JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 573-594 (2006)

A Classification Tree Based on Discriminant Functions*

BEEN-CHIAN CHIEN, JUNG-YI LIN[1] AND WEI-PANG YANG[1,2]
Department of Computer Science and Information Engineering
National University of Tainan
Tainan, 700 Taiwan
[1]*Department of Computer and Information Science*
National Chiao Tung University
Hsinchu, 300 Taiwan
[2]*Department of Information Management*
National Dong Hwa University
Hualien, 974 Taiwan

The classification problem is an important topic in knowledge discovery and machine learning. Traditional classification tree methods and their improvements have been discussed widely. This work proposes a new approach to construct decision trees based on discriminant functions which are learned using genetic programming. A discriminant function is a mathematical function for classifying data into a specific class. To learn discriminant functions effectively and efficiently, a distance-based fitness function for genetic programming is designed. After the set of discriminant functions for all classes is generated, a classifier is created as a binary decision tree with the *Z*-value measure to resolve the problem of ambiguity among discriminant functions. Several popular datasets from the UCI Repository were selected to illustrate the effectiveness of the proposed classifiers by comparing with previous methods. The results show that the proposed classification tree demonstrates high accuracy on the selected datasets.

Keywords: knowledge discovery, machine learning, genetic programming, classification, discriminant function, decision tree, classifier

1. INTRODUCTION

The task of classification is to classify unknown objects into predefined classes based on their observed attributes using a classification model learned from a set of training data. Many applications, such as characters recognition, decision-making and disease diagnosis, can be viewed as extensions of the classification problem [14]. A classifier can be modeled using different structures such as decision graphs, decision trees, neural networks and rules. Reducing the processing time and increasing the classification rate are the two main issues in the classification problem. Many methods for designing classifiers have been proposed, such as the Bayesian classifier, decision trees, distance- based classifiers and neural network classifiers. However, each model has its own advantages and disadvantages. It is not easy for a classifier to be trained

Received February 12, 2004; revised June 18 & November 5, 2004 & February 21 & April 7, 2005; accepted August 17, 2005.
Communicated by Chuen-Tsai Sun.
* Part of this paper was presented at the Sixth International Conference on Knowledge-Based Intelligent Information Engineering Systems,16-18 September, 2002, Podere d'Ombriano, Crema, Italy.

efficiently and classify objects effectively simultaneously. To find faster learning methods for building classifiers with high classification accuracy, some newly developed techniques have been recently applied to the classification problem, such as association rules-based learning [22], support vector machines [12], and evolutionary computation [17, 18]. Related research of the classification problem is reviewed briefly in section 2.

A traditional decision tree is a tree-like structure in which each internal node performs a test on one of the attributes, each branch represents an outcome of the test, and each leaf node denotes a class. This paper proposes a new decision tree in which each internal node is represented by a discriminant function used to determine whether an object belongs to a specific class or not. A discriminant function is a mathematical expression that maps a set of numerical features to a specific range for identifying the class of objects. A classifier using discriminant functions is efficient since mathematical functions can be easily calculated. The classifier is also concise because the number of functions does not exceed the number of predefined classes, as Fig. 1 shows. However, two problems have to be considered when a classifier uses discriminant functions as decision nodes. First, it is a difficult to learn accurate discriminant functions. Second, two ambiguous situations, *conflict* and *reject*, may occur while classifying multi-category data using discriminant functions. The conflict case occurs when an object is recognized by two or more discriminant functions simultaneously. The reject case arises when an object is recognized by no discriminant function. Both situations decrease the classification accuracy.

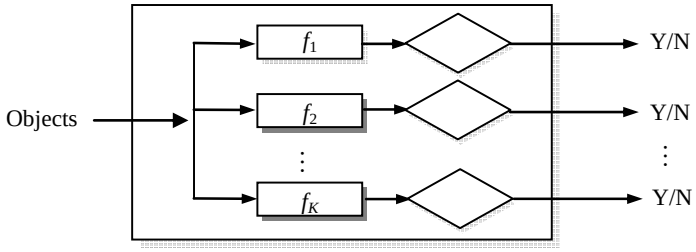

Fig. 1. A classifier using discriminant functions.

Genetic programming (GP) technique was proposed by Koza [17, 18]. GP can discover the underlying data relationships and present them as expressions. In this paper, we apply GP to generate discriminant functions by proposing a distance-based fitness function. Using the distance-based fitness function, discriminant functions can be learned more efficiently and effectively than using the traditional fitness function based on accuracy in genetic programming. After the discriminant functions are generated, the testing results of discriminant functions on the training data are used to construct a binary decision tree based on functions. The classification tree is then used to resolve the conflict cases among the discriminant functions. An object following the root of the classification tree can only be recognized by at most one of the discriminant functions. If an object cannot be recognized by any node of the classification tree, the resolution of *Z*-value measure is proposed for the reject case. Five well-known datasets were selected to show

the effectiveness of the proposed classifier: Fisher's Iris dataset, Wisconsin Breast Cancer dataset, BUPA liver dataset, Vehicle dataset and PIMA dataset [2]. The experimental results are discussed in detail and compared with previous works.

The rest of this paper is organized as follows. Section 2 briefly reviews related work on classification. Section 3 proposes the distance-based fitness function for genetic programming to learn effective discriminant functions. Section 4 gives the proposed algorithms for constructing a decision tree based on discriminant functions and the *Z*-value measure. Section 5 describes and discusses the experimental results. Finally, conclusions are drawn in section 6.

2. RELATED RESEARCH ON CLASSIFICATION PROBLEMS

Many learning approaches are available for constructing classifiers. The Bayesian classifier and the decision tree are well-known and widely used. The Bayesian classifier [13] is based on Bayesian decision theory, which learns the conditional probability of each attribute given a specific class from training data. Classification is accomplished by employing Bayes rule to compute the probability of each class given the attributes of an instance and then assigning to the class with the highest probability. The decision tree is a tree-like structure where each internal node in the decision tree denotes a decision on an attribute. Each branch represents a decision outcome of the decision and leaf nodes represent classes. ID3 [26] and C4.5 [27] are decision-tree classifiers which use an entropy-based measure known as information gain to select the attribute. The selected attribute is the "decision" attribute represented as an internal node that can separate the samples into classes well. Such a partition repeats in each internal node until all the samples in a node belong to the same class or no more attribute remained can be further partitioned. By the learning process of training data, the Bayesian classifier and decision-tree classifiers can partition the domain space into classes. Although the reject case does not exist in the Bayesian classifiers while classifying data, the conflict case may still happen. In decision-tree classifiers, however, the conflict case never occurs, but the reject case may appear [14]. The traditional solution for the above ambiguous cases is majority voting, which labels the data with the most common class in training data.

