FTXI: Fault Tolerance XCS in Integer

Hong-Wei Chen
Ying-Ping Chen

NCLab Report No. NCL-TR-2006003
January 2006

Natural Computing Laboratory (NCLab)
Department of Computer Science
National Chiao Tung University
329 Engineering Building C
1001 Ta Hsueh Road
HsinChu City 300, TAIWAN
`http://nclab.tw/`

FTXI: Fault Tolerance XCS in Integer

Hong-Wei Chen and Ying-Ping Chen
Department of Computer Science
National Chiao Tung University
HsinChu City 300, Taiwan
{hweichen, ypchen}@cs.nctu.edu.tw

January 31, 2006

Abstract

In the realm of data mining, several key issues exists in the traditional classification algorithms, such as low readability, large rule number, and low accuracy with information losing. In this paper, we propose a new classification methodology, called fault tolerance XCS in integer (FTXI), by extending XCS to handle conditions in integers and integrating the mechanism of fault tolerance in the context of data mining into the framework of XCS. We also design and generate appropriate artificial data sets for examine and verify the proposed method. Using the real world data as well, our experiments indicate that FTXI can provide the least rule number, obtain high prediction accuracy, and offer rule readability, compared to C4.5 and XCS in integer without fault tolerance.

1 Introduction

No matter in what business, there are some history records. For example, a bank has a lot of credit card transactions, and a supermarket has many shopping records. We believe there is certain information in these history records. If we can get this information, we will be able to design better strategy to get higher profit. For example, one Midwest grocery chain find that when men bought diapers on Thursdays and Saturdays, they also tended to buy beer. The grocery chain move the beer display closer to the diaper display. Then, beer and diapers were sold at full price on Thursdays to increase revenue. Some relative algorithms were proposed, and we call them methods of data mining [1]. Data mining is composed of many components. The most famous are associative mining [2] [3], classification, and cluster.

Classification is a kind of the supervised method and is commonly used by industry. In supervised analysis, the initial step is to assemble a subset of samples, called the training or learning set, which were previously diagnosed by an external supervisor. Then we use the training set to build the prediction model. Finally, we use the model to predict the class level of the unclassified data. The classic classification algorithms include C4.5, ID3, Artificial Neural Network(ANN) [4], Learning Classifier System(LCS) [5] and XCS [6]. The above algorithms work successfully under their respective hypotheses.

Although the above algorithms have high accuracy, they still have problems or limitations, such as that some data patterns can not be recognized and that the number of generated rules is too large. If the rule number is too large, the classification model will be useless in company decision making. For example, we can not design a strategy with one thousand different rules contain the same class level, but it is easy when we just have ten rules. Therefore, the above problems will result in losing the important information and in lowering readability. In order to handle this situation, we want to propose a strategy named *Fault Tolerance XCS in Integer* (FTXI), which can provide the least rule number, get high accuracy, and achieve high readability. Particularly, we extend XCS into the integer domain. Then we modify XCS in integer to permit fault tolerance.

In the remainder of this paper, we will discuss in detail about the problems of traditional classification algorithms and how to handle those problems with the mechanism of fault tolerance in section 2. We will briefly review XCS in section 3. Then, the proposed modification to integrate the mechanism of fault tolerance into XCS will be described in detail in section 4, followed by displaying our experimental results in section 5. Finally, we conclude the present work in section 6.

2 Fault Tolerance in Data Mining

In this section, we will first discuss problems of traditional algorithms. Then, we will introduce the idea of fault tolerance in the context of data mining. Finally, we we tell about how to use the technique of fault tolerance to handle problems of traditional data mining algorithms.

2.1 Problem Description

Traditional classification algorithms can be divided into three categories. In the first category, the classification model is a system which contains no rules, and artificial neural networks (ANN) belongs to this category. Although ANN has a high prediction accuracy, ANN can not tell the users why such classification decisions are made.

