Feature Selection for Modular GA-based Classification

Fangming Zhu and Steven Guan[1]

Department of Electrical and Computer Engineering

National University of Singapore

10 Kent Ridge Crescent, Singapore 119260

[1] Corresponding author: eleguans@nus.edu.sg

Feature Selection for Modular GA-based Classification

Abstract

Genetic algorithms (GAs) have been used as conventional methods for classifiers to adaptively evolve solutions for classification problems. Feature selection plays an important role in finding relevant features in classification. In this paper, feature selection is explored with modular GA-based classification. A new feature selection technique, Relative Importance Factor (RIF), is proposed to find less relevant features in the input domain of each class module. By removing these features, it is aimed to reduce the classification error and dimensionality of classification problems. Benchmark classification data sets are used to evaluate the proposed approach. The experiment results show that RIF can be used to find less relevant features and help achieve lower classification error with the feature space dimension reduced.

Keywords: classification, feature selection, genetic algorithm, class decomposition

1. Introduction

Classification problems play a major role in various fields of computer science and engineering, such as image processing and data mining. A number of soft computing approaches, such as neural networks (Anand *et al.*, 1995; Lu and Ito, 1999; Guan and Li, 2003), evolutionary algorithms (Corcoran and Sen, 1994; Bramerier and Banzhaf, 2001; Falco *et al.*, 2002), and fuzzy logic (Ishibuchi *et al.*, 1999; Setnes and Roubos, 2000), have been widely used to adaptively evolve solutions for classification problems. Among them, GA-based solutions have attracted much attention and become one of the popular techniques for classification (Merelo *et al.*, 2001).

However, when GA is applied to larger-scale real-world classification problems, it still suffers from some drawbacks, such as the inefficiency in searching a large space, the difficulty in breaking internal interference of training data, and the possibility of getting trapped in local optima. A natural approach to overcome these drawbacks is to decompose the original task into several sub-tasks based on certain techniques. Generally, a decomposition approach divides a task into smaller and simpler sub-tasks, supervises the learning of each sub-task, and finally recombines individual solutions into the final solution. Various task decomposition methods have been proposed. These methods can be roughly classified into the following categories: functional modularity, domain modularity, class decomposition, and state decomposition, according to different partition strategies (Anand *et al.*, 1995; Guan and Li, 2002; Jenkins and Yuhas, 1993; Lu and Ito, 1999).

A number of features are usually available for classification problems. However, not all of the features are equally important for a specific task. Some of them may be

redundant or even irrelevant. Better performance may be achieved by discarding some features (Verikas and Bacauskiene, 2002). In other circumstances, we may aim to reduce the dimensionality of input space to save some computation effort, although classification accuracy may be slightly deteriorated. There are many feature selection techniques developed from various perspectives such as performance (Setiono and Liu, 1997), mutual information (entropy) (Battiti, 1994; Kwak and Choi, 2002), and statistic information (Lerner *et al*., 1994).

Principal component analysis (PCA) and linear discriminant analysis are two traditional techniques used to reduce dimensionality by creating new features that are linear combinations of the original ones (Fukunaga, 1990). Fisher's linear discriminant (FLD) is the most popular goodness-score function used in feature selection. It is simple in computation and does not need strict assumptions in the distribution of features. Generally, various combinations of features in the original feature space can be evaluated with the goodness-score function by excluding some features in the feature space. Because all possible combinations of the features should be tried, the computation effort of such techniques is very high. In order to reduce computation time, some search algorithms are developed, such as knock-out and backtrack tree (Lerner *et al.*, 1994; Gonzalez and Perez, 2001).

Some feature selection techniques based on neural network and fuzzy set theory have been proposed. Setiono and Liu (1997) proposed a technique based on the performance evaluation of a neural network, where the original features are excluded one by one and the neural network is retrained and evaluated repeatedly. Pal et al. (2000) demonstrated a way of formulating neuro-fuzzy approaches for feature

selection under unsupervised learning. A fuzzy feature evaluation index for a set of features is defined in terms of degree of similarity between two patterns. Sherrah et al. (1996, 1997) presented an evolutionary pre-processor, a system which automatically extracts features for classification problems by using genetic programming.

In this paper, we employ a modular GA-based scheme for classification. This modular scheme uses class decomposition, which partitions a classification problem into several class modules in the output domain. Each module is responsible for solving a fraction of the original problem. These modules can be trained in parallel and independently, and the results obtained from them are integrated to form the final solution. Then, we propose a new feature selection technique - Relative Importance Factor (RIF) based on the optimal transformation weights from Fisher's linear discriminant function. The RIF technique can detect features that are less relevant to the classification problem and remove them from the feature space to improve classification performance in terms of accuracy. We integrate RIF into the modular GA-based scheme by employing it in finding a suitable feature subset for each class module. We aim to explore the application of feature selection in the GA domain, which appears to be missing in the literature. A modular-GA based classification approach will be more effective for RIF-based feature selection, as it is easier to find the less relevant features (LRFs) in each individual class, eliminating the interference from the other classes. Three benchmark data sets are used to evaluate the performance of RIF. The experiment results show that RIF can help achieve higher classification accuracy with the feature space dimension reduced.

