heuristic, the length-3 itemset {127} must be checked to see whether it is a frequent pattern. With the Apriori heuristic, there is no need to check itemset {127} since subset {7} is not a frequent itemset.

Later on, based on the Apriori algorithm, Fault-tolerant Frequent Pattern Mining algorithm was proposed to mine more general, longer patterns from the database.

2.6. Fault-tolerant Frequent Pattern Mining (FT-FPM)

In 2000, fault-tolerant frequent pattern mining (FT-FPM) was introduced by Pei, J, et al [PTH01] to mine more general, longer patterns from the database. According to Pei, in FT-FPM, the "frequent patterns" discovered and the traditional frequent patterns do not have to be exact matches. These "frequent patterns" are called fault-tolerant frequent patterns. Fault-tolerance[1] is allowed in order to discover more general information. The definition of fault-tolerant frequent pattern follows. According to Pei, Given a fault tolerance δ ($\delta > 0$), P is an itemset, $|P| > \delta$. A transaction T = (tid, X) is said to **FT-contain** itemset P iff there exists $P` \subseteq P$ such that $P` \subseteq X$ and $|P`| \geq (|P| - \delta)$. The number of transactions in a database FT-containing itemset P is called the **FT-support** of P, denoted as FT_sup(P). Let B(P) be the set of transactions FT-containing itemset P. It is called the **FT-body** of P. Given (1) a **frequent-item support threshold $\mathbf{min\_sup^{item}}$** and (2) a **FT-support threshold $\mathbf{min\_sup^{FT}}$**. An itemset P is called a **fault-tolerant frequent pattern**, or **FT-pattern** in short, iff: 1) $FT\_sup(P) \geq min\_sup^{FT}$; and 2) for each item $x \in P$, $sup_{B(P)}(x) \geq min\_sup^{item}$, where $sup_{B(P)}(x)$ is the number of transactions in B(X) containing item x. Pei also states that the frequent-item support threshold is used to filter out infrequent item, since users may want to see patterns consisting of only items with statistic significance. On the other hand, FT-support threshold is used to capture frequent patterns in the sense of allowing at most δ (the fault tolerance) mismatches" [PTH01].

[1] Fault-tolerance – Partial item mismatch.

FT-Apriori Algorithm [PTH01]**:**

"Input: Transaction database TDB, frequent-item support threshold min_sup^{item}, FT-support threshold min_sup^{FT}, fault tolerance and length threshold min_length.
Output: The complete set of FT-patterns.
Method:

1. Scan TDB once, find the set F_1 of global frequent items. An item x is global frequent iff $sup(x) \geq min\_sup^{item}$;
2. Let $C_{\delta+1}$ be the set of all length-(δ+1) subsets of F_1. Let i = δ +1. Do {
   (a) Scan TDB, check candidate itemsets in Ci;
   (b) Let Fi be the set of FT-patterns in Ci; If (i ≥min_length) then output patterns in Fi;
   (c) If Fi is not empty, generate C_{i+1} from Fi. A length-(i+1) itemset X is in C_{i+1} iff every length-i subset of X is in F_i;
   (d) i = i+1;
   } until either F_{i-1} or C_i is empty;" [PTH01]

Let's look at table 3.

Customer ID	(Ordered) Frequent Items
1	1, 2, 3, 4, 5
2	1, 2, 4
3	2, 3
4	1, 3, 5
5	1, 3, 4, 5

Table 3: Customer Transactions (version3)

In table 3, given min_sup^{item} = 2, min_sup^{FT}= 3, min_length = 4, δ =1, consider an itemset X = {1345}. X's length-3 subsets {134}, {135}, {345} are contained in transaction 1, 4, 5 respectively[1]. Thus, X is FT-contained in the table at least[2] three times, which satisfies $min\_sup^{FT}(3)$. X's length satisfies min_length (4). Each item in X: {1}, {3}, {4}, {5} is contained in subsets {134}, {135}, {345} at least twice, which satisfies min_sup^{item} (2). Each subset has only one mismatch compares to X, which satisfies

[1] X's other subset such as {145} is also contained in transactions 1,5. In this case, since $min\_sup^{FT}(3)$ is already satified, it is not necessary to consider subset {145}
[2] At least – The same reason as in footnote 1.

fault-tolerance δ (1). Thus, X is a fault-tolerant frequent pattern. Compares to the frequent patterns mined previously, X is a length-4 frequent pattern ($|X| = 4$) with one item mismatch allowed. Such patterns also include {1245}, {1234}. Recall that we were unable to generate length-4 frequent itemset from frequent pattern mining. Since we use fault-tolerance, longer patterns are generated.

Fault-tolerant frequent pattern mining with classifiers is based on Fault-tolerant frequent pattern mining, also, it is based on classification.

2.7. Classification[1]

Classification is a data mining technique used to predict membership of objects in the classes based on one or more categorical dependent variables. In Classification, objects are divided into different classes based on one or more classification variables called Classifiers. Thus, in each class, objects share similar properties.

Base on Fault-tolerant frequent pattern mining and classification, this paper demonstrates how to mine fault-tolerant frequent patterns with classifiers in the next chapter.

[1] Classification – Definition is from http://www.statsoft.com/textbook/stdatmin.html.

CHAPTER 3

DESIGN AND IMPLEMENTATION

In this chapter, we present:

1. Fault-tolerance analysis
2. Constraint-based pattern growth frequent pattern mining .vs. fault-tolerant frequent pattern mining with classifiers
3. Fault-tolerant frequent pattern mining with classifiers (CFT-FPM)
4. Example of CFT-FPM
5. Apply CFT-FPM to real world data

Before CFT-FPM is introduced, let's have a better understanding of fault-tolerance, which was first introduced by Pei in [PTH01].

3.1. Fault-tolerance Analysis

According to Pei, in FT-FPM, fault-tolerance is the mismatch item count between fault-tolerant frequent pattern and frequent pattern [PTH01]. The smaller the fault-tolerance, the similar the frequent pattern and the fault-tolerant pattern, the more accurate the mining result. In order to be frequent, fault-tolerant frequent patterns found should be as close to the frequent patterns as possible. The *maximum value* of fault-tolerance is the length of the frequent pattern (|frequent pattern|). If the fault-tolerance is set to the

maximum value, too many infrequent patterns will be generated, mining will be completely pointless.

The *minimum value* of fault tolerance is zero. In this case, we look for fault-tolerant patterns that match the frequent patterns exactly and this fault-tolerant frequent pattern mining process becomes frequent pattern mining (FPM). Thus, frequent pattern mining is a special case of fault-tolerant frequent pattern mining when fault-tolerance is set to zero.

In a movie rental store, the manager predicts new movie rental possibilities based on a set of old movie rentals. For example, if he is looking for frequent movie sets (frequent patterns) of size (length) five, fault tolerance should be between the min value (0) and the max value(5). He might not want to set fault-tolerance to a higher value of four or five; if he does, he will get too many infrequent movie sets. It is more realistic for him to set the fault-tolerance to a lower value of one or two (three will generate more infrequent patterns than one or two). This means maximum two old movie mismatches still count a certain customer in. Follow the same logic; if we are looking for frequent patterns of length 50, it is more realistic to set fault-tolerance to a relevant lower value compares to the length of the frequent patterns that we are looking for, such as a number that is less than 20. In both cases, fault-tolerance is set to a relevant lower value compares to the length of the frequent patterns in order to get patterns that are frequent. The higher the fault-tolerance, the more infrequent patterns are mined, the less accurate the result; and wise versa.

In a supermarket, the majority of shoppers buy many items each visit. The same logic follows. If we are looking for frequent itemsets of size ten, the fault-tolerance

should be set to a relatively lower value, such as four or some other lower value compared to the number of items purchased (ten).

In the medical field, in order to predict a future symptom of a patient, doctors must look at the patient's medical history, including a set of previous symptoms that the patient had in the past. In this case, fault-tolerance needs to be as small as possible since the prediction needs to be as accurate as possible, otherwise, the patient will receive wrong treatment which is extremely dangerous.

Thus, to set the fault-tolerance, we must:

1. Identify the type of industry or the field of application.
    a. If it is a field such as a movie rental store or a supermarket, in which a small error does not incur a large penalty, go to step two.
    b. If it is a field such as a medical field or a chemical research laboratory, in which a small error incurs a large penalty, the dataset and the prediction need to be as accurate as possible. In this case, it is better not to apply fault-tolerance at all.

    Note: Fault-tolerance is used only if the frequent patterns discovered are: not long enough, or not general enough (the number of frequent patterns discovered is too small). If the frequent patterns discovered are long and general enough, fault-tolerance is not necessary.
2. Set the length of the traditional frequent itemset and determine the value of fault-tolerance. Fault-tolerance should be set to a relatively lower value compares to the length (size) of the frequent itemset. The greater the fault-tolerance, the more infrequent patterns are discovered, the less accurate the prediction.