In the second category, the classification model works successfully only for linearly separable data, such as C4.5 and ID3. Such a model assumes that the data is linearly separable, so it will fail when the data is not linearly separable. In addition, this model will result in a large rule number. Take Tables 1 and 2 as example, Table 1 shows each attribute's possible values. A, B, C, and D are attributes, and Class level is the attribute which we want to classify. Integer between one to five is the attribute for possible catalog values. Zero stands for "don't care". Table 2 shows some rule examples of the linearly separable algorithms. The first row of Table 2 means {(A=1) and (B=2) and (C=3) and (D=4) $\rightarrow$ (Class level=5)}, and the other rows can be interpreted in the same way. The example uses four rules for class level five, but the above four examples have only one attribute different between each other. There are many similar examples and they result in a rule number which is too large for a practical purpose.

In the last category, the classification tool ignores some noise information to get better accuracy. Learning Classifier Systems (LCS) and XCS belong to this category. Take Table 2 as example, the current rule set have rules as shown in Table 2, but the four rules will be discarded if they are rarely triggered. As a consequence, we lose these rules in the system.

From the analysis, we can observe that the traditional data mining algorithms suffer from huge rule numbers, lower rule readability, and information losing when the model ignores noise by design.

2.2 Fault Tolerance

In 2001, Han et al [7] indicated that the real world tends to be diverse and dirty. There are few non-trivial rules with high support and confidence in real data sets. *Support* is the times

Attribute	Possible values
A	0,1,2,3,4,5
B	0,1,2,3,4,5
C	0,1,2,3,4,5
D	0,1,2,3,4,5
Class level	1,2,3,4,5

Table 1: Attributes of rule

Attribute	A	B	C	D	Class level
rule1	1	2	3	4	5
rule2	2	2	3	4	5
rule3	1	3	3	4	5
rule4	1	2	3	5	5

Table 2: Example of linearly separable data rules

a rule is triggered. *Confidence* stands for accuracy of a rule. For this reason, we know that knowledge discovery from large real-world data sets needs the mechanism of fault tolerance, as demonstrated in the following examples.

Example 1 : To study students' performance in several courses, we might find the following rules:
R1: good(x,data structure) and good(x,algorithm) and
good(x,AI) and good(x,DBMS) $\rightarrow$ good(x,data mining), where good(x,y): student x get A in course y.
R2: A student who is good in at least three out of four courses: data structure, algorithm, AI, and DBMS is also good at data mining.

R1 stands that the student x get A in data structure, algorithm, AI, and DBMS can imply a student will get A in data mining. Although the rule has a high predict accuracy (high confidence), it cover only a small set of cases (low support). Instead, R2 is more general than R1 because every student satisfies R1 also satisfies R2, but not vice versa. Therefore, R2 gets more support than R1 does. Han et al named that the rule requires data to match only part of its left side as *fault tolerance.* They expect that fault tolerance operation will generate rules with high accuracy and support.

2.3 Problems Solving

Inspired by fault tolerance, we will use such a mechanism to handle the problems of the traditional classification algorithms.

First, we focus on the problems of huge result rule numbers. In Table 2, we find it needs four rules to represent class level 5, and this situation will become worse for the real word data. Now we can use one rule with fault tolerance to represent these rules as shown in the following.
R3: (A=1) and (B=2) and (C=3) and (D=4) $\rightarrow$ (Class level=5) with fault tolerance=1
As aforementioned, we know R3 is more general than the rules listed in Table 2. Each rule in Table 2 can be explained by R3. With fault tolerance, now we can reduce the four rules to only one rule, in other words, we can make the size of the rule set one quarter in this case.

Secondly, we concentrate on low accuracy with information losing. Each rule in Table 2 is rarely triggered, but the summation of the triggering events for all of the rule in the table is significant enough. Therefore, if we use the R3 to replace the rules in Table 2, R3 will be triggered more frequently and will survive during the process of building the classification model because the record of each student satisfies the rule in Table 2 also satisfies R3. With the two examples, we find that we can use fault tolerance to handle these traditional classification problems successfully.

3 Brief Review of XCS

Since we adopt XCS as the underlying data mining methodology for our present work, in this section, we will give a brief review of XCS. We will first present the history and the algorithm of XCS. Then, we discuss the properties of XCS, followed by the discussion on why we choose XCS as the base classification tool.