We first elaborate rule-based classification with a genetic algorithm in section 2. Then, a new feature selection technique RIF with modular GA-based classification is introduced in section 3. The experiment results on benchmark data sets and their analysis are reported in section 4. Section 5 concludes the paper and presents future work.

2. Rule-based Classification with Genetic Algorithm

2.1 Encoding Mechanism

In our approach, a rule set consisting of a certain number of rules is a solution candidate for a classification problem. An IF-THEN rule is represented as follows:

$$R_i: \text{IF } (V_{1\min} \le x_1 \le V_{1\max}) \wedge (V_{2\min} \le x_2 \le V_{2\max}) \wedge ... \wedge (V_{n\min} \le x_n \le V_{n\max}) \text{ THEN } y = C \quad (1)$$

where R_i is a rule label, n is the number of features, (x_1, x_2,…, x_n) is the input feature set, and y is the output class category assigned with a value of C. V_{jmin} and V_{jmax} are the minimum and maximum bounds of the jth feature x_j respectively. Each rule R_i is encoded according to the mechanism shown in Figure 1.

Antecedent Gene 1			……	Antecedent Gene n			Consequence Gene
Act_1	V_{1min}	V_{1max}	……	Act_n	V_{nmin}	V_{nmax}	C

Note: 1. Act_j denotes whether condition j is active or inactive, encoded as 1 or 0.
2. If V_{jmin} is larger than V_{jmax} at any time, this gene will be regarded as an invalid gene. The invalid genes will make no contribution in the classification rule.

Figure 1. Encoding mechanism for classification rules

Each antecedent gene represents a feature, and the consequence gene stands for a class. Each chromosome CRj consists of a set of classification rules Ri (i=1,2,…,m) by string concatenation:

$$CR_j = \bigcup_{i=1,m} R_i \qquad j = 1,2,...,s \qquad (2)$$

where m is the maximum number of rules allowed for each chromosome (ruleNumber), s is the size of the population (popSize). Therefore, one chromosome will represent one rule set. Classifiers can learn the V_{jmin}, V_{jmax} (for each feature) and C from the training pattern. Then, a suitable step size is determined for each feature. For example, V_{jmin}=0, V_{jmax}=100, stepsize=0.1, then the possible number of values for Vj is 1000. Those values are evenly distributed, and they are not necessary the same values as presenting in the training set. Then, V_{jmin}, V_{jmax}, and C are encoded each as a character by finding their positions in the ranges. (The maximum number of characters is 2^{16}, e.g., under the *char* type in Java).

2.2 Genetic Operators

Genetic operators such as crossover, mutation, and reproduction play important roles in GA. One-point crossover is used in all experiments. Referring to the encoding mechanism, we note that crossover will not cause inconsistency and thus can take place in any point of chromosome. On the contrary, the mutation operator has some constraints. The mutation point is randomly selected with a certain probability. According to the position of a selected point, we can determine whether it is an activeness, minimum or maximum element. Different mutation is available for each. For example, if an activeness element is selected for mutation, it will just be toggled. Otherwise when a boundary-value element is selected, the algorithm will randomly select a substitute in the range of that feature. Figure 2 shows the operations of crossover and mutation. The rates for mutation and crossover are selected as 0.01 and 1.0 in our experiments (mutationRate=0.01, crossoverRate=1). For reproduction, we set the survival rate as 50% (SurvivorsPercent=50%), which means half of the parent chromosomes with higher fitness will survive into the new generation, while the other

half will be replaced by the newly created children resulting from crossover and/or mutation.