In CFT-FPM, the more attributes the records have, the more difficult it is to use CFT-FPM. When we categorize data, we must pre-process the data, which includes sorting. If the records have n (n is a positive integer) attributes and each attribute has m (m is a positive integer) different values, we have m^n attribute combinations. It is very tedious to match up each combination with a particular fault-tolerance and then do FT-FPM in each combination. This paper demonstrates CFT-FPM with only one attribute.

Pei introduced the method of constraint-based frequent pattern mining in [PH02]. Many readers might consider that classifiers are a type of constraints since they constraint data into different categories. The following section explains the difference between Pei's constraint-based frequent pattern mining and the proposed fault-tolerance frequent pattern mining.

3.2. Constraint-based frequent pattern mining .vs. fault-tolerant frequent pattern mining with classifiers.

According to Pei, constraint-based frequent pattern mining[1] integrates users' interest into the mining result. The purpose of adding constraints to pattern mining is to mine patterns tailor to users' interest [PH02]. The purpose of adding classifiers to FT-FPM is to integrate various degrees (values) of fault-tolerance based on data properties to discover more realistic patterns. Thus, FT-FPM with classifiers and constraint based frequent pattern mining are different.

Let's look at fault-tolerant frequent pattern mining with classifiers in the next section.

[1] Constraint-based frequent pattern mining - In Pei [PH02], Pei introduced constraint-based pattern growth mining, which is a type frequent pattern mining. Pattern growth method is an alternative to the Apriori method.

3.3. Fault-tolerant frequent pattern mining with classifiers (CFT-FPM)[1]:

Let's consider a transaction, T, together with its properties (classifiers) in TDB; e.g., T = (tid, classifier1, classifier2…classifier n, X), where tid the transaction-id; classifier1, classifier2…classifier n are the **Classifiers**, which are the properties associate with itemset X. For itemsets X, Y, a transaction T = (tid, classifier1, classifier2…classifier n, X) is said to contain itemset Y iff:

1. $Y \subseteq X$
2. Y's classifiers value = X's classifiers value.

   E.g., (classifierY1 = classifier X1) & (classifierY2 = classifier X2) & …& (classifier Yn = classifier Xn)

The catch is: we want to mine fault-tolerant frequent patterns based on certain classifier(s) and categorize data based on their properties, then apply various degree of fault-tolerance to each category.

In CFT-FPM, since we mine fault-tolerant patterns tailored to the classifiers, we have **n** (n is a positive integer) fault tolerances instead of one in FT-FPM, where:

1) n = number of categories of items divided by classifiers

2) fault-tolerance list: Delta = $\{\delta_1, \delta_2, \ldots, \delta n\}$(δi is a non-negative integer, where $0 \leq i \leq n$)

Each fault-tolerance is corresponding to one category based on the properties of itemset. The next example shows how to mine fault-tolerant frequent patterns with classifiers.

[1] CFT-FPM – the definition of CFT-FPM is derived from Pei's FT-FPM definition in [PTH01].

3.4. Example of CFT-FPM

Example 4: A game Store manager wants to find out to whom a new play station game should be recommended. The sales database has information about customers' gender, age range and the purchase history. We'd like to mine the database with classifiers since the possibility to purchase a new game is not only based on purchase history, but also depends on other properties such as customers' age and gender. Table 4 contains games sales information.

Customer ID	Customer Gender	Customer Age Range	Purchased Games
1	M	18-30	2, 3, 4, 5
2	F	18-30	1, 2, 3, 4, 5
3	M	18-30	1, 2, 4
4	F	18-30	1, 2, 4
5	F	Above 50	2, 3, 4, 5
6	F	18-30	2, 3
7	F	Above 50	1, 4, 5
8	F	18-30	1, 3, 5
9	M	18-30	1, 4, 5
10	F	18-30	1, 3, 4, 5

Table 4: Customer Transactions (Games)

In table 4, there are three types of customers: males under age 30, females under age 30 and females over age 50. According to entertainment software association's research on game players in 2002, 19% of gamers are above 50 years old, 55% of the gamers are males [ESA02]. Thus, it is reasonable to consider that males who are under age 30 have the highest possibility to purchase a new game without having purchased lots of games in the past. Females who are under age 30 have less possibility since female has less interest in gaming compare to males in general. Females who are above age 50

have the lowest possibility since older people has less interest in gaming compare to younger generation in general. In other words, for a female who is under age 50 to buy a new game, she has to be a game fan whose game purchase history is outstanding. Fault-tolerances are set based on this consideration. Since the possibility for customers to buy games is: males under 30 > females under 30 > females above 50, it is reasonable to set fault-tolerance to '2' for males under 30, to '1' for females under 30, to '0' for females above 50 years old. (Database might have other types of people who do not fit in above three categories. However, we only consider three categories in this example for demonstration purposes.)

In table 4 above, there are ten customers. Each record stores information about a customer along with the games purchased. It contains:

1) Customer ID: primary key

2) Customer Gender, Customer Age Range: classifiers

3) Purchased Games: itemset

To mine CFT-patterns in table 4, we set input parameters to following values:

Input:

1) TDB (transaction database): Customer Transactions

2) minimum item support threshold: min_sup^{item} (2)

3) FT-support threshold: min_sup^{FT} (3)

4) fault-tolerance list: Delta =

{0: for females above 50

1: for females under 30

2: for males under 30}

Table 5 shows classifier and fault-tolerance matching pairs.

Customer Gender	Customer Age Range	Fault-tolerance
F	Above 50	0
F	18-30	1
M	18-30	2

Table 5: classifier & fault-tolerance matching pairs

5) minimum pattern length: min_length = 4

Output: CFT- FP for each category

Procedure:

1. Categorize data based on two classifiers: Customer Gender, Customer Age Range

Customer ID	Customer Gender	Customer Age Range	Purchased Games
2	F	18-30	1, 2, 3, 4, 5
4	F	18-30	1, 2, 4
6	F	18-30	2, 3
8	F	18-30	1, 3, 5
10	F	18-30	1, 3, 4, 5

Category 1(females under 30)

Customer ID	Customer Gender	Customer Age Range	Purchased Games
1	M	18-30	2, 3, 4, 5
3	M	18-30	1, 2, 4
9	M	18-30	1, 4, 5

Category 2 (males under 30)

Customer ID	Customer Gender	Customer Age Range	Purchased Games
5	F	Above 50	2, 3, 4, 5

7	F	Above 50	1, 4, 5

Category 3 (females above 50)

Table 6: Customer Transaction categories (3 categories)

2. Eliminate the categories that have fewer than three records.

Since $min\_sup^{FT} = 3$, each candidate category must have at least $min\_sup^{FT}(3)$ records. We delete category 3 since it has only two records. Thus, we have two categories after elimination.

Group 1: 2, 4, 6, 8, 10

Group 2: 1, 3, 9

3. We use FT-FPM to mine category 1 and 2 separately with different fault-tolerances.

Customer ID	**Customer Gender**	**Customer Age Range**	**Purchased Games**
2	F	18-30	1, 2, 3, 4, 5
4	F	18-30	1, 2, 4
6	F	18-30	2, 3
8	F	18-30	1, 3, 5
10	F	18-30	1, 3, 4, 5

Table 7: Category 1(Females under 30)

Table 7 contains the data for females who are under age 30, which has fault-tolerance of one. Following steps[1] demonstrate mining fault-tolerant patterns with classifiers.

Steps:

1) Scan table 7 the first time to find the length-1 itemsets that appear in at least min_sup^{item} (2) records[2]. They are: {1, 2, 3, 4, 5}.

[1] The steps are derived from the fault-tolerance mining process in [PTH01].
[2] They are also called the global frequent items [PTH01].

2) From {1, 2, 3, 4, 5}, we generate ${}_5C2\,(1, 2, 3, 4, 5) = 10$ combinations of length-2 candidates and they are {12, 13, 14, 15, 23, 24, 25, 34, 35, 45}. According to the lemma 2.1 in [PTH01][1], it is not necessary to check them against the $min\_sup^{item}(2)$.

3) Scan table 7 the second time to check length-2 candidates against $min\_sup^{FT}(3)$. For instance, candidate {2,5} is FT-contained in customers 2(contains items 2, 5), 4(contains item 2), 6(contains item 2), 8(contains item 5), 10(contains item 5). Thus, $FT\_sup\{2,5\} = 5 \geq min\_sup^{FT}(3)$. It turns out that every length-2 candidate is a FT-pattern. However, since their length is only 2, they are not in the output.

4) Generate length-3 candidates from length-2 FT- patterns. According to Pei, a length-(k+1) candidate is generated iff its every length-k subset is an FT-pattern [PTH01]. For instance, {123} is a length-3 candidate since its subsets {12}, {13}, and {23} are length-2 FT-patterns. There are ten length-3 candidates generated: {123, 124, 125, 134, 135, 145, 234, 235, 245, 345}.