3.1 History of XCS

Learning Classifier Systems (LCS) are rule-based classifiers, consisting of a set of rules and procedures for performing classifications and discovering rules using genetic operators. LCS was proposed by Holland [5], who laid out a comprehensive framework that included the condition sensor, internal memory, reinforcements, and rule generation by using a genetic algorithm (GA).

Because of LCS's overgeneralization and the difficulty to implement a comprehensive system, Wilson eliminated the internal memory to propose a minimalist version LCS, called Zeroth-Level Classifier System (ZCS) [8] in 1994. ZCS has remarkably good performance with a simple framework. ZCS still has some problems, such as path habits and recombination rules from entirely different niches. Path habits means ZCS prematurely converge onto suboptimal rules before the space can be properly explored. Recombining rules from different niches will result in useless rules. Recognizing these drawbacks, Wilson proposed XCS in 1995 [6]. XCS uses the accuracy and experience to construct its fitness to avoid the path habit problem and applies recombination to the action set for making meaningful rules. With the new mechanisms, XCS is improved in accuracy and system complexity.

3.2 Algorithm of XCS

XCS framework consists of two cycles, the rule evaluation cycle and the rule discovery cycle. The rule evaluation cycle is composed of representation, fitness evaluation, cover operation, and subsumption. The rule discovery cycle runs GA with the information collected from the rule evaluation cycle. In the following paragraphs, we will discuss these components.

3.2.1 Representation

Representation of XCS contains three parts, condition, action and prediction reward. Condition composed of many attributes is used to model the environment event. The value of an attribute value can be 0, 1, or #. Zero stands for the attribute is not triggered, one means the attribute is triggered, and # represents that the attribute is ignored. Action stands for the decided class level when the condition match the environment event. If we take this action, in other words, decide the class level, and put it to the environment, we will get some rewards from environment, called prediction reward.

3.2.2 Fitness Evaluation

After describing the representation, we will discuss how to determinate the fitness of a rule. We view the prediction reward and experience as a rule's fitness and initialize the value with zero in the beginning. For each environment event, there will be some match rules and they will decide a action, class level, put it to the environment, and get rewards. If the class level XCS decides is equal to its real class level, the reward will increase for the rule in the action set. Instead, if we take a wrong action, the rule's fitness will be decreased.

3.2.3 Cover Operation and Subsumption

There are two important operations in XCS, cover and subsumption. When an environment event can not be matched with XCS's current rule set, the cover operation will generate a new rule to match this event for improving the accuracy. Besides, there may be some rules with the same meaning in XCS's framework. The subsumption operation is an mechanism designed for this situation, and it make a rule absorb other rules if the rule is more general than other rules.

3.2.4 Genetic Algorithm

With information from the rule evaluation cycle, we apply GA to the current rule set. GA usually uses one-point crossover and bitwise mutation on the rule set to generate new rules. In addition, GA will hold and delete rules based on rules' fitness values. In general, XCS will delete rules with low accuracy and hold rules with high accuracy.

3.2.5 Flow of XCS

In the beginning, XCS randomly initializes the rule set with zero reward. The rule evaluation cycle have four steps. First, detectors detect the state of the environment. Secondly, the system examines the condition part of each rule to determine the match set. Thirdly, the match set will be grouped into different sets based on their own actions, and the predict payoff for each action is calculated to determinate the action set. Finally, effectors implement the action in the environment, get the reward, and distribute it to the rules in the action set. After a specified period of time, GA is executed to generate new rules and delete bad rules in the rule discovery cycle. With this framework, Wilson expects they can find the classification rules with high accuracy.

3.3 Properties of XCS

In the previous sections, we talk about XCS's history and algorithm, we will discuss XCS's properties in this section.