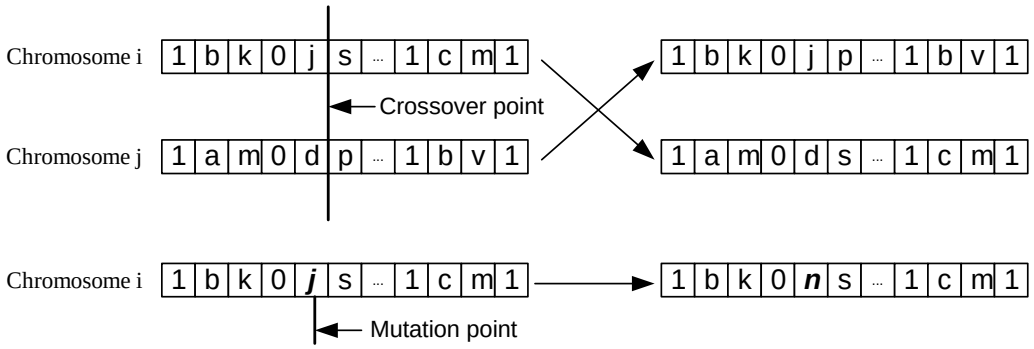

Figure 2. Crossover and mutation

Selection mechanism deals with the selection of a population that will undergo genetic operations. Roulette wheel selection (Michalewicz, 1996) is used in this paper. In this investigation, the probability that a chromosome will be selected for mating is given by the chromosome's fitness divided by the total fitness of all the chromosomes. By this means, chromosomes with higher fitness have a higher probability of producing offspring during selection for the next generation than those with lower fitness.

2.3 Fitness Function

The fitness of a chromosome reflects the success rate (i.e., classification accuracy) achieved while the corresponding rule set is used for classification. The genetic operators use this information to evolve better chromosomes over generations. As each chromosome in our approach comprises an entire rule set, the fitness function actually measures the collective behavior of the rule set. The fitness function simply measures the percentage of instances that can be correctly classified by the chromosome's rule set, which can be represented as:

$$f = \frac{C}{N} = \frac{number\ \ of\ \ instances\ \ correctly\ \ classified}{total\ \ number\ \ of\ \ instances} \qquad (3)$$

Since there is more than one rule in a chromosome, it is possible that multiple rules match the conditions for all features but predicting different classes. We use a voting mechanism to resolve conflict. That is, each rule casts a vote for the class predicted by itself, and finally the class with the highest votes is regarded as the conclusive class. If there is a tie on one instance, it means that this instance cannot be classified correctly by this rule set.

2.4 Stopping Criteria

There are three factors in the stopping criteria. The evolution process stops after a preset generation limit, or when the best chromosome's fitness reaches a preset threshold (which is set as 1.0 through this paper), or when the best chromosome's fitness has no improvement over a specified number of generations -- stagnation limit. The detailed settings are reported along with corresponding results in Section 4.

3. Feature Selection for GA-based Classification

3.1 Modular GA-based Classification with Class Decomposition

Let us assume a classification problem has *c* classes in the *n*-dimensional feature space. *p* vectors $X_i = (x_{i1},\ x_{i2},\ ...,\ x_{in})$, $i = 1,2,...,p,\ \ p >> c$, are given as training patterns. The task of classification is to assign instances to one out of the pre-defined *c* classes, by discovering certain relationship among the features. Then, the discovered rules can be evaluated by classification accuracy or error rate either on the training data or test data.

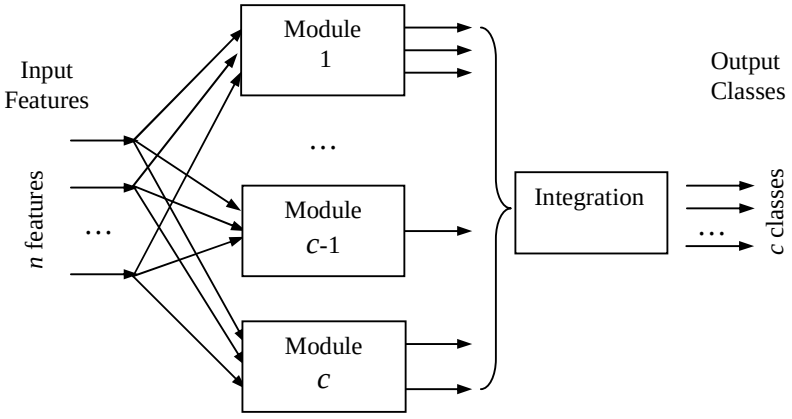

Figure 3. Illustration of modular GA-based classification with class decomposition

A traditional GA maps features to classes directly in a batch manner, which means all the features, classes, and training data are used together to train a group of GA chromosomes. Our approach -- GA with class decomposition is significantly different. As shown in Figure 3, it generally consists of three steps. Firstly, the original problem is divided into *c* sub-problems in terms of classes. Then, *c* GA modules are constructed for these sub-problems, and GA in each module will be responsible for evolving a sub-solution. Finally, these sub-solutions are integrated to form the final solution for the original problem. We present the details for each step in the following subsections.

3.1.1 Class Decomposition

The first step is to decompose a classification problem with a high-dimensional class space into sub-problems with low-dimensional class spaces, in terms of class categories.