5) Scan table 7 the third time to check length-3 candidates against $min\_sup^{FT}(3)$ and $min\_sup^{item}(2)$. For instance, candidate {123}'s subsets {12}, {23}, {13} are contained in customer 2 (contains {12}), 6(contains {23}), 8(contains {13})[2]. which means, $FT\_sup(123) \geq min\_sup^{FT}(3)$. Length-1 item 1, 2, 3, each appears in customer 2, 6, 8 at least $min\_sup^{item}(2)$ times. Thus, candidate {123} is a FT-pattern. It turns out that all the length-3 candidates are length-3 FT-patterns. However, they are not part of output since min_length = 4.

[1] Lemma 2.1 – Let X be an itemset contains (δ +1) global frequent items. X is a FT-pattern iff $FT\_sup(X) >= min\_sup^{FT}$

[2] Subset {13} is also contained in customer 10. Since $min\_sup^{FT}(3)$ is already satisfied, it is not necessary to consider customer 10.

6) From length-3 FT-patterns, 5 length-4 candidates are generated. They are: {1234, 1235, 1245, 1345, 2345}.

7) Scan table 7 the fourth time to check length-4 candidates against $min\_sup^{FT}(3)$ and $min\_sup^{item}(2)$. For instance, for candidate {1234}, subsets {123}, {124}, {134} are contained in customer 2, 4, 10 respectively, which means $FT\_sup\{1234\} \geq min\_sup^{FT}(3)$. Length-1 items 1, 2, 3, 4 are all contained in customer 2, 4, 10 at least min_sup^{item} (2) times. Thus, candidate {1234}is a FT-pattern. However, for candidate {1235}, subsets {135}, {123} are contained in customer (8, 10), (2) respectively, which satisfies $min\_sup^{FT}(3)$ but not min_sup^{item} (2) since item '2' is contained in customer (8, 10, 2) only once (min_sup^{item} (2) is not satisfied). Thus, candidate {1235} is not a FT-pattern. Only three length-4 FT-patterns are found and they are {1245}, {1234}, {1345}.

8) There is no length-5 candidate can be generated since there are only three length-4 FT-patterns (To generate a length-k candidate, we need at least k length-(k-1) FT-patterns according to Pei). Thus, mining process terminates for category 1. FT-patterns {1245}, {1234}, {1345} are the CFT-FPs for category 1. Let's look at category 2 in table 8, which has a fault-tolerance of 2.

Customer ID	**Customer Gender**	**Customer Age Range**	**Purchased Games**
1	M	18-30	2, 3, 4, 5
3	M	18-30	1, 2, 4
9	M	18-30	1, 4, 5

Table 8: Category 2 (males under 30)

9) Scan table 8 once to find the length-1 items that appear in at least min_sup^{item} (2) records. They are: {1, 2, 4, 5}.

10) From {1, 2, 4, 5}, we generate ${}_4C3$ (1, 2, 4, 5) = 4 combinations of length-3 candidates[1] and they are {124, 125, 145, 245}. It is not necessary to check them against min_sup^{item} according to the lemma 2.1 in [PTH01].

11) Scan table 8 the second time to check length-3 candidates against $min\_sup^{FT}(3)$. For instance, candidate {124}is FT-contained in customer 1(contains {2})[2], customer 3(contains {1}), customer 9 (contains {4}), which satisfies min_sup^{FT} (3). Thus, candidate {124} is a FT-pattern. It turns out that every length-3 candidate is a FT-pattern. However, since their length is only 3, they are not in the output.

12) Generate length-4 candidates from length-3 FT-patterns. There is only one length-4 candidate generated from length-3 FT-patterns and it is {1245}.

13) Scan table 8 the fourth time to check candidate {1245}against $min\_sup^{FT}(3)$ and min_sup^{item} (3). {1245}'s subsets {245}, {124}, {145}are contained in customer 1, 3, 9 respectively, which satisfies $min\_sup^{FT}(3)$. Each length-1 item 1, 2, 4, 5 is contained in customers 1, 3, 9 at least min_sup^{item} (2) times. Since {1245} also satisfies min_length(4). Thus, it is a length-4 FT-pattern and it is in the output.

14) No length-5 candidate can be generated from only one length-4 FT-patterns. Mining process terminates and FT-pattern {1245} is the CFT-FP for category 2.

Therefore, in total, there are four length-4 CFT-patterns discovered from

[1] Since fault-tolerance is 2, we do not consider length-2 patterns, according to Pei in [PTH01].
[2] Customer 1 also contains item {4}. In this case, it is not necessary to consider {4} since min_sup^{FT} is already satisfied. Same reason applies to the fact that it's not necessary to count item {2} and {4} for customer 3.

CFT-FPM. They are {1234, 1245, 1345} for category 1(females under age 30) and {1245} for category 2(males under age 30).

Thus, the game store might consider recommending certain new game that is related to old games {1},{2}, {3}, {4}, {5} to females who are under age 30 and purchased three out of four games in the game sets {1234}, {1245}or {1345}. The game store should also recommend certain new game that is related to old games {1}, {2}, {4}, {5} to males who are under 30 and purchased two out of four games in the game set {1245}.

3.5. CFT-Apriori Algorithm[1]:

Input: 1) TDB (transaction database)

2) minimum item support threshold: min_sup^{item}

3) minimum CFT-support threshold (same for every category): min_sup^{CFT}

4) fault-tolerance list: deltas = $\{\delta_1, \delta_2, \ldots, \delta n\}$

Classifier 1	Classifier 2	Classifier n	Fault tolerance
C_{11}	C_{21}	Cn_1	δ_1
C_{12}	C_{22}	Cn_2	δ_2
…	…	…	…
C_{1n}	C_{2n}	Cn_n	δn

Table 9: classifier & fault-tolerance matching pairs

6) minimum length threshold: min_length

Output: The complete set of CFT-FPs that satisfies the above input requirements.

Method:

1. Categorization

[1] CFT-FPM Apriori algorithm is derived from Pei's FT-FPM Apriori.

2. for (k = 1; k ≤ number of categories; k = k + 1)

   {

   Mine fault-tolerant frequent patterns with the corresponding fault-tolerance.

   }

3. Gather all the fault-tolerant frequent patterns generated in each category.

3.6. Methodology

The project contains five steps:

1) Generate suitable dataset for CFT-FPM by using program gen.java[1]
2) Input dataset into the CFTmine.java[2].
3) Categorization.
4) Loop though transaction categories and apply fault-tolerant frequent pattern mining algorithm to each category.
5) Gather the total result and output to the screen.

This project utilized the following software and hardware:

1) Dataset: retail.txt
2) Programming language: Java
3) Operating system: Sun Solaris Unix (Oklahoma State University Computer Science department CSA server)

[1] gen.java – is the java program to generate proper dataset for CFT-FPM.
[2] CFTmine.java – is the java program composed by author to mine fault-tolerant frequent patterns with classifiers.

3.6.1. Select real world data

By searching the online data repository and other resources, we were unable to find a proper dataset to fit the CFT-FPM. However, we found a dataset that fits the FT-FPM which we can modify to fit the CFT-FPM.

The test dataset "retail" is downloaded from http://fimi.cs.helsinki.fi/data/. It was donated by Tom Brijs and contains the retail market basket data from an anonymous Belgian retail store [BSVW99]. The dataset contains 88,163 transactions and each transaction contains item purchased in each transaction[1]. Below are the first ten transactions in the dataset and it contains item identification numbers.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32
33 34 35
36 37 38 39 40 41 42 43 44 45 46
38 39 47 48
38 39 48 49 50 51 52 53 54 55 56 57 58
32 41 59 60 61 62
3 39 48
63 64 65 66 67 68
32 69

The dataset is sparse and suitable for market basket analysis. However, since the dataset contains private sale information about the store, for security purposes, the dataset does not specify the details of the data, such as which number means what item. In this case, this paper assumes that this store sells video games.

In order to generate a dataset which is more suitable for CFT-FPM, an "age" attribute into the dataset as the first column by a program named "Gen.java"[2]. **"**Age**"**[3] is

[1] We emailed Tom Brijs at tom.brijs@luc.ac.be for acknowledgement. However, it said an error occurred and the email was bounced back.
[2] Gen.java - This program is written by the author of this paper.
[3] Age – is generated by Java function random (), which is an uniform distribution over a given number range.