- XCS has been shown capable of learning complex, non-linear classification functions that can be used to accurately predict new cases, on a variety of problem domains.
- XCS generalizes over attributes space and under ideal condition discover a maximally general, accurate rule set to perform classification.
- XCS offers the potential for explanatory data analysis in addition to predictive modeling because it is rule based. In real world data mining exercises, being able to explain how a technique forms classification is often as important as accuracy, and the techniques where

Course	value 1	value 2	value 3
DS	good	average	bad
	1	2	3
ALGO	good	average	bad
	1	2	3
AI	good	average	bad
	1	2	3
DBMS	good	average	bad
	1	2	3
Data Mining	good	average	bad
	1	2	3

use 0 stand for don't care condition

Table 3: Attributes of rule

there is difficulty to explain (such as neural networks) are often treated with suspicion in industry.

- Unlike most rule induction algorithms, XCS does not discover and evaluate rules in isolation. Instead, the two components work in parallel.
- XCS's model may change over time for maintaining and updating classification function without the requirement of retraining on all data.

Because of the above reasons, we believe XCS has better potential compared to ANN, C4.5, ID3, and LCS, we choose it for the developing base of our framework.

4 FTXI

Traditional XCS takes {0,1,#} as its input, which cannot accommodate the numeric classification problems. Hence, we will introduce a method that can transfer XCS into the integer domain. Then, we discuss how to modify XCS to permit fault tolerance. Finally, we design a simple strategy to avoid over-fitting. We call the extended version of XCS as the *fault tolerance XCS in integer*, FTXI.

4.1 XCS in Integer

Although XCSI [9] was proposed in 2001, it does not fulfill our need. We design a simple integer XCS for our own purpose. Using the same idea of XCS, for each attribute we use a numeric value standing for its real meaning. Take Tables 3 and 4 as example, for course data structure, we assume it can be divided into three class levels, good, average, and bad. We use 1 for good, 2 for average, 3 for bad, and 0 for don't care. By applying this transformation, we can get rules as in Table 4. The rule1 in Table 4 means if a student gets a good grade in data structure, an average grade in algorithm, a bad grade in AI, and an average grade in DBMS, we can imply that his grade of data mining will be average. With this transformation, we can take integers as the input to our system and handle numeric classification problems.

Attribute	DS	ALGO	AI	DBMS	Data Mining
rule1	1	2	3	2	2
rule2	2	2	3	2	2
rule3	1	3	3	2	2
rule4	1	2	3	1	2

Table 4: Example of XCS in integer

4.2 Fault Tolerance Representation

In the previous section, we tell about how to transfer XCS into the integer domain. Now we will discuss on how to combine XCS with fault tolerance. First, we focus on the representation. We add an attribute, FTN, which records how many attribute data do not necessarily match the rule. Take Table 5 as an example, rule1 means {(A=1) and (B=2) and (C=3) and (D=2)} → (Class level=2) with (FTN=1), in other words, a data item satisfying three attributes of rule1 still implies (Class level=2). So, the environment events {(A=1) and (B=2) and (C=3) and (D=3)}, {(A=2) and (B=2) and (C=3) and (D=2)}, {(A=1) and (B=1) and (C=3) and (D=2)} and {(A=1) and (B=2) and (C=1) and (D=2)} still match the fault tolerance rule. In the traditional methods, we must use four rules to represent the four patterns, but now we can use one rule to represent the four rules and use the sum of the four rules fitness as the fault tolerance rule's fitness. If the rule is rule2 in Table 5, the environment event must completely match the rule because the FTN is zero.

4.3 Fault Tolerance Operation

We define an operation called fault tolerance to achieve the effect of fault tolerance. The fault tolerance operation will change FTN of a rule to increase the fault capacity. Because applying the fault tolerance operation to the rules with high fitness or zero fitness is meaningless, we put all the rules which contain the non-zero fitness into a set, called NZS. Then, we divide NZS into (1) top seventy percentage; and (2) the other thirty percentage. We apply the fault tolerance operation to the bottom thirty percentage only. By do so, we can make the rules with low fitness more general and active.

4.4 Result Rule Clearing

The main objectives of FTXI is to provide the least classification rules as well as to offer high accuracy. Therefore, we will delete the rules of which the experience is equal one to avoid over-fitting in the final rule set. So, we will have a lower accuracy in the training set, but get a high accuracy in the test set. With this strategy, we can achieve our design goal.