Following the notations presented above, the original classification problem can be denoted as:

$$f: \quad X \rightarrow T \tag{4}$$

where, $X \in R^n$ is the set of features and $T \in R^c$ is the set of classes. The objective of GA is to find a certain f with a satisfactory classification rate on the whole training set ξ, which can be represented as:

$$\xi = \{(X_i, T_i)\}_{i=1}^{p} \tag{5}$$

Now the c-class problem is fully decomposed into c sub-problems. Denoting the class for each sub-problem as $T^{(j)}$, we have:

$$T = T^{(1)} \bigcup T^{(2)} \bigcup ... \bigcup T^{(k)} \tag{6}$$

Each sub-problem can be formulated as finding a certain f_j with a satisfactory classification rate on $T^{(j)}$:

$$f_j: \quad X \rightarrow T^{(j)} \tag{7}$$

3.1.2 Parallel Training

With the division of c sub-problems, classifiers can construct c GA modules and solve them in parallel. Each module is composed of the whole input features and a fraction of the class categories to produce a corresponding fraction of the original problem.

We denote:

$$\overline{T}^{(j)} = T - T^{(j)}, \quad j = 1,2,...,c \tag{8}$$

which means $\overline{T}^{(j)}$ is the complemented set of $T^{(j)}$. Then, the training set for each module can be represented as:

$$\xi_j = \left\{\left(X_q, T_q^{(j)}\right)\right\}_{q=1}^{M} \cup \left\{\left(X_q, \overline{T}_q^{(j)}\right)\right\}_{q=M+1}^{p} \qquad (9)$$

where we assume there are *M* instances in the training set whose classes belong to $T^{(j)}$, and the rest belong to $\overline{T}^{(j)}$.

Therefore, with the training of each module, GA in module *j* has two objectives. It will not only classify the data with the class in $T^{(j)}$ correctly, but also ensure that training data for the class(es) in $\overline{T}^{(j)}$ will not be wrongly classified into the class in $T^{(j)}$. In other words, for those class(es) in $\overline{T}^{(j)}$, GA will only distinguish them from the class in $T^{(j)}$. As a result, the GA in each module will converge more quickly.

3.1.3 Integration

Although each GA module has evolved a portion of the solution, we cannot just simply aggregate their sub-solutions as the final one. As discussed earlier, each GA module only classifies the class in $T^{(j)}$, but not the class(es) in $\overline{T}^{(j)}$. Therefore, when the sub-solutions are combined together, there may still exist conflicts among the sub-solutions. For example, rules from different modules may classify an instance into several classes. In order to resolve these conflicts and further improve the classification rate, the classifier employs some intelligent decision rules. The detailed integration process is explained as follows.

- The classifier constructs an overall rule set by aggregating all rules from *c* modules.
- Some decision rules are added to help resolve the above-mentioned conflicts. We believe that the ending classification rate obtained from

each module would be useful for this purpose. Currently, the following decision rules have been employed:

i) If an instance is classified into more than one class categories by the rule set, it will be classified into the class whose corresponding module achieves the highest classification rate in the parallel training phase, if available.

ii) If an instance is not classified into any class category by the rule set, it will be classified into the class whose corresponding module achieves the lowest classification rate in the parallel training phase, if available.

3.2 Relative Importance Factor (RIF) Feature Selection

Fisher's linear discriminant (FLD) algorithm projects an *n*-dimensional feature space to a *c-1* dimensional feature space by the function $y_i = w^t x_i$, in the direction *w* that maximizes the criterion function $J(w) = \frac{w^t S_B w}{w^t S_W w}$, where S_B is called as the between-class scatter matrix, and S_W the within-class scatter matrix (Duda and Hart, 2000).

As we aim to employ a feature selection technique in each class module which only distinguishes two classes, i.e., $T^{(j)}$ and $\overline{T}^{(j)}$, the projected feature space is one-dimensional (projected on one line) in this situation. Hence, the transformation matrix *w* that maximizes the criterion function *J(w)* is a vector $w = [w_1 \quad w_2 \quad ... \quad w_n]^t$. The elements in the transformation vector *w* can be viewed as weights for different features in the original feature space respectively. Thus, we can simplify the feature selection technique based on one observation: in an optimal transformation vector *w* of the Fisher's linear discriminant, a larger w_i means that the i_{th} feature is likely to be

more relevant to the module and a smaller w_i means the i_{th} feature is likely to be less relevant to the module. This observation forms the basis of the proposed RIF technique.

However, the weights obtained directly from the transformation vector *w* are not normalized. In order to derive a common feature selection metric across different sets of features in different problems, we propose a *Relative Importance Factor (RIF)*, $r = [r_1 \quad r_2 \quad ... \quad r_n]^t$, instead of using the transformation vector *w* directly for feature selection. The RIF is obtained through the following two steps (Guan and Li, 2003):

I. Normalize the length of the transformation vector *w*.

Since we are evaluating the relative importance of features, we are more interested in the relative weights of the features formed from the transformation vector *w*, which can be obtained through normalization:

$$w' = \frac{w}{\sqrt{\sum_{i=1}^{n} (w_i)^2}} \tag{10}$$

where w_i is the weight of the i_{th} feature in w, w' is the normalized transformation vector, and *n* is the number of features.