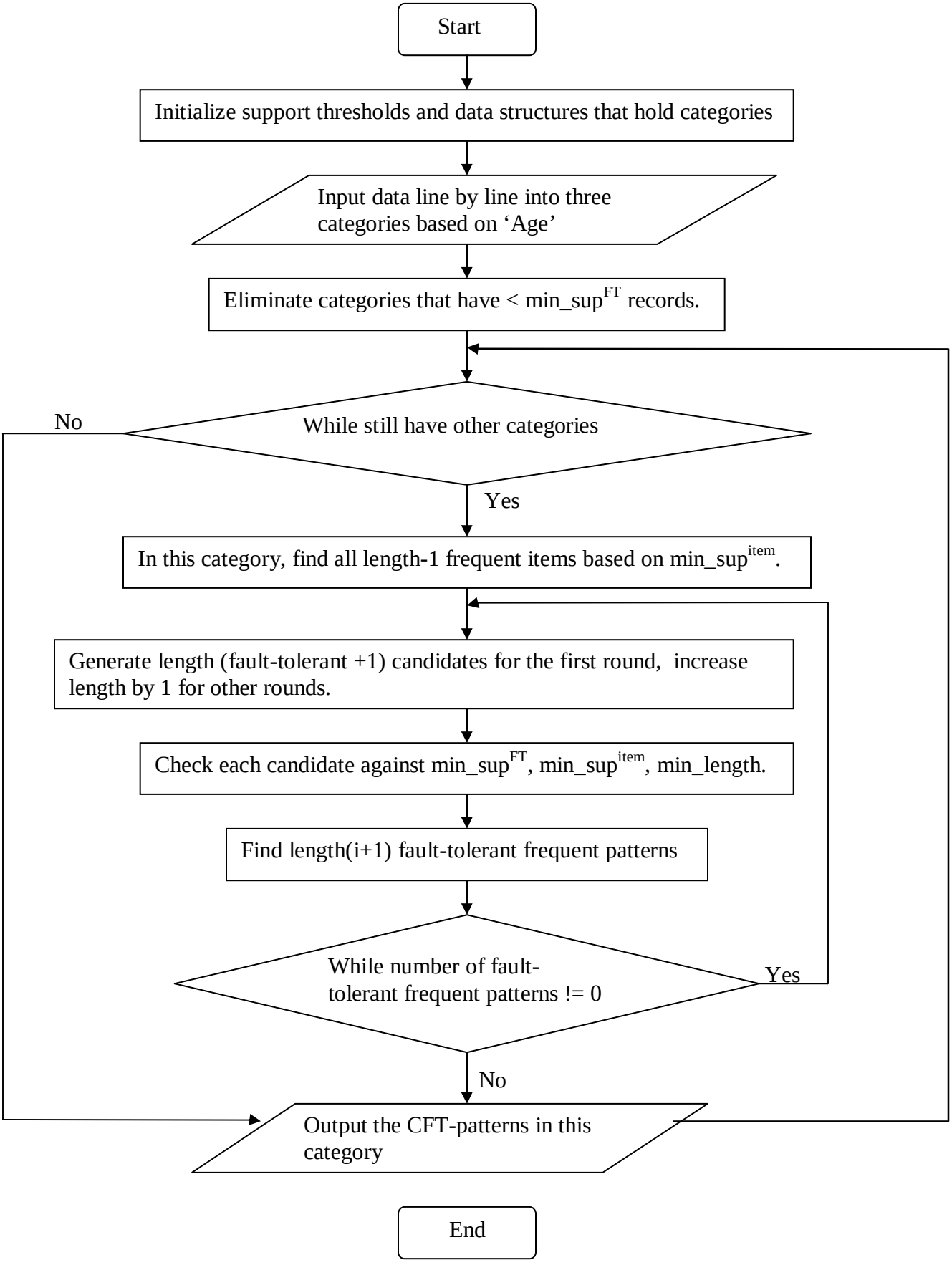
a random integer ranges between 16 and 80, which is suitable for customers. Below are the first ten transactions in the generated dataset:

44 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
70 30 31 32
77 33 34 35
60 36 37 38 39 40 41 42 43 44 45 46
23 38 39 47 48
75 38 39 48 49 50 51 52 53 54 55 56 57 58
66 32 41 59 60 61 62
76 3 39 48
45 63 64 65 66 67 68
21 32 69

3.6.2. Methodology diagram

Appendix D demonstrates step 2, 3, 4, 5 (mentioned in page 31) in this methodology.

Figure 1: CFTmine.java's methodology diagram


Start
Initialize support thresholds and data structures that hold categories
Input data line by line into three categories based on 'Age'
Eliminate categories that have < min_sup^FT records.
While still have other categories
No
Yes
In this category, find all length-1 frequent items based on min_sup^item.
Generate length (fault-tolerant +1) candidates for the first round, increase length by 1 for other rounds.
Check each candidate against min_sup^FT, min_sup^item, min_length.
Find length(i+1) fault-tolerant frequent patterns
While number of fault-tolerant frequent patterns != 0
Yes
No
Output the CFT-patterns in this category
End

CHAPTER 4

RESULTS AND FINDINGS

In this chapter, we present the results of functionality tests and also give a simple performance test for time and space complexity analysis, list as following:

1. FT-FPM .vs. CFT-FPM
    - FT-FPM
    - CFT-FPM
2. CFT-FPM performance test
    - Run Time .vs. Number of records
    - Number of patterns .vs. Number of records
    - Number of IO operations .vs. Number of records

4.1. FT-FPM .vs. CFT-FPM (tested on the first 10,000 records in the dataset)

4.1.1. FT-FPM result

We use fault-tolerant frequent pattern mining (FT-FPM) to mine the first 10,000 records in the dataset[1]. We set the parameters $min\_sup^{item} = 99$, $min\_sup^{FT} = 100$, min_length = 5, fault-tolerance = 1. Below is the empirical result for FT-FPM:

[1] We were unable to run the whole dataset on CSA because it gave "out of memory" error. our process on CSA only can be allocated 3800MB memory. Thus, we chose the first 10,000 records only.

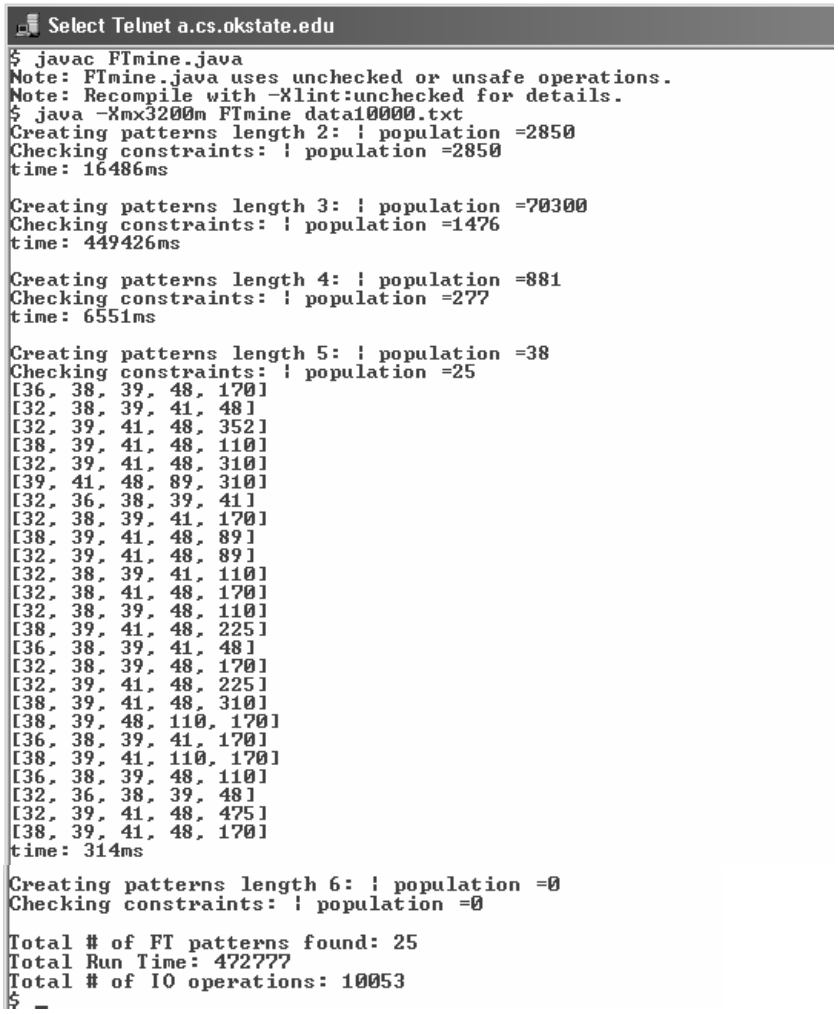


Figure 2: dataset 10,000 for FT-FPM

1. There are total of 2,850 length-2 patterns generated.

   After checking against min_sup^{FT} and min_sup^{item}, all 2,850 patterns pass the test.

   Time cost for above two steps is 16,486 milliseconds.

2. There are total of 70,300 length-3 patterns generated.

   After checking against min_sup^{FT} and min_sup^{item}, only 1,476 itemsets pass the test.

   Time cost for above two steps is 449,426 milliseconds.

3. There are total of 881 length-4 patterns generated.

   After checking against min_sup^{FT} and min_sup^{item}, only 277 itemsets pass the test.

Time cost for above two steps is 6,551 milliseconds.

4. There are total of 38 length-5 patterns generated.

   After checking against min_sup^{FT} and min_sup^{item}, only 25 itemsets pass the test.

   Time cost for above two steps is 314 milliseconds.