After transferring XCS into integer, applying fault tolerance operation, and deleting over-fitting rules, the Fault Tolerance XCS in Integer, FTXI, is then ready for action. We expect FTXI can provide the least rule number and offer high accuracy and readability.

5 Experimental Results

Our experiments is composed of two parts. One part is to use the artificial data that is designed for verifying the effectiveness of the mechanism of fault tolerance in classification. The artificial data sets contain three types, perfect samples, fault tolerance samples, and error fault tolerance

Attribute	A	B	C	D	FTN	Class level
rule1	1	2	3	2	1	2
rule2	2	2	3	2	0	2
rule3	1	3	3	2	0	2
rule4	1	2	3	1	1	2

Table 5: Example of FTXI

samples. The other part is the breast-cancer-wisconsin data set imported from the UCI Machine Learning Repository.

Each of the four data sets is divided into the training set and the test set. We use the training set to build the prediction model and the test set to test the model accuracy. Moreover, the four data sets are also tested with C4.5, Integer XCS, and FTXI in ten independent runs. We will first describe how to make meaningful artificial data by using our data generator. Then, the experimental results on the artificial data will be presented. Finally, we will introduce the real word data set and the experimental results.

5.1 Data Generator

Our data generator uses these symbols:

- n: integer value, seed rules number
- m: integer value, attribute number
- N: integer value, samples number
- *TN1*: integer value, training samples number
- *TN2*: integer value, test samples number
- FT: integer value, mismatch attribute number of data
- σ: integer value, the number of attributes will be change
- ρ: float value, the percentage of samples be change
- AR: array of integer, each integer represents the possible catalog of its corresponding attribute
- λ: boolean value, zero means attributes change will result in changing of class level and one mean class level will not change no matter attributes change or not.

5.1.1 Description of Perfect Sample

Perfect samples mean there is no noise in the samples. We randomly initialize n seed rules with m attributes and duplicate the n seed rules until the total number of sample is N. Then the sample will be divided into two parts, the training samples and the test samples. We use the training samples to build the predict model and use the test samples to verify the accuracy. In our experiment for the perfect case, we set n=20 and m=10, N=10000, *TN1*=2000, *TN2*=8000, FT=0, σ=0, ρ=0,AR={10,10,10,10,10,10,10,10,10,10} ,and λ=0.

	XCS in Integer	FTXI	C4.5
Rule number	30.9	27.5	71
Recognition rate	100%	100%	100%
No recognition rate	0%	0%	0%
Accuracy	100%	100%	100%

Table 6: Results of Perfect Sample in Training Set

5.1.2 Description of Fault Tolerance Data

Fault tolerance data mean that these data contain the same class level even if there is little difference in their attributes. In addition to the procedure of perfect sampling, we use ρ to decide what the percentage of data will become fault tolerance data and σ to decide how much attributes will be altered. Finally, we set λ=0 to set that class level will not be changed even if there are different attributes. In our experiment for the fault tolerance case, we set n=20 and m=10, N=10000, *TN1*=2000, *TN2*=8000, *FT*=1, σ=1, ρ=0.5,*AR*={10,10,10,10,10,10, 10,10,10,10}, and λ=0.

5.1.3 Description of Error Fault Tolerance Data

Error fault tolerance data mean these data contain the different class level if there is little difference in their attributes. In addition to the procedure of perfect sampling, we use ρ to decide what the percentage of data will become fault tolerance data and σ to decide how much attributes will be altered. Finally, we set λ=1 to set that class level will be changed even if there are different attributes. In our experiment for the error fault tolerance case, we set n=20 and m=10, N=10000, *TN1*=2000, *TN2*=8000, *FT*=1, σ=1, ρ=0.5,*AR*={10,10,10, 10,10,10,10,10,10,10} ,and λ=1.