A neural network [4, 25, 28, 37] is a multi-layered network structure. For an *n*-class classification problem with *m*-dimensional data, a training set is used to train a neural network consisting of *m* input nodes and *n* output nodes. A well-trained network can be regarded as an implicit function with *m* input attributes. An unknown instance then can be classified by assigning it to the class with the maximum output in the network. The drawbacks of the neural network method are that the knowledge representation is unknown and the training process is inefficient.

Classification methods using support vector machines (SVM) [5, 15, 33] have also been frequently discussed. For a training sample (x_i, y_i), let $x_i \in R^n$ be a feature vector and let $y_i \in \{\pm 1\}$ be a class label. SVM tries to find an optimal hyperplane to separate training data if the training data are linearly separable. When the training data are nonlinear, the data are mapped into a higher-dimensional feature space F via a mapping function Φ: $R^n \rightarrow F$. Then the SVM attempts to find a separating hyperplane with maximum margin on F. Different SVM architectures can be obtained through different kernel func-

tions $k(x_i, y_i) = (\Phi(x_i), \Phi(y_i))$. For example, $k(x_i, y_i) = (x \cdot y + 1)^p$ results in a classifier that is a p-degree polynomial.

Evolutionary computational approaches include genetic algorithms (GA) [35] and genetic programming (GP). Generally, a genetic algorithm encodes a set of classification rules as a sequence of bit strings called genes. The evolution operators such as reproduction, crossover and mutation then generate new classification rules with better fitness. After a specified number of generations are computed or the conditions of fitness functions are satisfied, a set of effective classification rules is obtained. For GP-based classifiers, classification rules [1, 11] or classification functions [3, 6, 16, 23] can be learned from the training dataset. Kishore [16] proposed an adaptive learning method for generating mathematical function to classify data and provide an approach called the strength of association measure (SA measure) to overcome the conflict case. The SA measure calculates the ratio of correctly classified examples in the training dataset to be an SA value for each discriminant function. If a conflict occurs, the unknown object is assigned to the class with the highest SA value. However, this approach can lead to a high misclassification rate due to the lower-SA functions being swamped by higher-SA functions. Furthermore, Kishore *et al*. does not solve the problem of the reject case. They simply classify the rejected objects into an extra rejected class [16]. Loveard proposed different fitness functions for genetic programming to learn accurate discriminant functions [23]. However, the training time is too long (several hours) compared with other GP-based methods. Due to all of these drawbacks, the proposed GP method in this paper tries to improve the efficiency of learning discriminant functions by designing a new fitness function, and enhances the classification accuracy by developing a classification tree to overcome the ambiguity among discriminant functions.

3. LEARNING DISCRIMINANT FUNCTIONS

This section introduces the learning method for generating discriminant functions based on genetic programming. First, the basic algorithm of genetic programming is reviewed and a formal description is presented for the classification problem. Then, a distance-based fitness function and the GP-based algorithm are provided for learning discriminant functions.

3.1 Genetic Programming

The technique of genetic programming has been applied to many applications, including symbolic regression, the evolution of robot control programs and the evolution of classification [18]. Genetic programming can discover underlying relationships between data and present them as expressions constructed by possible terminals, operations and functions. The most popular operations include arithmetic operations like addition, subtraction, multiplication and division, and conditional operations like IF and ELSE. Functions may include trigonometric functions like sine and cosine, or user-specific functions.

Genetic programming begins with a set of randomly-created individuals called a population. Each individual is a potential solution represented as a binary tree. Each bi-

nary tree is constructed by all possible compositions of the sets of operations and terminals. A suitable fitness function should be given for evaluating the fitness value of each individual. Then, a set of individuals with better fitness values is selected and used to evolve the next generation's population using the predefined genetic operators. At the end of the evolution, a set of individuals with good fitness is generated and the goal expression can be obtained. The genetic operators used to evolve individuals generally include reproduction, crossover and mutation.

Reproduction, the simplest operator, copies the individuals with better fitness values as the population of the next generation. Thus, the individuals with better fitness values can be kept continuously in offspring. The crossover operator needs additional actions to generate new individuals. First, two individuals are selected as parents. Next, two sub-trees are randomly selected from parents, respectively, and then swapped each other. Two new individuals are then generated. For example, Fig. 2 shows two individuals $(5 + X) + X$ and $(X + Y) - Z$. After the crossover operator is executed, two new individuals, $(X + Y) + X$ and $(5 + X) - Z$, are generated. The last operator, mutation, includes two types: single-node mutation and sub-tree mutation. In single-node mutation, a terminal or an operation in an individual is replaced. In sub-tree mutation, a whole sub-tree is replaced by a terminal or an operation. The mutation operator is usually used to avoid trapping the solution into a local optimum.

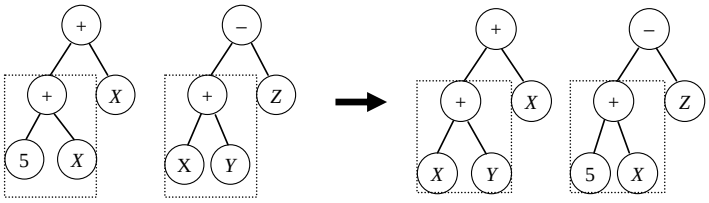

Fig. 2. A crossover operator.

After evolving for a number of generations, the individual with the best fitness value in the population can be taken as the solution. However, if the fitness values still do not satisfy the condition specified by the user, the evolution process may be continued until they are satisfied.

3.2 A Formal Description of Classification Problem

The notation used and a formal description of classification problem are first described in the following. Given a dataset S, each data $x_j \in S$ has n attributes. Let x_j be denoted as $x_j = (v_{j1}, v_{j2}, \ldots, v_{jt}, \ldots, v_{jn})$, $1 \le t \le n$, where $v_{jt} \in R$ is the tth attribute value of x_j. We assume that $C = \{C_1, C_2, \ldots, C_K\}$ is the set of K predefined classes and define $<x_j, c_j>$ as a sample if the object x_j has been assigned to a specified class c_j, $c_j \in C$. A training set (T) is then defined as a set of known samples, $T = \{<x_j, c_j> \mid x_j = (v_{j1}, v_{j2}, \ldots, v_{jn}), c_j \in C, 1 \le j \le m\}$, where m is the number of samples in T, so $|T| = m$. Let $m = (m_1 + m_2 + \ldots + m_i + \ldots + m_K)$, where m_i is the number of samples in T belonging to the class C_i, $1 \le i \le K$. A discriminant function f_i maps from R^n to R, and for a sample $<x_j, c_j>$, the function f_i should satisfy the following conditions,

$$\begin{cases} f_i(x_j) \geq a, & \text{if } c_j = C_i \\ f_i(x_j) < a, & \text{if } c_j \neq C_i \end{cases}, \text{ where } 1 \leq i \leq K, 1 \leq j \leq m.$$

A set of discriminant functions $\boldsymbol{F}$ is defined as $\boldsymbol{F} = \{f_i \mid f_i : R^n \rightarrow R, 1 \leq i \leq K\}$.