5. No length-6 fault-tolerant frequent patterns can be generated from the first 10,000 records. Process ends.

Thus, the number of fault tolerant frequent pattern generated based on the inputs is 25. Total time cost is 16,486 + 449,426 + 6,551 + 314 = 472,777ms, approximately 473 seconds. The number of IO operations is 10,053.

The result indicates that in each of 25 FT-patterns, 100 out of 10,000 customers bought 4 out of 5 games. Thus, to future customers who bought 4 out of 5 games in each FT-pattern, the store should recommend the 5^{th} games in the corresponding FT-pattern.

4.1.2. CFT-FPM result

We use fault-tolerant frequent pattern mining with classifiers (CFT-FPM) to mine the first 10,000 records in the dataset[1]. We set the parameters $min\_sup^{item} = 99$, $min\_sup^{FT} = 100$, min_length = 5, fault-tolerance = {2, 1, 0} (fault tolerance is 2 for age group under 30, since the possibility for young people buy games is the highest; 1 for age group between 30 and 50 inclusive; 0 for age group above 50, since the possibility for older people buy games is the lowest). Below is the empirical result of CFT-FPM:

[1] We were unable to run the entire dataset on Oklahoma State University CSA server because it gave an "out of memory" error. Thus, we chose to use the first 10,000 records only.

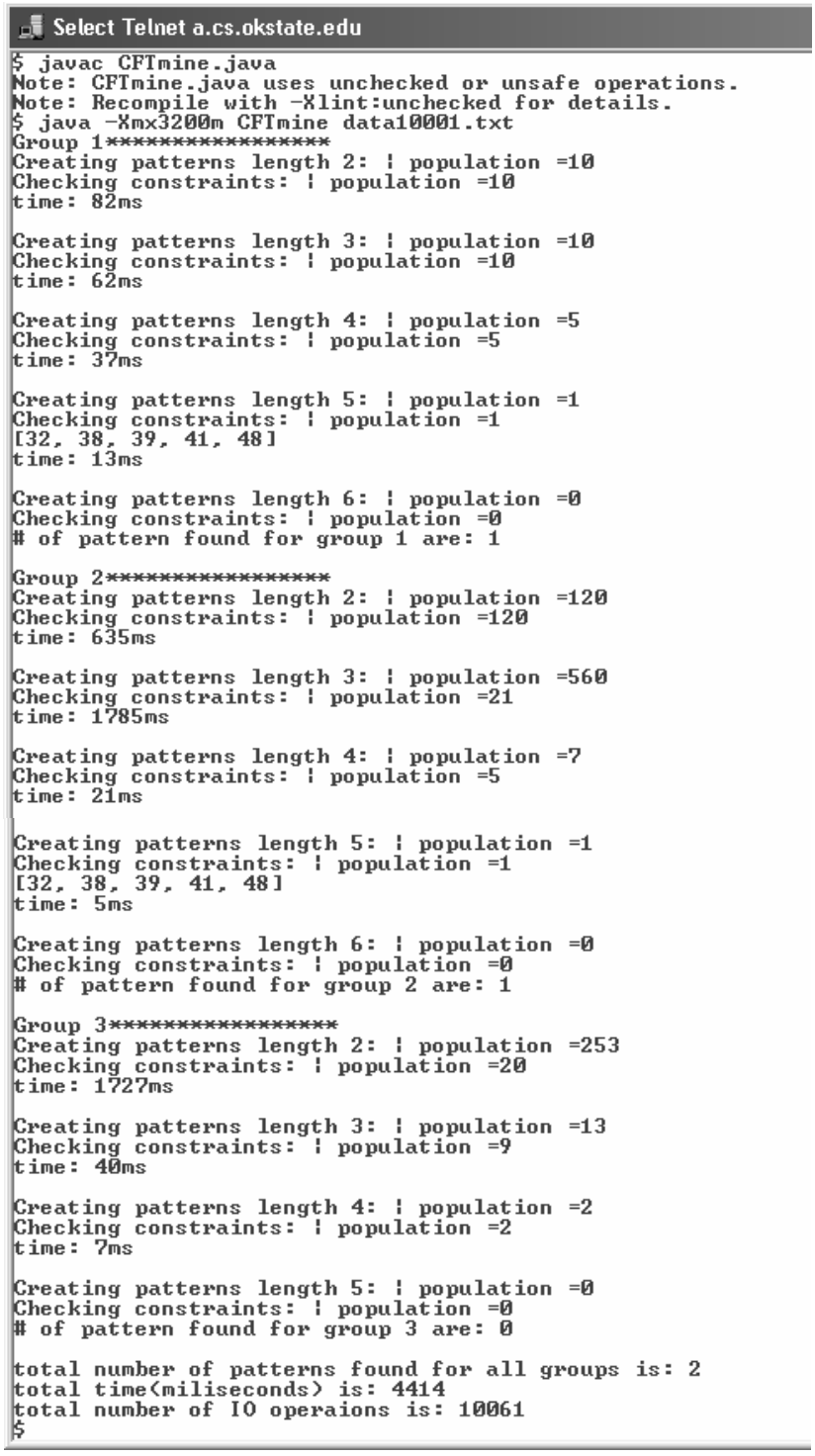


Figure 3: dataset 10,000 for CFT-FPM

1. For age group under 30 (group 1), we found one CFT-pattern satisfies the input parameters and it is pattern: [32, 38, 39, 41, 48]

2. For age group between 30 and 50, one CFT-pattern is found and it is also pattern: [32, 38, 39, 41, 48]

3. For age group above 50, no CFT-pattern is found.

Thus, total patterns found is 2; Total time cost is 4,414 milliseconds, approximately 4.4 seconds. The number of IO operations is 10,061.

The result indicates that 100 out of 10,000 people who are under 30 years old bought 3 out of 5 games in CFT-pattern [32, 38, 39, 41, 48]. 100 out of 10,000 people who are between age 30 and 50 inclusive bought 4 out of 5 games in pattern [32, 38, 39, 41, 48]. Thus, to future customers under age 30 who bought 3 out of 5 games in the pattern, the store should recommend them with the other 2 games in this CFT-pattern. Also, to future customers who are between age 30 and 50 inclusive and bought 4 out 5 games in the pattern, the store should recommend them with the 5^{th} games in this CFT-pattern.

4.1.3. FT-FPM .vs. CFT-FPM result

Compares with FT-FPM that has the same input parameters:

1) CFT-FPM is much faster since dataset are divided into n groups, less length k itemsets generated and tested in each group, thus, less length k+1 itemsets are generated and tested, and so on.
2) CFT-FPM generated less patterns since dataset are divided into n groups; there are less patterns satisfy the min_sup^{item} and min_sup^{FT}.

4.2. CFT-FPM Performance Test

4.2.1. Run Time .vs. Number of records

Figure 4 shows the Run time (seconds) .vs. Number of records.

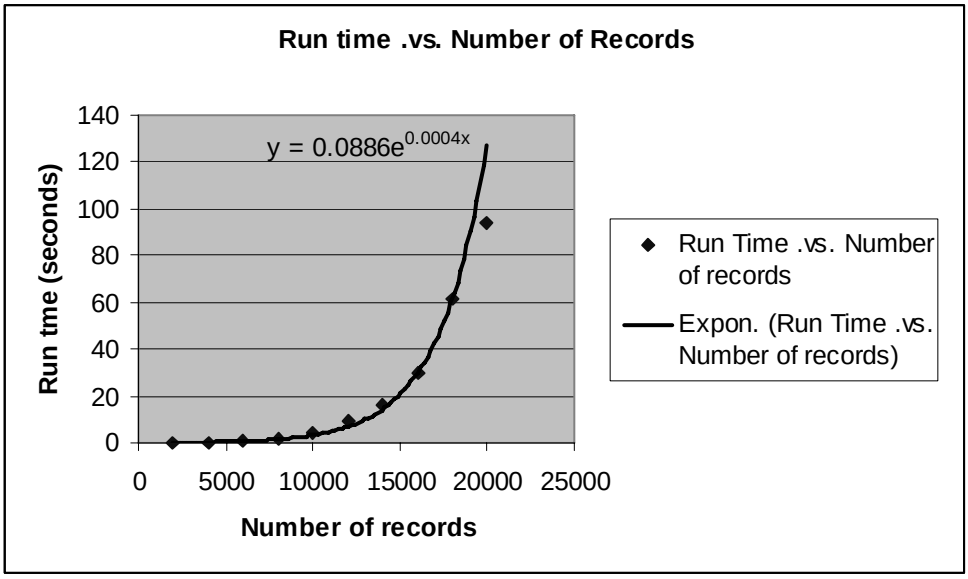


Figure 4: Run time (seconds) .vs. Number of records

In Figure 4[1], run time increases approximately exponentially when number of records increases. Run time = $0.0886e^{0.0004*\text{Number of Records}}$.

4.2.2. Number of patterns .vs. Number of records

Figure 5 shows the performance for Number of Patterns/Number of Records.