II. Render the importance factor independent of the number of features.

Since different problems have different numbers of features in their feature spaces, it is necessary to make the RIF values independent of the number of features in the feature space. This is achieved by the following function:

$$r = \frac{n}{\sum_{i=1}^{n} |w_i'|} w' \tag{11}$$

Combining (10) and (11), RIF values can be obtained from the transformation vector *w* directly as:

$$r = \frac{n}{\sum_{i=1}^{n}\left|\frac{w_i}{\sqrt{\sum_{i=1}^{n}(w_i)^2}}\right|} * \frac{w}{\sqrt{\sum_{i=1}^{n}(w_i)^2}} = \frac{n}{\sum_{i=1}^{n}|w_i|} w \qquad (12)$$

The elements of *r* represent the normalized importance of different features, which are independent of the magnitude of *w* and the number of features in the feature space.

RIF values are used as the feature selection tool in our modular GA-based classification. The feature selection technique can be summarized as follows:

Step 1: Calculate the Fisher's transformation vector *w* with respect to all features in the input feature space for each class module.

Step 2: Calculate the RIF value for each feature by using Eq. 12.

Step 3: Sort all features in each module according to their RIF values in descending order.

Step 4: In each class module, train a rule set with the original set of features first. Repeat knocking out the last feature with the least RIF value, and train a new rule set with the remaining feature set. Stop the knock-out process when the test error performance achieved with the new feature set degrades compared to the error rate achieved with the last feature set. The features knocked out can be determined as LRFs for each class module.

Step 5: Remove all LRFs from each module. A new set of features for each class module will be selected.

Step 6: Modular GA-based classification is then performed based on the new feature set for each class module, as presented in Section 3.1.

The proposed RIF technique requires much less computation time. Assume there are n input features in the original feature space. In order to obtain the relative importance of each feature, *n* FLD computations with *n-1* features included is needed each time using traditional knock-out techniques. With the RIF approach, the knock-out process can be simplified with the use of feature ranking on RIF values.

4. Experimental Results and Analysis

4.1 Experimental Scheme

We have implemented several classifiers running on three benchmark data sets to evaluate our approaches. The data sets chosen are the wine data, glass data, and diabetes data. The first one is taken from the UCI machine learning repository (Blake and Merz, 1998), and the last two are taken from the PROBEN1 collection (Prechelt, 1994). They all are real-world problems.

For benchmarking, the partitioning of data sets was adopted from the PROBEN1. Each data set is partitioned into two parts. 75% of the data instances were used for training (including the 50% training set and 25% validation set defined in the original PROBEN1 settings), while the rest 25% are used for testing. Also, three different permutations of the patterns available in the PROBEN1 were used for experiments for the glass and diabetes data. This should increase the confidence that results are independent of the particular distribution in the training and test sets.

All experiments were completed on Pentium IV 2.4GHz PCs with 768M RAM. The results reported are averaged over twenty independent runs. The parameters, such as mutationRate, crossoverRate, generationLimit, are given under the results. We recorded the evolution of each module and the integration process. We are only interested in some indicative metrics, which include initial classification error (CE), generation cost, training time, ending CE, and test CE. The CE in each generation is the lowest error rate achieved by the whole population.

We followed the six steps listed in the last section to determine the LRFs and evaluate the classifier performance with those LRFs removed. Then, by comparing to the performance of a classifier with the complete feature set, it can be shown whether the performance of our modular classifiers have improved or degraded as a result of removing LRFs. Furthermore, our results on all data sets (including all permutations) are compared with those using other approaches, such as neural networks and genetic programming, etc.

4.2 The Wine Data

The wine data contains the chemical analysis of 178 wines from three different cultivars in the same region in Italy. The analysis determines the quantities of 13 constituents found in each of the three types of wines. In other words, it has 13 continuous features, 3 classes, 178 instances, and no missing values.

Table 1 shows the RIF value for each feature in each class module. This is computed by using Eq. 12 on the available training patterns, i.e., 75% of the data instances. With the RIF feature selection technique presented in Section 3.2, it is found that feature 5

and 13 are regarded as LRFs in all class modules, as highlighted in the Table 1. Note that feature 13 has the lowest RIF values in both class modules, and feature 5 follows as the second lowest.