3.3 The Distance-Based Fitness Function and the Learning Algorithm

In the learning procedure, the training set T is first prepared. The samples in T include both positive instances and negative instances. Considering a specified class C_i and a sample $<x_j, c_j> \in T$, $<x_j, c_j>$ is regarded as a positive instance if $c_j = C_i$; otherwise, $<x_j, c_j>$ is a negative instance, where $1 \leq j \leq m$, $1 \leq i \leq K$. Learning can be started after the training set T is ready. Assuming that $\boldsymbol{\Omega}_1$ denotes the set of initial population, an individual $h \in \boldsymbol{\Omega}_1$ is a potential solution of a discriminant function. To evaluate whether h is good enough to be the final solution, a proper fitness function is required. A good fitness function improves not only the effectiveness of discriminant functions but also the learning efficiency. Here, a distance-based fitness function is designed for learning the discriminant functions of a classifier and is presented as follows.

Consider a discriminant function f_i for a class C_i and specify a constant a. The desired outcome is $f_i(x_j) \geq a$ for a positive instance (i.e., $c_j = C_i$) and $f_i(x_j) < a$ for a negative instance (i.e., $c_j \neq C_i$). Instead of using a directly, two parameters p and q are defined to achieve $f_i(x_j)$, where $p > a$, $q < a$ and $p + q = 2a$. The fitness function for evaluating an individual h is defined using two measurements. The first for a positive instance is defined as

$$D_p^i(x_j, c_j) = \begin{cases} 0, & \text{if } c_j = C_i \text{ and } h(x_j) \geq a \\ [p - h(x_j)]^2, & \text{if } c_j = C_i \text{ and } h(x_j) < a \end{cases}, \tag{1}$$

$$D_n^i(x_j, c_j) = \begin{cases} [h(x_j) - q]^2, & \text{if } c_j \neq C_i \text{ and } h(x_j) \geq a \\ 0, & \text{if } c_j \neq C_i \text{ and } h(x_j) < a \end{cases}. \tag{2}$$

Using Eqs. (1) and (2), we define the fitness value of an individual h for the training set T as

$$Fitness^i(h, T) = -\sum_{j=1}^{m} (D_p^i(x_j, c_j) + D_n^i(x_j, c_j)), \tag{3}$$

where $<x_j, c_j> \in T$, $1 \leq j \leq m$. Since the negative of the measurements is used as the fitness value, the best fitness value is zero. If the fitness value of the individual h is zero, then h can discriminate the samples of class C_i from those of the other classes in the given training set T. The individual h with fitness value zero thus can be chosen as the discriminant function f_i for class C_i. The detailed learning algorithm is given as Algorithm 1. The algorithm learns a single discriminant function for only a single class. The algorithm must be run K times for the K-class problem.

Algorithm 1 Genetic Programming for Learning a Discriminant Function
Input: The training set T.
Output: The discriminant function with the best fitness value.

Step 1: Initialize the population.
Let $gen = 1$ and generate the set of initial individuals $\mathbf{\Omega}_1 = \{h_1^1, h_2^1, ..., h_w^1\}$, where $\mathbf{\Omega}_{(gen)}$ is the population in the generation gen; $h_k^{(gen)}$ stands for the kth individual in $\mathbf{\Omega}_{(gen)}$; w indicates the number of individuals in $\mathbf{\Omega}_{(gen)}$.

Step 2: Evaluate the fitness value of each individual in the training set.
We compute their fitness values $E_k^{(gen)} = fitness^i(h_k^{(gen)}, T)$ for all $h_k^{(gen)} \in \mathbf{\Omega}_{(gen)}$, where the fitness evaluating function $fitness^i()$ is defined by Eq. (3).

Step 3: Decide if the conditions of termination is satisfied.
If the best fitness value of $E_k^{(gen)}$ satisfies the termination conditions or gen is equal to the specified maximum generation, then $h_k^{(gen)}$ with the best fitness value is returned and the algorithm is halted; otherwise, $gen = gen + 1$.

Step 4: Generate the next generation of individuals and go to step 2.
The new population of next generation $\mathbf{\Omega}_{(gen)}$ is generated by P_r, P_c and P_m, where P_r, P_c and P_m represent the probabilities of reproduction, crossover and mutation operations, respectively. Then goes to step 2.

Example: We give an example of Fisher's Iris dataset [10]. The data set has 150 data separated into three classes: *Setosa*, *Versicolor* and *Virginica*. Four numerical attributes, sepal length, sepal width, petal length and petal width, denote *SL*, *SW*, *PL* and *PW*, respectively. For the class *Setosa*, we first randomly generate w individuals as $\mathbf{\Omega}_1 = \{h_1^1, h_2^1, \ldots, h_w^1\}$. Each h_k^1 is an expression tree like Fig. 2. After the evolution of steps 3 and 4, one of the expression trees satisfying the termination condition may be obtained. We usually represent the expression tree as an inorder sequence for users' understanding. For example, the discriminant function for the class *Setosa* is

$$f_{setosa} = SW - PL.$$

The same procedure can be used to generate the other two discriminant functions $f_{versicolor}$ and $f_{virginica}$. The following two functions show the possible results of discriminant functions,

$$f_{versicolor} = ((((((((((((PL + PL) - (-33/PL))/-22) - 11)/(SW - PL)) \times 99) - 121) - PW) \times PL)/-31) + 45)/-21) - PW)/43),$$

$$f_{virginica} = ((((PW \times PL) + (-11/PW)) - ((((-26 - 92)/(PW \times 59))/-7)/PL))/SL).$$

4. PROPOSED CLASSIFIERS

After the discriminant functions are learned, we construct a binary decision tree for classification. Generally, a classifier cannot correctly recognize all objects in real applications. While building a classifier using discriminant functions, two situations of ambiguity will happen except in addition to misclassification.

1. An object is simultaneously recognized by more than one discriminant function.
2. An object is not recognized by any discriminant function.

Since each object belongs to a unique class, the first situation is called *conflict*, and the second is called *rejection*. In the following, we propose a classification tree based on discriminant functions and the *Z*-value measure to resolve the problems of conflict and rejection, respectively.

4.1 Decision Trees Using Discriminant Functions

We propose a decision tree based on discriminant functions (DFT) as follows. A DFT is a skew tree-like structure where nodes are discriminant functions. An unknown object first is computed by the discriminant function of the root node. If the object is identified by the root node, i.e. $f_{(1)}(x) = f_i(x) \geq a$, it is recognized as the corresponding class C_i; otherwise, we go on examining the discriminant function at the next level. While an object cannot be identified by all nodes of the decision tree, it is classified into the reject class.