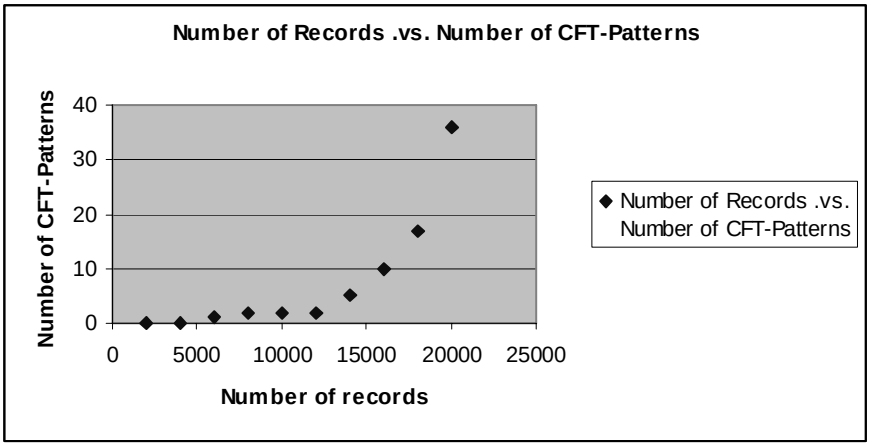


Figure 5: Number of Patterns .vs. Number of Records

[1] We used Excel to graph and fit the data.

In Figure 12, number of patterns increases approximately when number of records increases (the data did not fit any curves).

4.2.3. Number of IO operations .vs. Number of records

Figure 6 shows the performance for Number of IO operations/Number of Records.

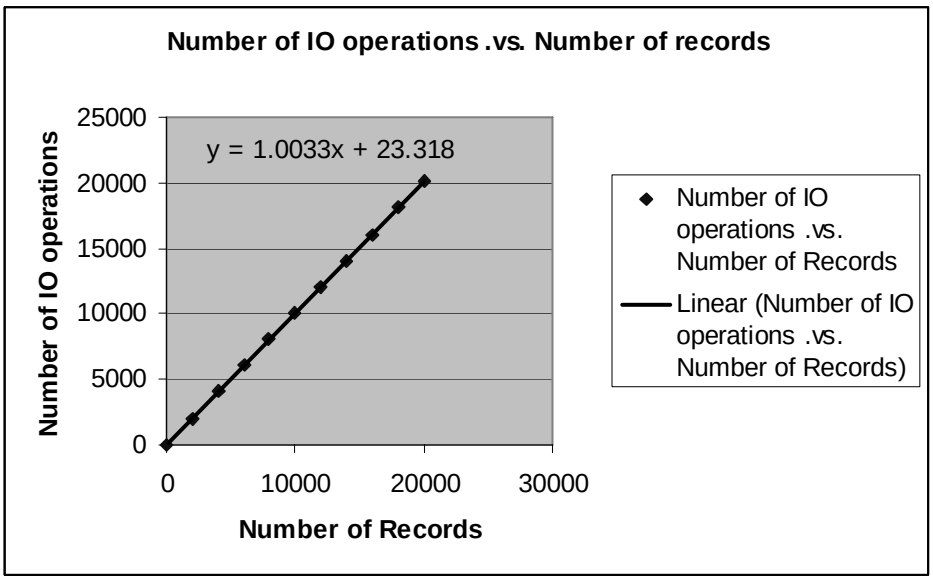


Figure 6: Number of IO operations .vs. Number of Records

In Figure 13, data are well approximated by a straight line through the origin with slope 1.0033, Number of IO operations = 1.0033 * Number of records + 23,318.

From the experimental results, we have the following observations:

1) Compares with FT-FPM, CFT-FPM finds fault-tolerant patterns based on data properties. Thus, less patterns are founded since constraints are added.
2) Compares with FT-Apriori, CFT-Apriori is much faster since entire dataset is divided into n groups, thus, in each group, less length-k itemsets are generated and checked. Thus, less length-(k+1) are generated and checked, and so on.
3) Since CFT-Apriori is based on FT-Apriori, it is both memory-consuming and time consuming over large number of records.

CHAPTER 5

CONCLUSIONS

5.1. Summary and Conclusions

This paper proposes to mine fault-tolerant frequent patterns with classifiers (CFT-FPM). CFT-FPM mines fault-tolerant frequent patterns (FT-FP) based on the data's properties. Furthermore, the paper implements and tests CFT-FPM over modified retail supermarket shopping basket data. This research observed that:

1) Compares to FT-FPM, CFT-FPM finds fault-tolerant patterns based on classifiers. Thus, patterns closer to reality are found since classifiers are added.
2) In CFT-FPM, since the dataset is classified into n[1] smaller groups that contain fewer records compared to the entire dataset, support thresholds are harder to be reached. Thus, compared to FT-FPM, CFT-FPM finds fewer patterns.
3) In CFT-FPM, since the entire dataset is classified into n smaller groups, in each group, less length-k itemsets are generated and checked, thus, less length-(k+1) are generated and checked, and so on. Thus, compared to FT-Apriori, CFT-Apriori is much faster.
4) CFT-Apriori is based on FT-Apriori and FT-Apriori is known to be very memory-consuming and time-consuming over large number of records, CFT-FPM is also very memory-consuming and time-consuming.

[1] n – is a random non-negative integer.

5.2. Future Work

More efficient CFT-FPM algorithms need to be developed in the future, such as

1) Algorithms that are based on H-Mine or the Pattern-Growth Method.

2) Algorithms that consume less memory.

3) Algorithms that can sort fast over multiple attributes (classifiers).

BIBLIOGRAPHY

[AIS93] Agrawal R, T. Imielinski, and A. Swami. "Mining association rules between sets of items in large databases". In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93),* pages 207–216, Washington, DC, May 1993

[AS94] Agrawal R and R. Srikant. "Fast algorithms for mining association rules". In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, Sept. 1994.

[AS95] Agrawal R and R. Srikant. "Mining sequential patterns". In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, Mar. 1995.

[B98] Bayardo R. J. "Efficiently Mining long patterns from databases". In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 85–93, Seattle, WA, June 1998.

[BMS97] Brin S, R. Motwani, and C. Silverstein. "Beyond market basket: generalizing association rules to correlations". In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97),* pages 265–276, Tucson, Arizona, May1997.

[BSVW99]Brijs T., Swinnen G., Vanhoof K., and Wets G. (1999), The use of association rules for product assortment decisions: a case study, in: Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining, San Diego (USA), August 15-18, pp. 254-260. ISBN: 1-58113-143-7

[DL99] Dong G and J. Li. "Efficient mining of emerging patterns: discovering trends and differences". In *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, pages 43–52, San Diego, CA, Aug. 1999.

[ESA02] http://www.theesa.com/facts/gamer_data.php

[FAL97] Fischoff S, J.Antonio, D.Lewis. "Favorite Films and Film Genres As A Function of Race, Age, and Gender (Paper Originally Presented at American Psychological Association Convention, Chicago, August, 1997) Journal of Media Psychology, Volume 3, Number 1, Winter, 1998 http://www.calstatela.edu/faculty/sfischo/media3.html

[HDY99] Han J, G. Dong, and Y. Yin. “Efficient mining of partial periodic patterns in time series database”. In *Proc. 1999 Int. Conf. Data Engineering (ICDE’99),* pages 106–115, Sydney, Australia, April 1999.

[KHC97] Kamber M, J. Han, and J. Y. Chiang. “Metarule-guided mining of multidimensional association rules using data cubes”. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD’97)*, pages 207–210, Newport Beach, CA, Aug. 1997.

[MTV97] Mannila H, H Toivonen, and A. I. Verkamo. “Discovery of frequent episodes in event sequences”. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[PH02] Pei J, J. Han. “Constraints in data mining: constrained frequent pattern mining: a pattern-growth view”. June 2002. ACM SIGKDD Explorations Newsletter, Volumn 4 Issue 1.

[PTH01] Pei J, A. K. H. Tung, and J. Han, “Fault-tolerant frequent pattern mining”, Proc. 2001 ACM-SIGMOD Int. Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'01), Santa Barbara, CA, May 2001.

[P02] Pei J. “Pattern growth method for frequent pattern mining”, thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the School of Computing Science Simon Fraser University.

[SBMU98]Silverstein C, S. Brin, R. Motwani, and J. Ullman. “Scalable techniques for mining causal structures”. In Proc. 1998 Int. Conf. Very Large Data Bases (VLDB’98), pages 594–605, New York, NY, Aug. 1998.

APPEDIX A

GlOSSARY

Data Mining (DM): Find hidden patterns in the database.

Frequent Pattern Mining (FPM): A mining technique to find the complete set of frequent patterns in a given transaction database with respect to a given support threshold.

Fault-tolerant Frequent Pattern Mining (FT-FPM): A mining technique that allows certain number of mismatch items in frequent pattern mining in order to achieve higher support and discover longer, more general, frequent patterns.