Table 1. RIF value for each feature in different class modules - wine data

RIF	Class=1	Class=2	Class=3
Feature 1	1.0729	1.2194	1.1621
Feature 2	0.3557	0.2543	0.9223
Feature 3	3.3961	3.0028	3.0893
Feature 4	0.3211	0.0975	0.0888
Feature 5	**0.0073**	**0.0081**	**0.0152**
Feature 6	0.9826	0.8317	0.3374
Feature 7	1.8716	0.6643	2.1069
Feature 8	1.1662	0.9327	2.4267
Feature 9	0.8742	0.6158	0.0857
Feature 10	0.0803	0.1306	0.9423
Feature 11	1.4363	4.8893	1.7115
Feature 12	1.4303	0.3511	0.1107
Feature 13	**0.0054**	**0.0024**	**0.0011**

Notes:
1. Each row in the table records the RIF value for each feature under each class module;
2. Those features determined as LRFs are highlighted.

Table 2. Performance comparison of the classifier with/without feature selection - wine data

		Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)
Using All Features	Initial CE	0.1188	0.2357	0.1459
	Generations	36.2	67.1	33.5
	T. time (s)	35.9	66.8	32.9
	Training CE	0.0008	0.0038	0.0019
	Test CE	0.0422	0.0763	0.0689
		Integration		
	Training CE	0.0064		
	Test CE	0.0944		
Removing features 5, 13 from each module	Initial CE	0.1417	0.2165	0.1342
	Generations	48.8	65.2	33.9
	T. time (s)	46.4	64.0	34.2
	Training CE	0.0045	0.0060	0.0008
	Test CE	0.0344	0.0744	0.0567
		Integration		
	Training CE	0.0113		
	Test CE	0.0689		

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=10, popSize=200, generationLimit=150, stagnationLimit=30.
3. "Initial CE" means the lowest classification error rate achieved by the initial population on the training data;
   "Generations" means the generation needed to reach the stopping criteria;

"T. time (s)" means the training time cost, and its unit is second;
"Training CE" means the lowest classification error achieved by the resulting population on the training data;
"Test CE" means the lowest classification error rate achieved by the resulting population on the test data.

4. The following tables regarding the performance of classifier follow the same notation as this table.

Table 2 shows the comparison of the classifier performance with/without feature selection on the wine data. We can find that the test CEs are improved in all modules as a result of removing all LRFs. For example, the test CE of module 3 gets an improvement from 0.0689 to 0.0567 by 17.7%. In addition, we can also find that the overall test CE is improved with a decrease from 0.0944 to 0.0689 by 27%.

4.3 The Glass Data

The glass data set contains data of different glass types. The results of a chemical analysis of glass splinters (the percentage of eight different constituent elements) plus the refractive index are used to classify a sample to be either float processed or non-float processed building windows, vehicle windows, containers, tableware, or head lamps. This data set consists of 214 instances with 9 continuous features from 6 classes.

Table 3. RIF value for each feature in different class modules – glass1 data

RIF	Class=1	Class=2	Class=3	Class=4	Class=5	Class=6
Feature 1	0.0645	0.2187	2.0409	0.1927	0.0596	0.2577
Feature 2	1.4093	1.4884	1.2032	1.3094	2.5947	1.2916
Feature 3	1.1428	0.9304	0.4649	0.8526	0.8253	0.9006
Feature 4	0.5118	0.5577	1.3768	0.5023	0.3428	0.3427
Feature 5	1.2445	1.1260	1.8705	0.7560	1.6632	1.3296
Feature 6	1.4864	1.4024	1.2752	2.5161	1.2902	1.3839
Feature 7	2.3679	2.4237	0.4870	2.3102	1.9069	1.8742
Feature 8	0.7631	0.8267	0.1806	0.5286	0.2871	1.6008
Feature 9	**0.0096**	0.0258	0.1008	**0.0320**	**0.0302**	**0.0189**

Notes:
1. Each row in the table records the RIF value for each feature under each class module;
2. Those features determined as LRFs are highlighted.

Table 3 shows the RIF values for each feature in different class modules of the glass1 data. The LRFs found are highlighted in the table. It is found that feature 9 is regarded as a LRF in the class modules 1, 4, 5, and 6, and no feature is found as LRF s in the modules 2 and 3.

Table 4. Performance of the classifier with the complete set of features – glass1 data

	Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)	Module 4 (Class=4)	Module 5 (Class=5)	Module 6 (Class=6)
Initial CE	0.3012	0.2835	0.0783	0.0683	0.0385	0.0432
Generations	138.9	138.8	72.8	94.1	75.1	39
T. time (s)	41.5	43.8	18.6	24.6	18.9	12.7
Training CE	0.0876	0.1078	0.0655	0.0177	0.0196	0.0022
Test CE	0.1509	0.2236	0.0840	0.0198	0.0462	0.0604
	Integration (Class=1, 2, 3, 4, 5, 6)					
Training CE	0.2298					
Test CE	0.3694					

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=5, popSize=100, generationLimit=150, stagnationLimit=50.