As we know, it is not difficult to construct such a decision tree. The only question is at which level a discriminant function should be located. We solve this problem as the method of finding a sequence of the discriminant functions that can maximize the recognition rate from the permutation of discriminant functions. The information we have to use to determine the sequence of the discriminant functions includes two accuracy measures, the precision p_i and the recall r_i, which are defined as

$$p_i = N^i_{f_i} / N_{f_i} \text{ and } r_i = N^i_{f_i} / m_i,$$

where m_i is the number of objects belonging to class C_i, N_{f_i} is the number of objects recognized by the discriminant function f_i, and $N^i_{f_i}$ is the number of objects that belong to class C_i and are recognized by the discriminant function f_i.

From the above definitions, we know that a discriminant function f_i with higher precision p_i has a lower misclassification rate for class C_i, and a discriminant function f_i with higher recall r_i means that the f_i has a higher recognition rate for class C_i. Hence, after evaluating the precision p_i and the recall r_i for each discriminant function f_i on the training set T, the rules for generating the sequence of discriminant functions f_i are

1. If $p_i > p_j$, then f_i goes ahead of f_j for $1 \leq i, j \leq K$.
2. If $p_i = p_j$ and $r_i \geq r_j$, then f_i goes ahead of f_j for $1 \leq i, j \leq K$.

Thus, we see that the sequence can be obtained directly by sorting the pairs of $<p_i, r_i>$ in descending order with primary key p_i and secondary key r_i. Assume that the final sequence of discriminant functions after sorting is $f_{(1)}, f_{(2)}, \ldots, f_{(i)}, f_{(i+1)}, \ldots, f_{(K)}$.

Let $p_{(i)}$ and $r_{(i)}$ be the precision and the recall of $f_{(i)}$, respectively. We have $p_{(i)} \geq p_{(i+1)}$, and $r_{(i)} \geq r_{(i+1)}$ if $p_{(i)} = p_{(i+1)}$. The next example may help understand the rules more clearly.

Example: For a five-class classification problem, the discriminant function is first generated using genetic programming for each class. The training set was classified and the precision and recall of the discriminant functions are shown as Table 1. We found that f_4 and f_5 have the same precision, but the recall of f_5 is greater than f_4; thus f_5 should be ahead of f_4. The sequence of remaining functions, f_1, f_2 and f_3, is determined only by the order of precision. The final sequence of the five discriminant functions is illustrated in Table 2.

Table 1. The precision and recall of the five discriminant functions.

	f_1	f_2	f_3	f_4	f_5
Precision	0.8	0.6	0.7	1.0	1.0
Recall	0.7	0.9	0.8	0.9	1.0

Table 2. The final sequence of discriminant functions.

$f_{(i)}$	$f_{(1)}$	$f_{(2)}$	$f_{(3)}$	$f_{(4)}$	$f_{(5)}$
Function	f_5	f_4	f_1	f_3	f_2

As described in the example, the discriminant function with the highest precision and recall, f_5, is used to classify the data first. Since the data of class C_5 recognized by f_5 can be filtered out correctly, other discriminant functions have no chance to recognize them again. Hence, it increases precision and recall of other discriminant functions if they also can recognize the data of C_5. Note that we should not process the discriminant function with higher recall but lower precision before the function with high precision but lower recall, that is, f_3 should not be ahead of f_1. The reason is that although f_3 can recognize many data belonging to C_3, it also misclassifies more data that could be recognized by the other correct discriminant functions. Hence, it may decrease precision and recall of the other discriminant functions toward the rear of the sequence.

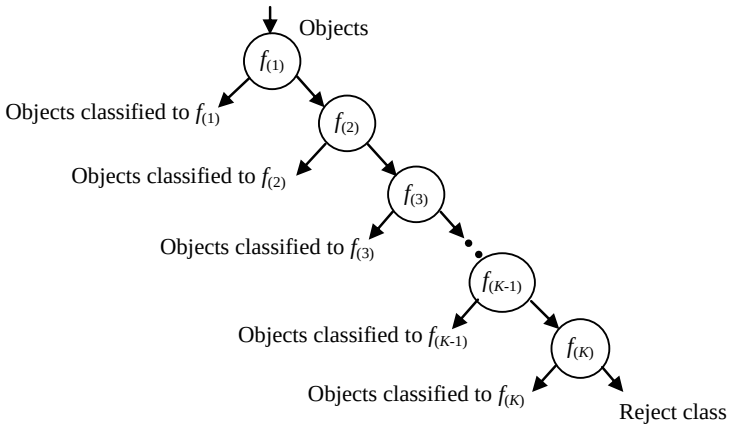

Fig. 3. A DFT: the decision tree using discriminant functions.

Actually, in our approach, two kinds of classification trees are considered. The first is to build a *K*-function decision tree, as shown in Fig. 3. Such a decision tree will produce a reject class at leaf. The classification algorithm using such kind of decision trees is presented as Algorithm 2 (DFT). The second approach uses only $K - 1$ discriminant functions. If objects cannot be recognized by the front $K - 1$ discriminant functions in the decision tree, they are directly assigned to the last class. The classification algorithm with $K - 1$ discriminant functions is listed as Algorithm 3 (DFT*).

Algorithm 2 Classification Tree with *K* Discriminant Functions (DFT)
Input: The classification tree DFT and an unknown object *x*.
Output: The classification result of *x*.
Step 1: Initially, $i = 1$ and the constant a is the same to the constant in the fitness function.
Step 2: If $i > K$, the object is assigned to the reject class and the algorithm stops.
Step 3: If $f_{(i)}(x) \geq a$, the object x is recognized by $f_{(i)}$. We output the class representing $f_{(i)}$ and the algorithm stops.
Step 4: If $f_{(i)}(x) < a$ then $i = i + 1$, go to step 2.

Algorithm 3 Classification Tree with $(K - 1)$ Discriminant Functions (DFT*)
Input: The classification tree DFT and an unknown object *x*.
Output: The classification result of *x*.
Step 1: Initially, $i = 1$ and the constant a is the same to the constant in the fitness function.
Step 2: If $i \geq K$, the object is assigned to the class representing $f_{(K)}$ and the algorithm stops.
Step 3: If $f_{(i)}(x) \geq a$, the object x is recognized by $f_{(i)}$. We output the class representing $f_{(i)}$ and the algorithm stops.
Step 4: If $f_{(i)}(x) < a$ then $i = i + 1$, go to step 2

4.2 The *Z*-Value Measure

In this subsection, we propose the *Z*-value measure to process the reject class in Algorithm 2 (DFT). This method provides a mechanism to evaluate all possible discriminant functions with ambiguous results and uses the *Z*-values to determine the class of an unknown object. First, we define the *Z*-value of a discriminant function.