Fault-tolerant Frequent Pattern Mining with Classifiers (CFT-FPM): A mining technique that mines fault-tolerant frequent patterns based on one or a set of classifier(s).

APPEDIX B

The Generation Program: gen.java

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Author: Bei Xu
// Date: 5/01/2006
// Program purpose: Generate suitable dataset “retail1.txt” for CFT-FPM from dataset
//“retail.txt” which is suitable for FT-FPM by inserting a random variable ‘age’ at the
//beginning of each record.  ‘age’ is generated by uniform distribution among 16-80.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//import java libraries
import java.io.*;
import java.util.*;

public class gen{
    public static void main(String args[])throws IOException
    {
        BufferedReader in = new BufferedReader(new FileReader("retail.txt"));

        File outFile = new File("retail1.txt");
        FileOutputStream fos = new FileOutputStream(outFile);
        PrintWriter out = new PrintWriter(fos);

        int age=0;
        Random rand=new Random();   //set random seeds

        //pre: each record is fit for FT-FPM
        //post: generate age, combine with FT-FPM record, each record is fit for CFT-FPM
        for (String line = in.readLine(); line != null; line = in.readLine())
        {
            ArrayList person=new ArrayList();

            //generate random value for age by uniform distribution
            age =16 + rand.nextInt(64);  //age is between 16-80

            person.add(String.valueOf(age));

            //pre: age and FT-FPM are separated
```

```
            //post: age combines each FT-FPM record and become CFT-FPM record.
            for (StringTokenizer ST = new StringTokenizer(line);ST.hasMoreTokens();)
            {
                  Integer item = new Integer(ST.nextToken());
                  person.add(item);
            }
            //pre: made the CFT-FPM records
            //post: output CFT-FPM records.
             for (int i=0;i<person.size();i++)
            {
                  out.print(person.get(i)+" ");
             }
             out.println();
             out.flush();   //flush the buffer
          }

         out.close();     //close file out pointer
         in.close();      //close file in pointer
      }//end of main()
} //end of gen.java
```

APPEDIX C

CFT program: CFTmine.java

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Author: Bei Xu
//Date: 5/01/2006
//Program purpose: This program demonstrates the fault-tolerant frequent pattern mining
//with classifier.  The dataset is retail1.txt, which contains 10,000 sales records from a
//store's sales information. This program outputs all the CFT-FP's found, execution time
//and IO time.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import java.util.*;
import java.io.*;

public class CFTmine{
        public static void main(String[] args) throws IOException
        {    //initialize support variables
                final int MIN_ITEM_COUNT = 99;          //min_sup^item
                final int MIN_PATTERN_COUNT = 100;   //min_sup^FT
                final int MIN_PATTERN_LENGTH =5;     //min_length
                final int [] MAX_MISSING_ITEMS= {2,1,0};   //fault-tolerance

            //create three empty categories to hold each items
                TreeMap [] items =new TreeMap[3];   //three categoires
                for (int eachAge=0;eachAge<3;eachAge++)
                {
                  items[eachAge]=new TreeMap();
                }

             //create three empty categories to hold each record
                ArrayList [] peopleAge=new ArrayList[3];   //three categories
                for (int eachAge=0;eachAge<3;eachAge++)
                {
                  peopleAge[eachAge]=new ArrayList(0);
                }

               int totalPatterns=0;   //calculate total CFT-patterns discovered
```

```
        long totalTime=0;     //calculate total run time
        int IOcount=0;        //calculate total IO operations

    //fill people's lists, read in items from input file.
         BufferedReader in = new BufferedReader(new FileReader(args[0]));
    //input each record as a person
         for(String line = in.readLine(); line != null; line = in.readLine())
         {   //each person is a record in the file
                  HashSet person = new HashSet();

                  StringTokenizer ST = new StringTokenizer(line);
                  IOcount=IOcount+1;
          //input first column in the file, which is, the age value.
                 Integer age =new Integer(ST.nextToken());
          // input the rest data in the record to person.
                  for(;ST.hasMoreTokens(); )
                  {
                           Integer item = new Integer(ST.nextToken());

                  //Assume that each people cannot have same item multiple times?
                           if(person.contains(item)) continue;
                  //If it is not a duplicated item, add it to the person
                         person.add(item);

                          // if age is under 30
                          if (age.intValue()<30)
                          {    //count occurences of this item
                                   Integer oldCount = (Integer)items[0].get(item);
                                   if(oldCount == null)
                                     items[0].put(item, new Integer(1));
                                   else
                                     items[0].put(item, new
                                                 Integer(oldCount.intValue() + 1));
                           }
                          //if age is between 30 and 50 inclusive
                         else if ((age.intValue()>=30) && (age.intValue()<=50))
                          {
                                   Integer oldCount = (Integer)items[1].get(item);
                                   if(oldCount = = null)
                                       items[1].put(item, new Integer(1));
                                   else
                                       items[1].put(item, new
                                                    Integer(oldCount.intValue() + 1));
                           }
                          else //if (age.intValue()>50)
                          {
```

```
                                    Integer oldCount = (Integer)items[2].get(item);
                                    if(oldCount == null)
                                       items[2].put(item, new Integer(1));
                                    else
                                       items[2].put(item, new
                                                Integer(oldCount.intValue() + 1));
                                }//end of if/else
                            }//end of for loop
                        //add each person to the proper category
                         if (age.intValue()<30)
                              peopleAge[0].add(person);
                         else
                            if((age.intValue()>=30) && (age.intValue()<=50))
                                 peopleAge[1].add(person);
                            else
                                 peopleAge[2].add(person);
                    }//end of each input record

                    in = null;

        //mine FT-patterns in each category
            for(int eachAge =0; eachAge<3; eachAge++)
            {
                    HashSet nxtPop =new HashSet();     //length-k candidates
                    HashSet curPop = new HashSet();    //length-k FT-patterns

                    int numPatterns = 0;
                    int newSize = 1;
                    long time;
                    System.out.println("Group "+ (eachAge+1)+"******************");
                    IOcount=IOcount+1;
             // If number of person in each group is less than min_sup^FT, eliminate the group.
                    if (peopleAge[eachAge].size()<MIN_PATTERN_COUNT)
                    {
                         System.out.println("Sorry, not enough records in group "+
                                        (eachAge+1)+"\n");
                         IOcount=IOcount+1;
                         continue;
                    }

                  //remove scarce items, leave frequent length-1 items only
                    for(Iterator i = items[eachAge].values().iterator(); i.hasNext();)
                    {
                            int count = ((Integer)i.next()).intValue();
                            if(count < MIN_ITEM_COUNT)
                                    i.remove();
```

```
                }

                //generate length-1 item list
                for(Iterator i = items[eachAge].keySet().iterator(); i.hasNext();)
                {
                        ArrayList pattern = new ArrayList(3);
                        pattern.add(i.next());
                        curPop.add(pattern);   //curPop contains length-1 frequent items
                }

                while(!curPop.isEmpty())   //curPop has length-1 frequent pattern
                {
                        newSize++;
                        nxtPop = new HashSet();
                        System.out.print("Creating patterns length " + newSize + ": ");
                        time = System.currentTimeMillis();

                //create length n+1 candidates
loop:           for(Iterator i = curPop.iterator(); i.hasNext();)
                {
                    ArrayList pattern = (ArrayList)i.next();

loop4:              for(Iterator j = items[eachAge].keySet().iterator();;)
                    {
                        Object item = j.next();
                        if(pattern.lastIndexOf(item) >= 0)
                            continue loop;

                        //create new pattern
                        ArrayList newPattern = (ArrayList)pattern.clone();
                        newPattern.add(item);

                        //Are all sublists in current candidates
                        for(ListIterator k = newPattern.listIterator(); k.hasNext();)
                        {
                                item = k.next();
                                k.remove();
                                if(!curPop.contains(newPattern))
                                continue loop4;
                                k.add(item);
                        }

                        nxtPop.add(newPattern);
                }
        }
                        System.out.println("| population =" + nxtPop.size());
```

```
                    IOcount=IOcount+1;

                    curPop = nxtPop;

                    System.out.print("Checking constraints: ");

          //check candidates against support thresholds
loop2:    for (Iterator i = curPop.iterator(); i.hasNext();)
          {
               ArrayList pattern = (ArrayList)i.next();

               //reset item counts
               For (Iterator j = pattern.iterator(); j.hasNext();)
                    items[eachAge].put(j.next(), new Integer(0));

               int bodySize = 0;   //FT-body size
loop3:         for (Iterator j = peopleAge[eachAge].iterator(); j.hasNext(); )
               {
                    HashSet person = (HashSet)j.next();
                    //check if person is in the body of set
                    int misses = 0;
                    for (Iterator k = pattern.iterator(); k.hasNext();)
                    {
                         Object item = k.next();
                         if(!person.contains(item))
                              misses++;
                         if(misses > MAX_MISSING_ITEMS[eachAge])
                              continue loop3;
                    }
               bodySize++;

               //If person is in FT-body, then add to item counts
               int min = Integer.MAX_VALUE;
               for (Iterator k = pattern.iterator(); k.hasNext();)
               {
                    Object item = k.next();
                    int count = ((Integer)items[eachAge].get(item)).intValue();
                    if(person.contains(item))
                         items[eachAge].put(item, new Integer(++count));
                    if(count < min)
                         min = count;
               }

               //check against min_sup^item and min_sup^FT
               if ((bodySize >= MIN_PATTERN_COUNT) &&
                   (min >= MIN_ITEM_COUNT))
```

```
                continue loop2;
}

        //if min_sup^FT or min_sup^item count is not met, item is removed
        i.remove();
}
                //output length-n patterns
                System.out.println("| population =" + curPop.size());
                IOcount=IOcount+1;

                if(curPop.isEmpty()) break;
                //check against min_length
                if(newSize >= MIN_PATTERN_LENGTH)
                {
                        //print results
                        for(Iterator i = curPop.iterator(); i.hasNext();)
                        {
                                ArrayList pattern = (ArrayList)i.next();
                                //print in sorted order
                                System.out.println(new TreeSet(pattern));
                                IOcount=IOcount+1;
                        }
                        numPatterns += curPop.size();
                }

                time = System.currentTimeMillis()- time;  //calculate system time
                System.out.println("time: " + time + "ms");
                IOcount=IOcount+1;

                totalTime = totalTime + time;

                System.out.println();
                IOcount=IOcount+1;

        }

        System.out.println("# of pattern found for group "+ (eachAge+1) + " are: "
                        + numPatterns+"\n");
        totalPatterns = totalPatterns + numPatterns;
        IOcount=IOcount+1;

  }//end of eachAge (for loop)

  //output statistics
  System.out.println("total number of patterns found for all groups is:
                "+totalPatterns);
```