Table 5. Performance of the classifier with all LRFs removed – glass1 data

	Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)	Module 4 (Class=4)	Module 5 (Class=5)	Module 6 (Class=6)
Initial CE	0.3053	0.2835	0.0783	0.0634	0.037	0.0385
Generations	145.1	138.8	72.8	93.9	79.3	30.2
T. time (s)	42.6	43.8	18.6	24.1	20.1	9.8
Training CE	0.0925	0.1078	0.0655	0.0081	0.0177	0.0006
Test CE	0.1481	0.2236	0.0840	0.0195	0.0406	0.0585
	Integration (Class=1, 2, 3, 4, 5, 6)					
Training CE	0.2081					
Test CE	0.3538					

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=5, popSize=100, generationLimit=150, stagnationLimit=50.

The performance of the classifier trained with the complete set of features and the one with LRFs removed is shown in Table 4 and 5 respectively. Comparing the corresponding results in these tables, we can find that the test CEs for modules 1, 4, 5,

and 6 are improved, and the overall test CE is also improved from 0.3694 to 0.3538 by 4.2%.

4.4 The Diabetes Data

The diabetes problem diagnoses diabetes of Pima Indians. It has 8 features, 2 classes, and 768 instances. All features are continuous, and they are number of times pregnant, plasma glucose concentration, diastolic blood pressure, triceps skin fold thickness, 2-hour serum insulin, body mass index, diabetes pedigree function, and age.

Since the diabetes data have only 2 classes, each feature has the same RIF value in the two class modules. Table 6 shows the RIF values on the diabetes2 data, and features 4, 5, and 8 are regarded as the LRFs for both modules.

Table 6. RIF value for each feature in different class modules – diabetes2 data

RIF	Class=1/ Class=2
Feature 1	0.4750
Feature 2	3.1144
Feature 3	0.6694
Feature 4	**0.0491**
Feature 5	**0.4219**
Feature 6	1.9772
Feature 7	0.8444
Feature 8	**0.4487**

Notes:
1. Each row in the table records the RIF value for each feature under each class module;
2. Those features determined as LRFs are highlighted.

Table 7. Performance of the classifier with different set of features – diabetes2 data

Using All Features	Module 1 (Class=1)	Module 2 (Class=2)
Initial CE	0.2994	0.2761
Generations	191.1	168.5
T. time (s)	357.5	323.4
Training CE	0.1601	0.1869
Test CE	0.2979	0.3042
	Integration (Class=1, 2)	
Training CE	0.1770	
Test CE	0.2870	

Removing all LRFs	Module 1 (Class=1)	Module 2 (Class=2)
Initial CE	0.3609	0.2778
Generations	161.1	163.7
T. time (s)	297.0	311.4
Training CE	0.1700	0.1842
Test CE	0.2820	0.2924
	Integration (Class=1, 2)	
Training CE	0.1825	
Test CE	0.2689	

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=15, popSize=100, generationLimit=200, stagnationLimit=30.

With all LRFs Removed from both modules, the resulting performance of our classifier is reported in Table 7, which compares the classifier performance under two scenarios, i.e., the special case when the LRFs are removed and the normal case when all features are used for classification. We notice that the test CEs for both modules are improved, and the final test CE is also improved from 0.2870 to 0.2689 by 6.3%. These results have shown again that the removal of LRFs successfully reduces the feature pace dimension and helps improve the classifier performance.

4.5 Overall Results and Comparison to Related Work

With the same procedures as the above three experiments, more experiments have been conducted for other permutations of the glass and diabetes data. Because of the limited space, we just report the final training CE and test CE for these experiments. Furthermore, we also compared our results with those reported in four literature works. Falco et al. (2002) and Sherrah et al. (1996) used genetic programming (GP) frameworks to discover classification rules, and the later paper also suggested a feature selection pre-processor. Prechelt (1994) provided a set of benchmark data sets known as PROBEN1, and reported the performance of neural networks on these data

sets. Brameier & Banzhaf (2001) introduced a new form of linear GP and GP performance was compared with those obtained by neural networks.