5.2 Parameters of XCS and Estimation Principle

We follow [10] to set the parameters for XCS: N=1000, α=0.1, ν=5, β=0.2, χ=0.7 ,μ=0.05 ,θsub=5, θdel=5, θGA=40, doGASubsumption=true, and doActionSetSubsumption=true. We initialize fitness and error with small values.

In the following experimental results, we use "Rule number" to represent how many rules the algorithm needs in its model, "Recognition rate" to stand for the number of successfully recognized data divided by the number of data which can be recognized, "No recognition rate" to represent the number of patterns the algorithm can not recognize divided by the number of all patterns, and "Accuracy" to be the number of successfully recognized data divided by the number of all patterns.

5.3 Results of Perfect Sample

5.3.1 Experimental Results

Observing Table 6, we find all the algorithms in the expreiment completely handle the training samples because of the accuracy of each algorithm is 100%. However, the rule number of each algorithm is quite different. On the average of ten independent runs, FTXI generates 27.5 rules, XCS in integer 30.9 generates rules, and C4.5 generates 71 rules. The experimental results demonstrate that FTXI work very well for the perfect samples.

	XCS in Integer	FTXI	C4.5
Recognition rate	100%	100%	100%
No recognition rate	0%	0%	0%
Accuracy	100%	100%	100%

Table 7: Results of Perfect Sample in Test Set

	XCS in Integer	FTXI	C4.5
Rule Number	35.5	30.6	961
Recognition Rate	99.22%	100%	99.56%
No Recognition Rate	34.06%	0%	0%
Accuracy	65.43%	100%	99.56%

Table 8: Results of Fault Tolerance Sample in Training Set

5.3.2 Discussion

Since our data generator will generate linearly separable data. In this case, even if the data is linearly separable, C4.5 cannot provide the least classification rule number for the training data.

C4.5 is constructed in a top-down recursive divide-and-conquer manner. At start, all the training examples are at root. Then examples are partitioned recursively based on the selected attributes and a test attribute is selected on the basis of the information gain. Although the information gain can decide the dividing attribute that makes the system entropy the least, it cannot guarantee that such a choice is the best dividing attribute. Therefore, C4.5 has to use 71 rules in this experiment of the perfect samples, although the ideal size of the rule set is 20.

XCS in integer and FTXI both use approximate 30 rules to explain the training samples because there are some rules with the same meaning which may subsume each other.

5.4 Results of Fault Tolerance Data

5.4.1 Experimental Results

In Table 8, we find C4.5 and FTXI can represent the training set completely, but XCS in integer cannot. XCS in integer can only represent 65.43% of the train set in 50% fault tolerance data. Comparing the rule number of FTXI and that of C4.5, we can find that the rule number generated by of FTXI is 3% of that generated by C4.5, in other words, we save 97% of the rule number for C4.5. To analyze of the test data, in Table 9, we can find that the rules generated for the training set can provide a very good estimation for the test set because the recognition rate, no recognition rate, and accuracy are all similar to those in Table 8.

	XCS in Integer	FTXI	C4.5
Recognition rate	99.93%	100%	100%
No recognition rate	34.74%	0%	0%
Accuracy	65.22%	100%	100%

Table 9: Results of Fault Tolerance Sample in Test Set

	XCS in Integer	FTXI	C4.5
Rule number	30.9	28.6	2371
Recognition rate	98.38%	94.84%	73.3%
No recognition rate	56.75%	42.98%	0%
Accuracy	42.55%	54.07%	73.3%

Table 10: Results of Error Fault Tolerance Sample in Training Set

	XCS in Integer	FTXI	C4.5
Recognition rate	98.58%	95.38%	57.6%
No recognition rate	57.46%	43.96%	0%
Accuracy	41.95%	54.80%	57.6%

Table 11: Results of Error Fault Tolerance Sample in Test Set

5.4.2 Discussion

In the beginning, we focus on the rule numbers of FTXI and C4.5. Reviewing Tables 5 and 4, C4.5 generates the rules like those in Table 4, and a large result rule number is inevitable. However, FTXI uses one rule like the first row of Table 5 to represent other rules. That is the reason why FTXI only uses 30.6 rules, but C4.5 needs 961 rules.