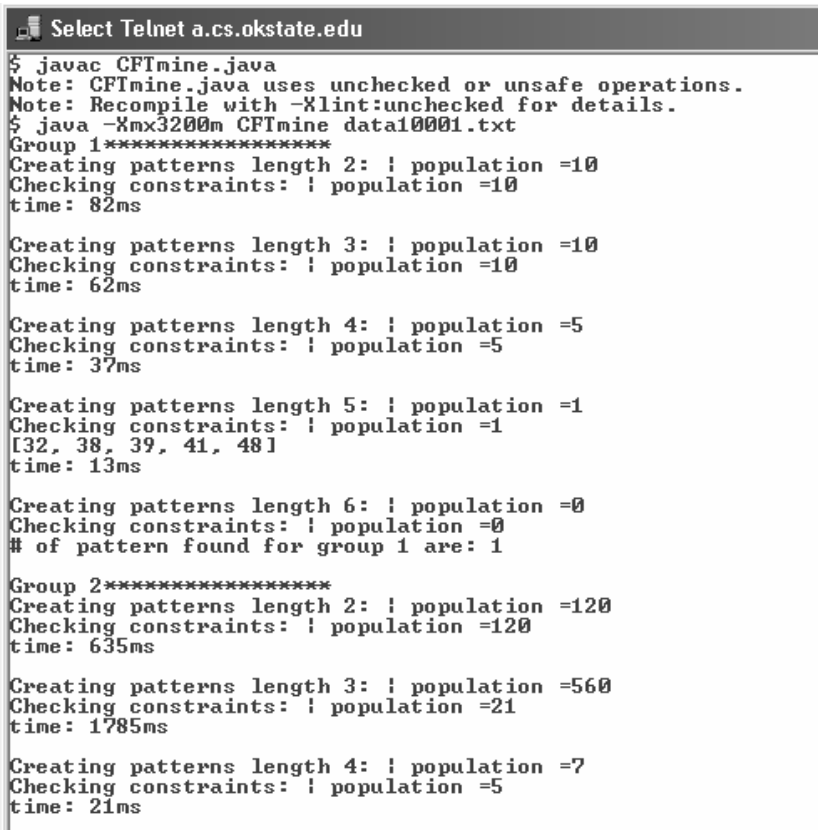
```
            IOcount=IOcount+1;

            System.out.println("total time(miliseconds) is: "+totalTime);
            IOcount=IOcount+2;
            System.out.println("total number of IO operaions is: "+IOcount);
          } //end of main()
} //end of the program
```

APPEDIX D

CFTmine.java users' manual

1. Log on to Oklahoma State University Computer Science department CSA server (a.cs.okstate.edu) with valid userid and password.
2. To compile, at command line, issue:
   javac CFTmine.java
3. To execute, at command line, issue:
   java –Xmx3200m CFTmine data10001.txt
   (note, -Xmx3200m means to allocate 3200 mb memory to this program, data10001.txt contains the first 10000 records of retail.txt)
4. following is the partial execution result.



Following is an explanation of the above result:

Group 1 //category 1
Creating pattern length 2 : | population =10 //generate 10 length-2 candidates
Checking constraints: | population 10 //all 10 candidates are fault-tolerant frequent Patterns
Time: 82ms //the above process takes 82 milliseconds.
…….
Number of pattern found in group 1 is :1 //only 1 fault-tolerant frequent pattern found for group 1

The original program contains support thresholds of:
min_sup^{item} = 99, min_sup^{FT} = 100, min_length=5, fault-tolerance list = {2,1,0}.

To change the values of support threshold, please modify line 6,7, 8, 9 (excluding the blank lines) in CFTmine.java.

Line 6,7, 8, 9 (excluding the blank lines) in CFTmine.java:

final int MIN_ITEM_COUNT = 99; //min_sup^{item}
final int MIN_PATTERN_COUNT = 100; //min_sup^{FT}
final int MIN_PATTERN_LENGTH =5; //min_length
final int [] MAX_MISSING_ITEMS= {2,1,0}; //fault-tolerance

APPEDIX E

CFTmine.java Programmer's manual

Variables:
MIN_ITEM_COUNT: min_sup^{item}
MIN_PATTERN_COUNT: min_sup^{FT}
MIN_PATTERN_LENGTH: min_length
MAX_MISSING_ITEMS: fault-tolerance
items[]: a category contains products' number.
peopleAge[]: a category contains product records (rows)
person: one record (row) of data
item: one product's number in the record
age: each customer's age, which is the first number in each record
oldCount: number of the same item
curPop: contains length-k FT-patterns
nxtPop: contains length-k candidates
IOcount: IO operation counts
time: time to generate and check length-n candidates.
totalTime: the program's total execution time.
numPatterns: number of FT-patterns found for each group
totalPattern : total number of CFT-patterns found for all categories

Data structures:
Treemap: items
Hashset: curPop, nextPop
Arraylist: peopleAge

VITA

BEI XU

Candidate for the Degree of

Master of Science

Thesis: FAULT-TOLERANT FREQUENT PATTERN MINING
WITH CLASSIFIERS

Major Field: Computer Science

Biographical:

Education: Graduated from Colcord High School, Arkansas in May 1998; Graduated from Northeastern State University, Oklahoma in May 2003; Received Bachelor of Science degree in Computer Science; Completed the Requirements for the (degree title) degree at Oklahoma State University, Stillwater, July 2006.

Experience:

Certification: Oracle Certified Professional in Database Administration (9i), Sun Certified Programmer

Software Developer May04-Nov04
Agriculture Department, Oklahoma State University, Stillwater OK

Network Operator/Server Monitor Aug04-May05
Network Operations, Oklahoma State University

Assistant Programmer May03-Aug03
Network Programming Team, Northeastern State University, Tahlequah, OK

Name: Bei Xu Date of Degree: July, 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: FAULT-TOLERANT FREQUENT PATTERN MINING WITH CLASSIFIERS

Pages in Study: 58 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: This paper proposes to mine fault-tolerant patterns with classifiers (CFT-FPM). With CFT-FPM: 1) one or more classifiers is (are) picked to be fixed at a set of specific values or specific ranges; 2) then, FT-FPM is used to mine patterns based on the corresponding fault-tolerance. Since the result is driven from classified data and proper fault-tolerance, patterns that are closer to reality are discovered.

Findings and Conclusions: Compares to FT-FPM, CFT-FPM finds fault-tolerant patterns based on classifiers. Thus, patterns closer to reality are found since classifiers are added. In CFT-FPM, since the dataset is classified into n smaller groups that contain fewer records compared to the entire dataset, support thresholds are harder to be reached. Thus, compared to FT-FPM, CFT-FPM finds fewer patterns. In CFT-FPM, since the entire dataset is classified into n smaller groups, in each group, less length-k itemsets are generated and checked, thus, less length-(k+1) are generated and checked, and so on. Thus, compared to FT-Apriori, CFT-Apriori is much faster. CFT-Apriori is based on FT-Apriori and FT-Apriori is known to be very memory-consuming and time-consuming over large number of records, CFT-FPM is also very memory-consuming and time-consuming.

Advisor's Approval _____Dr. George E Hedrick_____