For a discriminant function $f_i \in \boldsymbol{F}$ and samples $\langle x_j, c_j \rangle \in T$ with $c_j = C_i$, let μ_i be the mean of values of $f_i(x_j)$ for $1 \leq j \leq m_i$. That is,

$$\mu_i = \frac{1}{m_i} \sum_{\langle x_j, c_j \rangle \in TS,\, c_j = C_i} f_i(x_j),\ 1 \leq j \leq m_i,\ 1 \leq i \leq K. \tag{4}$$

For each μ_i, the standard deviation of values of $f_i(x_j)$, $1 \leq j \leq m_i$, is defined as

$$\sigma_i = \sqrt{\frac{1}{m_i} \sum_{\langle x_j, c_j \rangle \in TS,\, c_j = C_i} (f_i(x_j) - \mu_i)^2}, \tag{5}$$

where $1 \le j \le m_i$ and $1 \le i \le K$. For an object $x \in S$ and a discriminant function f_i, where S is the given data set, the Z-value of object x for f_i is defined as

$$Z_i(x) = \frac{|f_i(x) - \mu_i|}{\sigma_i}, \tag{6}$$

where $1 \le j \le |S|$ and $1 \le i \le K$. If the discriminant functions in $\boldsymbol{F}$ can determine the class of an object x uniquely, then the classification task is finished. However, if an ambiguous case occurs (including conflict and rejection), the Z-value measure will be applied to determine to which class the object should be assigned. The Z-value $Z_i(x)$ is a measure of the degree of $f_i(x)$ approximating μ_i. The smaller $Z_i(x)$ is, the greater likelihood that object x belongs to class C_i.

4.3 Combination of the DFT and the *Z*-value Measure

For resolving the problem of rejection in DFT, the method DFT is combined with the Z-value measure, named DFT-Z. For a K-class problem, DFT-Z uses the classification tree with K discriminant functions first to classify unknown objects. Then, the rejected objects are assigned to suitable classes using the Z-value measure. We describe the algorithm as follows.

Algorithm 4 Classification with DFT and the Z-value measure (DFT-Z)
Input: An unknown object x.
Output: The classification result of x.
Step 1: Initially, $i = 1$ and the constant a is the same to the constant in the fitness function.
Step 2: If $i > K$, go to step 5.
Step 3: If $f_{(i)}(x) \ge a$, the object x is recognized by $f_{(i)}$. We output the class representing $f_{(i)}$ and the algorithm stops.
Step 4: If $f_{(i)}(x) < a$ then $i = i + 1$, go to step 2.
Step 5: Let $\boldsymbol{Z} = \boldsymbol{F}$, where $\boldsymbol{F}$ is the set of discriminant functions as defined in section 3.2.
Step 6: Compute $Z_i(x)$, for all $f_i \in \boldsymbol{Z}$.
Step 7: Find the $k = \arg\min_{i \quad f_i \in \boldsymbol{Z}} \{Z_i(x)\}$, the object x will be assigned to the class C_k.

5. EXPERIMENTS AND DISCUSSIONS

Since GP Quick's source code is well-known and easily accessible from the web, the GP Quick 2.1 [31] was modified to fit the requirements of the proposed approaches and used to demonstrate the effectiveness and efficiency of the proposed classifiers. The experiments were conducted using a PC with 866MHz CPU and 128MB RAM. The test datasets included Fisher's Iris dataset (IRIS) [10], Wisconsin Breast Cancer dataset (WBC) [24], BUPA liver, Vehicle, and PIMA datasets. These datasets can be downloaded from the UCI Repository [2] and are well-known benchmarks for evaluating the performance of classifiers.

The classification accuracy is evaluated through 10-fold cross validation [14]. To avoid randomness of training data and show the stability of the learning process, each evaluation of 10-fold cross validation was run ten times in our experiments for a total of 100 runs. The results are listed in the Appendix. Table 3 shows the parameters of genetic programming used in GP Quick. The values used in the fitness function were empirically set to $p = 10$, $q = -10$ and $a = 0$ for all datasets, though the exact values do not significantly affect the classification accuracy if the number of evolving generations is large enough. The values $p = 10$, $q = -10$, and $a = 0$ were chosen because this setting has the shortest function generating time for most datasets.

Table 3. The parameters used in the experiments.

Parameters	Values	Parameters	Values
Node mutate weight	43.5%	Population size	1000
Mutate constant weight	43.5%	Generations per stage *g*	10000
Mutate shrink weight	13%	Crossover weight	28%
Selection method	Tournament	Crossover weight annealing	20%
Mutation weight	8%	*p*, *q*, *a*	10, – 10, 0
Mutation weight annealing	40%	Functions	+, –, ×, ÷

(1) Fisher's Iris Dataset

The first experiment used Fisher's Iris dataset (IRIS) [10], which is comprised of 150 data separated into three classes, *Setosa*, *Versicolor* and *Virginica*, each with 50 data. Each datum has four numerical attributes, sepal length (denoted as *SL)*, sepal width (*SW*), petal length (*PL*) and petal width (*PW*). After the learning procedure, 300 discriminant functions were obtained from 10-fold cross-validation repeating ten times. The experimental results are shown in Table 10 of the Appendix. The columns show the accuracy (Acc.) and number of misclassified data (# error) for each run. Clearly, DFT* and DFT-*Z* perform better than DFT. Many data are not classified correctly in DFT and are classified as reject cases, reducing the classifier accuracy. DFT* overcomes the ambiguity through its tree structure, increasing the accuracy. DFT-*Z* was found to produce the best result.

Table 4 compares the experimental results using the proposed methods with those using other methods. Due to the different experimental environments used, the comparison is described in different groups. Group *A* in Table 4 used one-half of IRIS as training data and the remaining half as test data. Group *B* used two-fold cross-validation. Group *C* used 10-fold cross-validation.

(2) Wisconsin Breast Cancer Dataset

The second experiment used Wisconsin Breast Cancer dataset (WBC) [2, 24], which contains 699 data in two classes, *Malignant* (containing 241 data) and *Benign* (458). The WBC dataset has nine numerical attributes. However, 16 WBC data have missing values; thus the remaining 683 data were used in the experiment. The *Malignant* class contained 239 data and *Benign* contained 444 data. The experimental results of 10-fold cross-validation are shown in Table 11 of the Appendix, and the experimental results are summarized in Table 5. DFT* was found to produce better than DFT and DFT-*Z* for the best

Table 4. Comparison of performance on IRIS dataset.