Then, we discuss why XCS in integer contains 34.06% samples which can not be recognized. In the XCS framework, XCS uses GA to preserve the rules with high fitness and delete the rules with low fitness. Fitness is composed of experience and accuracy. Experience stands for the times of the rule is triggered. Therefore, the rules generated by the cover operation will be deleted because of their low fitness caused by the rare triggering events. Hence, XCS cannot recognize those data and such information is lost.

5.5 Results of Error Fault Tolerance Data

5.5.1 Experimental Results

Table 10 shows that C4.5 offers the best representation of the training set, FTXI provides 54.07%, and XCS in integer has 42.55%. Although C4.5 offers high representation of the training samples, it has only 57.6% accuracy of the test samples. Comparing the accuracy of FTXI and C4.5 in Table 11, the results are of only slight difference. In addition, C4.5 uses 2371 rules in the prediction model, but FTXI uses only 28.6 rules. In the viewpoint of readability, FTXI has better performance and generality.

Then, we find XCS in integer has better recognition rate than FTXI with an approximately same rule number, and C4.5 has the worst recognition rate.

5.5.2 Discussion

Comparing C4.5's difference of accuracy between training and test sample results, we can conclude that C4.5 probably builds an over-fitting classification model.

Moreover, we find the recognition rate of FTXI is lower than that of XCS in integer. Because there are some data that cause the failure of the fault tolerance operation. Take Table 12 as an example, if we have a rule with accuracy and fitness like
R4: (A=1) and (B=2) and (C=3) $\rightarrow$ (Class level=1) (fault tolerance=1),
each row of Table 12 will decrease the rule fitness and accuracy because of the error action

Data	Attribute1	Attribute2	Attribute3	Class Level
1	1	2	3	3
2	2	2	3	2
3	1	3	3	3
4	1	2	2	4

Table 12: Error Fault Tolerance Data

decision. Despite this, FTXI's accuracy is higher than that of XCS in integer, and FTXI's recognition rate is still higher compared to C4.5. Because C4.5 will assign every pattern a class level no matter whether it collects enough information, but XCS in integer and FTXI will distinguish recognition patterns from non-recognition patterns.

5.6 Description of Real Word Data

We used the Wisconsin Breast Cancer Database which was donated to the UCI Repository [11] by Prof. Olvi Mangasarian and contains 699 instances collected over time by Dr. William H. Wolberg. Each instance contains nine attributes which are Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, and Mitoses. Each attribute has a value between 1 and 10 inclusive. Small sample of raw data is shown as follow:

```
1166654,4,10,3,5,1,10,5,3,10,2 1167439,4,2,3,4,4,2,5,2,5,1
1167471,2,4,1,2,1,2,1,3,1,1 1168359,4,8,2,3,1,6,3,7,1,1
1168736,4,10,10,10,10,10,1,8,8,8 1169049,4,7,3,4,4,3,3,3,2,7
1170419,4,10,10,10,8,2,10,4,1,1 1170420,4,1,6,8,10,8,10,5,7,1
1171710,2,1,1,1,1,2,1,2,3,1 1171710,4,6,5,4,4,3,9,7,8,3
1171795,2,1,3,1,2,2,2,5,3,2 1171845,4,8,6,4,3,5,9,3,1,1
1172152,4,10,3,3,10,2,10,7,3,3 1173216,4,10,10,10,3,10,8,8,1,1
```

The first number is a label, the next nine attributes are the attributes, and the last is the class level.

5.7 Results of Real World Data

5.7.1 Experimental Results

Table 13 shows that C4.5 has the highest representation of the training data, but it needs 210 rules. Besides, we can find the best is XCS in integer, then FTXI, and the last is C4.5 in the recognition rate. The discussion given in the previous section can also explain such a situation. In the viewpoint of the rule number, we find that FTXI uses a half of rules generated by XCS in integer and a quarter of rules generated by C4.5. Observing Table 14, we find the over-fitting in C4.5 still occurs, and its accuracy is 83%. No matter in Tables 13 or 14, we can see that FTXI has better performance in no recognition rate, accuracy, and the number of rules.