Group	Models or methods	Accuracy
A	DFT	94.7%
	GPCE [16]	96.0%
	FEBFC with 4 features [19]	96.7%
	FEBFC with 2 selected features [19]	97.1%
	DFT*	97.3%
	DFT-*Z*	100%
B	DLBAN [29]	93.4%
C	DFT (ave.)	92.3%
	CBA [22]	92.9%
	S-Lazy [36]	94.0%
	Naïve Bayesian [36]	94.0%
	DFT* (ave.)	94.4%
	DFT-*Z* (ave.)	94.7%
	C4.5 [22]	95.3%
	DFT (best)	95.3%
	DFT* (best)	96.7%
	DFT-*Z* (best)	96.7%

Table 5. Comparison of performance on WBC dataset.

Group	Models or methods	Accuracy
A	FEBFC with 9 features [19]	94.7%
	DFT*	94.7%
	DFT-*Z*	94.7%
	DFT	95.0%
	C4.5 [34]	95.0%
	FEBFC with selected 6 features [19]	95.1%
	HCL [34]	95.3%
B	NNFS with all features [27]	93.9%
	NNFS with selected features [27]	94.2%
C	Bayes [8]	97.3%
D	SVM [15]	95.6%
E	DFT (ave.)	95.1%
	CBA [22]	96.1%
	C4.5 [22]	96.1%
	DFT* (ave.)	96.5%
	DFT- *Z* (ave.)	96.4%
	DFT (best)	97.1%
	S-Lazy [36]	97.1%
	Naïve Bayesian [36]	97.4%
	DFT-*Z* (best)	97.4%
	DFT* (best)	97.8%

classification tree. The results for Group *A* were obtained with half the BCW data being used for training. The results of Group *B* used 315 data for training, 35 data for the validation, and 349 data for testing. The Bayesian classifier in Group *C* used 500 data for training. The SVM method used in Group *D* used 409 for training and 274 for testing. Finally, methods in Group *E* used 10-fold cross validation.

(3) BUPA Liver Disorders Dataset

The third experiment used the BUPA liver disorders dataset selected from the UCI repository [2]. BUPA is a two-class problem containing 345 data, each composed of six numerical features. The experimental results for 10-fold cross-validation are shown in Table 12 of the Appendix, and are similar to those of WBC. DFT* beats DFT-*Z* again by a very small margin, and DFT preformed the worst. Table 6 compares these results with those of other methods. The results in Group *A* were obtained using one-half of the BUPA data as the training set. The results in Group *B* were obtained using 10-fold cross-validation. The table demonstrates that the proposed FTD* classifier is close to the SVM classifiers and is better than Naïve Bayesian classifiers.

Table 6. Comparison of performance on BUPA dataset.

Group	Models or methods	Accuracy
A	HCL [34]	61.8%
	C4.5 [34]	63.1%
B	S-Lazy [36]	60.9%
	Naïve Bayesian [36]	63.2%
	DFT (ave.)	63.4%
	1-norm SVM [20]	64.3%
	MODLEM [32]	65.8%
	DFT (best)	67.8%
	DFT- *Z* (ave.)	67.8%
	DFT* (ave.)	68.0%
	Classical SVM [20]	69.9%
	DFT-*Z* (best)	70.1%
	SSVM [20]	70.3%
	DFT* (best)	71.0%

(4) Vehicle Dataset

The fourth experiment used the Vehicle dataset from UCI [2]. Vehicle is a four-class classification problem containing 846 data with 18 numerical features. The classification results of 10-fold cross-validation, shown in Table 13 of the Appendix. The experimental results reveal that DFT performed much worse than DFT* and DFT-*Z*. Vehicle dataset has four classes and thus four discriminant functions. While the data cannot be recognized by top three functions, they are assigned to the last class by DFT*. However, since the training data may not properly generate the discriminant functions, many data are misclassified into the reject class. DFT-*Z* resolves such problem and thus yields the best result. Table 7 shows the comparisons, which are split into two groups.

Table 7. Comparison of performance on Vehicle dataset.

Group	Models or methods	Accuracy
A	SVM infoprop with 18 features [30]	49.64%
	SVM infoprop with 12 features [30]	70.21%
B	DFT (ave.)	41.1%
	DFT (best)	47.8%
	DFT* (ave.)	55.2%
	DFT* (best)	58.6%
	Naïve Bayesian [36]	60.5%
	S-Lazy [36]	64.8%
	DFT-*Z* (ave.)	63.8%
	DFT-*Z* (best)	68.3%
	CBA [22]	68.8%
	C4.5 [22]	72.6%

In Group *A*, the SVM [30] uses 564 training data and 282 testing data. Group *B* uses 10-fold cross-validation. The comparison shows that DFT-*Z* has the highest classification rate in this dataset.

(5) PIMA Dataset

The last experiment used the PIMA dataset [2]. PIMA is a two-class problem, containing 768 data with 8 features. Two datasets were created for this experiment. The original PIMA dataset is denoted PIMA1. For the modified dataset, PIMA2, the serum insulin feature containing physically impossible values, was eliminated, giving 532 data, each with 7 features. Tables 14 and 15 of the Appendix present the classification results of ten runs of 10-fold cross-validation on PIMA1 and PIMA2. The tables show that DFT-*Z* performed better than the others. After removing the serum insulin feature, a 2% to 3% improvement is obtained. Table 8 compares our experimental results of the PIMA1 dataset with previous methods. Since PIMA2 was modified as in [21], comparisons of accuracy can be found in [21]. In Table 8, Group *A* used 345 data for training, 39 data for validation and 384 data for testing. Group *B* used 10-fold cross-validation. DFT-*Z* had the best single data run, but the average accuracy of DFT* tied with DFT-Z's. The experimental results show that no classification method is best overall. The original DFT method was found to perform poorly due to the serious ambiguity problems. The proposed methods, DFT* and DFT-*Z*, can resolve the ambiguity, and thus improve the classification accuracy. Furthermore, except for the BUPA dataset, DFT-*Z* is better than DFT* generally. The SVM-based classifiers also have high classification rates.

GP Quick uses a "steady state" GA, generating one new individual and replacing one old individual at a time, as opposed to making a whole new batch as a "generation" [31]. Hence, although the generation parameter was set to 10000, the evolution was fast. Table 9 lists the average learning time for discriminant functions based on the experimental datasets. The process of learning discriminant functions was found to be possible in a few seconds or minutes at most, depending on the number of instances in the training datasets. The proposed learning algorithm is efficient while comparing with the training time in [23] which is more than an hour.

Table 8. Comparison of performance on PIMA1 dataset.

Group	Models or methods	Accuracy
A	NNFS with all features [27]	71.0%
	NNFS with selected features [27]	74.3%
B	DFT (ave.)	72.9%
	CBA [22]	73.1%
	DFT (best)	73.9%
	S-Lazy [36]	74.7%
	Naïve Bayesian [36]	75.0%
	DFT* (ave.)	75.2%
	C4.5 [22]	75.5%
	DFT* (best)	76.1%
	DFT- *Z* (ave.)	76.1%
	SSVM [20]	78.1%
	DFT-*Z* (best)	78.2%

Table 9. Average learning time of discriminant functions for test datasets.

Data sets	Classification functions	Learning time (in sec.)	
		average	stddev
IRIS	f_{setosa}	2.07	0.18
	$f_{versicolor}$	3.10	0.49
	$f_{virginica}$	2.40	0.57
WBC	$f_{malignant}$	32.41	4.83
	f_{benign}	33.34	4.20
BUPA	f_1	14.78	2.32
	f_2	14.89	2.27
Vehicle	f_{opel}	33.71	4.50
	f_{saab}	34.45	6.46
	f_{van}	34.91	8.39
	f_{bus}	33.61	7.48
PIMA1	$f_{positive}$	32.51	6.40
	$f_{negative}$	32.93	8.91
PIMA2	$f_{positive}$	25.87	4.54
	$f_{negative}$	24.66	4.24

6. CONCLUSIONS

This work proposes a new classification tree with discriminant functions learned by genetic programming. This approach includes a distance-based fitness function and the resolution of ambiguity. We create a classification tree to resolve the conflict cases of discriminant functions and use the *Z*-value measure to handle the reject cases. The experimental results show that if the problem of ambiguity between discriminant functions

can be overcome, the classifier with discriminant functions is accurate and work well under various classification problems. Finally, although the proposed method works effectively with numerical attributes, it does not work on problems with categorical attributes or datasets with missing values. We hope to be able to extend the proposed methods to include these kinds of classification problems.

REFERENCES

1. C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas, "Discovering comprehensible classification rules using genetic programming: a case study in a medical domain," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 953-958.
2. C. Blake, E. Keogh, and C. J. Merz, *UCI Repository of Machine Learning Databases*, http://www.ics.uci.edu/~mlearn/MLRepository.html, Dept. of Information and Computer Science, University of California, Irvine, 1998.
3. M. Bramrier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," *IEEE Transactions on Evolutionary Computation*, Vol. 5, 2001, pp. 17-26.
4. K. H. Chen *et al*., "A multiclass neural network classifier with fuzzy teaching inputs," *Fuzzy Sets System*, Vol. 91, 1997, pp. 15-35.
5. Y. Chen and J. Z. Wang, "Support vector learning for fuzzy rule-based classification systems," *IEEE Transactions on Fuzzy Systems*, Vol. 11, 2003, pp. 716-728.
6. B. C. Chien and J. Y. Lin, "Learning discriminating functions with fuzzy attributes for classification using genetic programming," *Expert Systems with Applications*, Vol. 23, 2002, pp. 31-37.
7. T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, Vol. 13, 1967, pp. 21-27.
8. P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine Learning*, Vol. 29, 1997, pp. 103-130.
9. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.
10. R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics, Part II*, Vol. 7, 1936, pp. 179-188.
11. A. A. Freitas, "A genetic programming framework for two data mining tasks: classification and generalized rule induction," in *Proceedings of 2nd Annual Conference Morgan Kaufmann*, pp. 96-101, 1997.
12. M. A. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt, "Trends and controversies: support vector machines," *IEEE Intelligent Systems*, Vol. 13, 1998, pp. 18-28.
13. D. Heckerman and M. P. Wellman, "Bayesian networks," *Communications of the ACM*, Vol. 38, 1995, pp. 27-30.
14. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, New York, 2001.
15. B. Karacali, R. Ramanath, and W. E. Snyder, "A comparative analysis of structural risk minimization by support vector machines and nearest neighbor rule," *Pattern*

Recognition Letters, Vol. 25, 2004, pp. 63-71.
16. J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal, "Application of genetic programming for multicategory pattern classification," *IEEE Transactions on Evolutionary Computation*, Vol. 4, 2000, pp. 242-258.
17. J. R. Koza, *Genetic Programming: on the Programming of Computers by means of Natural Selection*, MIT Press, Cambridge, Massachusetts, 1992.
18. J. R. Koza, D. E. Goldberg, and D. B. Gogel, (eds.), *Genetic Programming*, MIT Press, Cambridge, Massachusetts, 1996.
19. H. M. Lee, C. M. Chen, J. M. Chen, and Y. L. Jou, "An efficient fuzzy classifier with feature selection based on fuzzy entropy," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 31, 2001, pp. 426-432.
20. Y. J. Lee and O. L. Mangasarian, "SSVM: a smooth support vector machine," *Computational Optimization and Applications*, Vol. 20, 2001, pp. 5-22.
21. T. S. Lim, W. Y. Loh, and Y. S. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," *Machine Learning*, Vol. 40, 2000, pp. 203-228.
22. B. Liu, W. Hsu, and Y. Ma, "Integrating classification and association rules mining," in *Proceedings of 4th International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 80-86.
23. T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Proceedings of the Congress on Evolutionary Computation*, 2001, pp. 1070-1077.
24. O. L. Mangasarian and W. H. Wolberg, "Cancer diagnosis via linear programming," *SIAM News*, Vol. 23, 1990, pp. 1-18.
25. D. Nauck and R. Kruse, "A neuro-fuzzy method to learn fuzzy classification rules from data," *Fuzzy Sets System*, Vol. 89, 1997, pp. 277-288.
26. J. R. Quinlan, "Induction of decision trees," *Machine Learning*, Vol. 1, 1986, pp. 81-106.
27. J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, California, 1993.
28. R. Setiono *et al*., "Neural-network feature selector," *IEEE Transactions on Neural Networks*, Vol. 8, 1997, pp. 654-662.
29. H. B. Shi, Z. H. Wang, G. Webb, and H. K. Huang, "A new restricted Bayesian network classifier," in *Proceedings of 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD*), in K. Y. Whang, J. Jeon, K. Shim, and J. Srivastava, (eds.), 2003, pp. 265-270.
30. V. Sindhwani, P. Bhattacharyya, and S. Rakshit, "Information theoretic feature crediting in multiclass support vector machines," in *Proceedings of the SIAM International Conference on Data Mining*, 2001.
31. A. Singleton, "Genetic programming with C++," *BYTE Magazine*, 1994, pp. 171-176.
32. J. Stefanowski, "Changing representation of learning examples while inducing classifiers based on decision rules," in *Proceedings of the Symposium on Methods of Artificial Intelligence* (*AI-METH*), 2003, pp. 297-301.
33. V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

34. R. Vilalta and M. Achari, "A hierarchical approach to classification for systems with complex low-level interactions," in *Proceedings of the IEEE International Symposium on Intelligent Control*, 2003, pp. 110-115.
35. C. H. Wang, T. P. Hong, and S. S. Tseng, "Integrating fuzzy knowledge by genetic algorithms," *IEEE Transactions on Evolutionary Computation*, Vol. 2, 1998, pp. 138-149.
36. Z. Wang, G. I. Webb, and F. Zheng, "Adjusting dependence relations for semi-lazy TAN classifiers," *Lecture Notes on Artificial Intelligence*, Vol. 2903, 2003, pp. 453-465.
37. G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 30, 2000, pp. 451-462.