5.7.2 Discussion

As the experimental results we obtained on three artificial data sets, we once more have the similar situations and outcomes for the real world data set. Therefore, the observations, expla-

	XCS in Integer	FTXI	C4.5
Rule number	99.8	46.3	210
Recognition rate	99.55%	98.60%	92%
No recognition rate	19.64%	15.35%	0%
Accuracy	80%	83.46%	92%

Table 13: Results of Real World Sample in Training Set

	XCS in Integer	FTXI	C4.5
Recognition rate	98.68%	96.82%	83%
No recognition rate	24%	14.85%	0%
Accuracy	75%	82.3%	83%

Table 14: Results of Real World Sample in Test Set

Run	1	2	3	4	5	6	7	8	9	10
XCS in Integer										
Recognition number	149	148	150	148	150	164	148	148	148	148
No Recognition number	48	50	46	51	46	32	48	47	47	49
FTXI										
Recognition number	164	164	164	164	164	164	170	164	164	164
No Recognition number	30	29	32	30	32	29	25	30	28	32
C4.5										
Recognition number	166	166	166	166	166	166	166	166	166	166
No Recognition number	0	0	0	0	0	0	0	0	0	0

Table 15: Results of Real World Sample in Test Set

nations, and interpretations for the experimental results are fundamentally identical to those presented in the previous sections. Because of the very similar results, our design to integrate the mechanism of fault tolerance into XCS is useful and effective in the real world. Currently, in Table 15, we can find that recognition number of XCS in integer is around 148 and recognition number of FTXI is around 164. So, we can observe that fault tolerance=1 can make FTXI recognize 16 more patterns out of the 200 training samples in this case.

6 Conclusions

In this paper, we first described several problems existing in the traditional classification algorithms, such as low readability, large rule number, and low accuracy with information losing. Then, we reviewed algorithm of XCS. Finally, we proposed FTXI modified from XCS to handle problems of traditional classification algorithms. With experimental results, we find that FTXI solve successfully these problems.

FTXI can use the least rule number and get high accuracy and readability. It will provide better and interesting strategies in commerce for higher profit. Besides, it also helps biologist understand the mystery of human body. But there still are some problems in FTXI, such as error fault tolerance data which lower the recognition rate, some rules with the same meaning that cannot subsume each other, and automation on fault tolerance number. How to cluster the rule set into smaller set is still an interesting work. How to apply FTXI to association rule mining is a good question waiting to be answered.

Research along this line should be continuously pursued and conducted in order to develop not only feasible but also practical classification systems to further advance the business, science, and even art in the world.

Acknowledgments

The work was partially sponsored by the National Science Council of Taiwan under grant NSC-94-2213-E-009-120. The authors are grateful to the National Center for High-performance Computing for computer time and facilities.

References

[1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001.

[2] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining Association Rules between Sets of Items in Large Databases," in *ACM SIGMOD*, 1993.

[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *Very Large Data Bases(VLDB)*, 1994.

[4] J. M. Zurada, *Introduction to Artificial Neural Systems*, 1992.

[5] J. H. Holland, "Adaptation," in *Progress in Throretical Biology*, vol. 4, 1976.

[6] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 2(2), pp. 149–175, 1995.

[7] J. Pei, A. K. H. Tung, and J. Han, "Fault-Tolerant Frequent Pattern Mining: Problems and Challenges," in *ACM-SIGMOD*, 2001.

[8] S. W. Wilson, "ZCS:A zeroth level classifier system," *Evolutionary Computation*, vol. 2(1), pp. 1–18, 1994.

[9] ——, "Mining Oblique Data with XCS," in *Advances in Learning Classifier Systems*, 2001.

[10] M. V. Butz and S. W. Wilson, "An Algorithm Description of XCSs," in *Advances in Learning Classifier Systems*, 2001.

[11] C. Blake and C. Merz, "Uci repository of machine learning databases," 1998, http://www.ics.uci.edu/ mlearn/MLRepository.html.