APPENDIX

Table 10. Experimental results using 10-fold cross validation on IRIS for 10 runs.

	DFT		DFT*		DFT-Z	
Runs	Acc.	# errors	Acc.	# errors	Acc.	# errors
1	**0.953**	7	0.953	7	**0.967**	5
2	0.9	15	0.933	10	0.94	9
3	0.913	13	0.933	10	0.933	10
4	0.907	14	0.927	11	0.947	8
5	0.92	12	0.94	9	0.933	10
6	0.92	12	0.947	8	0.94	9
7	0.92	12	0.953	7	0.947	8
8	**0.953**	7	**0.967**	5	**0.967**	5
9	0.913	13	0.947	8	0.947	8
10	0.927	11	0.94	9	0.947	8
Average	0.923	11.6	0.944	8.4	0.947	8

Table 11. Experimental results using 10-fold cross validation on WBC for 10 runs.

	DFT		DFT*		DFT-Z	
Runs	Acc.	# errors	Acc.	# errors	Acc.	# errors
1	0.962	26	0.966	23	0.966	23
2	0.962	26	0.965	24	0.968	22
3	0.956	30	0.960	27	0.960	27
4	0.953	32	0.969	21	0.965	24
5	0.959	28	0.969	21	0.966	23
6	**0.971**	20	**0.978**	15	**0.974**	18
7	0.956	30	0.962	26	0.968	22
8	0.959	28	0.968	22	0.963	25
9	0.934	45	0.941	40	0.943	39
10	0.898	70	0.966	23	0.963	25
Average	0.951	33.5	0.965	24.2	0.964	24.8

Table 12. Experimental results using 10-fold cross validation on BUPA for 10 runs.

	DFT		DFT*		DFT-*Z*	
Runs	Acc.	# errors	Acc.	# errors	Acc.	# errors
1	**0.678**	111	0.701	103	0.699	104
2	0.643	123	0.693	106	0.672	113
3	0.632	127	0.661	117	0.678	111
4	0.620	131	0.658	118	0.655	119
5	0.623	130	0.652	120	0.661	117
6	0.623	130	0.678	111	0.667	115
7	0.635	126	**0.710**	100	**0.701**	103
8	0.632	127	0.675	112	0.693	106
9	0.641	124	0.704	102	0.690	107
10	0.614	133	0.670	114	0.667	115
Average	0.634	126.2	0.680	110.3	0.678	111

Table 13. Experimental results using 10-fold cross validation on Vehicle for 10 runs.

	DFT		DFT*		DFT-*Z*	
Runs	Acc.	# errors	Acc.	# errors	Acc.	# errors
1	0.371	532	0.539	390	0.599	339
2	0.382	523	0.569	365	**0.683**	268
3	0.410	499	0.545	385	0.617	324
4	0.400	508	0.535	393	0.626	316
5	0.375	529	0.509	415	0.616	325
6	0.434	479	0.577	358	**0.683**	268
7	**0.478**	442	0.569	365	0.621	321
8	0.450	465	**0.586**	350	0.664	284
9	0.398	509	0.556	376	0.665	283
10	0.409	500	0.533	395	0.609	331
Average	0.411	498.6	0.552	379.2	0.638	305.9

Table 14. Experimental results using 10-fold cross validation on PIMA1 for 10 runs.

	DFT		DFT*		DFT-*Z*	
Runs	Acc.	# errors	Acc.	# errors	Acc.	# errors
1	**0.723**	213	**0.75**	192	0.749	193
2	0.695	234	0.737	202	**0.751**	191
3	0.719	216	0.733	205	0.732	206
4	0.691	237	0.728	209	0.741	199
5	0.701	230	0.723	213	0.738	201
6	0.68	246	0.719	216	0.727	210
7	0.697	233	0.736	203	0.737	202
8	0.704	227	0.733	205	0.75	192
9	0.711	222	0.732	206	0.733	205
10	0.704	227	0.732	206	0.727	210
Average	0.702	228.5	0.732	205.7	0.738	200.9

Table 15. Experimental results using 10-fold cross validation on PIMA2 for 10 runs.

	DFT		DFT*		DFT-*Z*	
Runs	Acc.	# errors	Acc.	# errors	Acc.	# errors
1	0.718	150	0.754	131	0.765	125
2	0.731	143	0.759	128	0.761	127
3	0.729	144	0.746	135	0.748	134
4	**0.739**	139	0.75	133	0.763	126
5	0.731	143	0.752	132	0.763	126
6	0.737	140	**0.761**	127	0.761	127
7	0.733	142	0.756	129	**0.782**	116
8	0.737	140	0.754	131	0.758	129
9	0.716	151	0.752	132	0.765	125
10	0.722	148	0.737	140	0.741	138
Average	0.729	144	0.752	131.8	0.761	127.3

Been-Chian Chien (錢炳全) received a B.S. in Computer Engineering from National Chiao Tung University in 1987, an M.S. and a Ph.D. in Computer Science and Information Engineering in 1989 and 1992 from National Chiao Tung University, Hsinchu, Taiwan, respectively. He is currently an associate professor and the Head of the Department of Computer Science and Information Engineering, National University of Tainan, Tainan, Taiwan, since August 2004. His current research activities involve multimedia databases, intelligent content-based information retrieval, machine learning, knowledge discovery, and data mining.

Jung-Yi Lin (林忠億) was born in Taitung, Taiwan. He receives the B.S. degree in Applied Mathematics and the M.S. degree in Computer Science and Information Engineering from I-Shou University in 2000 and 2002, respectively. Lin is currently a Ph.D. candidate in Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. His research interests include evolutionary computing, machine learning, data mining, and knowledge discovery.

Wei-Pang Yang (楊維邦) was born in Hualien, Taiwan. He received the B.S. degree in Mathematics from National Taiwan Normal University in 1974, and the M.S. and Ph.D. degrees from the National Chiao Tung University in 1979 and 1984, respectively, both in Computer Engineering. Dr. Yang currently is a professor in Computer and Information Science in National Chiao Tung University, Hsinchu, Taiwan. Since 2004, he has transferred to National Dong Hwa University at Hualien, Taiwan, and acted as the Head of the Department of Information Management and the Dean of School of Management. His research interests include database theory and application, information retrieval, data miming, digital library, and digital museum.