Table 8. Performance comparison on the benchmark data sets

Data Set	Falco, 2002		Prechelt, 1994		Brameier, 2001		Sherrah, 1996		RIF	
	Train. CE (%)	Test CE (%)	Train. CE (%)	Test CE (%)	Train. CE (%)	Test CE (%)	Train. CE (%)	Test CE (%)	Train. CE (%)	Test CE (%)
Diabetes1	23.41	24.84	-	24.10	-	23.96	-	21.9	18.82	25.86
Diabetes2	22.71	30.36	-	26.42	-	27.85	-	-	18.25	26.89
Diabetes3	24.30	26.09	-	22.59	-	23.09	-	-	18.85	24.69
Glass1	38.25	40.92	-	32.70	-	-	-	48.1	20.81	35.38
Glass2	36.63	43.39	-	55.57	-	-	-	-	24.57	36.89
Glass3	35.39	42.63	-	58.40	-	-	-	-	25.50	40.66

Table 8 compares the performance of our RIF approach with those reported in the above-mentioned four papers. The dash line '-' inside denotes that the element was not reported in the corresponding papers. The findings in the table demonstrate the effectiveness of the RIF approach. It can be found that the training CE of the RIF approach outperforms those reported in (Falco, 2002) on all data sets. Regarding the test CE, the performance of RIF on the diabetes data is comparable to other results. Generally, it ranks in the middle among all results obtained. For instance, the test CE on the diabetes2 data attained with RIF is inferior to that reported in (Prechelt 1994), but better than those reported the other two papers. In terms of the glass data, the RIF approach performs better. It has the best results on the glass2 and glass3 data, and ranks second on the glass1 data.

Sherrah et al. (1996) employed an evolutionary pre-processor with GP to extract features. The features extracted for the diabetes data are features 3, 4, 5 and 8. In (Guan and Li, 2003), feature 4 is found as an irrelevant feature, and features 5 and 8 are regarded as boundary features. With the RIF approach, LRFs found for the

diabetes data are features 4, 5, 8 (cf. table 6), which conform to those reported in the literature.

5. Conclusions and Discussions

This paper proposes a new feature selection technique, Relative Importance Factor (RIF), to find the less relevant features (LRFs) in the input domain of a classification problem. By removing these features, we aim to improve classification accuracy and reduce the dimensionality of the classification problems. RIF is employed in modular GA-based classifiers. In this modular approach, a classification problem is decomposed into several modules in terms of class decomposition, and each module is responsible for solving a fraction of the original problem. These modules are trained in parallel, and the sub-solutions obtained from them are integrated to form the final solution. RIF is used as a feature selection technique to detect the LRFs in each class module.

Benchmark classification data sets were used to evaluate the proposed approaches. The experiment results showed that RIF can be used as a simple and yet effective feature selection technique to determine less relevant features and help achieve higher classification accuracy with the feature space dimension reduced.

The integration of RIF feature selection with a modular GA approach brings forth some advantages. First, as each module is only responsible for one class, it is easier to use RIF to find the LRFs in that particular class, eliminating the interference from other classes. Second, RIF requires relatively small computation cost compared to other feature selection techniques such as the full knock-out technique. It is based on

the statistic distribution of features in the input feature space. Furthermore, RIF is independent of the learning algorithms used, and it can also be used with other soft computing techniques such as neural network and other types of classifiers such as Bayes classifiers.

In this paper, classifiers partition the output classes in a non-overlapping manner, which means each module only tackles one class. Alternatively, classifiers can have some degrees of overlapping in class decomposition. RIF may need a modification to accommodate this overlapping situation. Furthermore, the design and implementation of GA can be improved. For example, the value of *m* (ruleNumber) is selected empirically. As future work, we will add the selection of rule number as an additional module for classifiers. Starting from one rule, the rule set is increased gradually until the performance does not improve with a further increase of the rule number. Thus, a compact rule set will be obtained finally.

Appendix

Rule Set Samples for the Diabetes Data

The following two lists show the resulting rule sets for class module 1 of the diabetes2 data before and after feature selection respectively - removing features 4, 5 and 8 (cf. Table 6 and 7). We can see that features 4, 5 and 8 (X4, X5, and X8 in the rule set) do not appear in the second list, as they have been removed from the feature space. It is also found that the rule set after feature selection is more concise.

(Rule set for module 1 with all features)

```
1. IF (0.65<=X1<=0.73) AND (0.30<=X2<=0.60) AND (0.26<=X3<=0.49) AND (0.20<=X4<=0.44)
   AND (0.44<=X5<=0.49) AND (0.73<=X7<=0.94) AND (0.13<=X8<=0.19) THEN Class=1
2. IF (0.44<=X1<=0.98) AND (0.68<=X2<=0.88) AND (0.37<=X6<=0.69) THEN Class=1
3. IF (0.78<=X2<=0.96) AND (0.17<=X3<=0.70) AND (0.11<=X7<=0.64) AND (0.10<=X8<=0.32)
   THEN Class=1
4. IF (0.27<=X4<=0.71) AND (0.44<=X7<=0.64) THEN Class=1
5. IF (0.52<=X2<=0.64) AND (0.18<=X4<=0.57) AND (0.17<=X5<=0.24) AND (0.09<=X8<=0.71)
   THEN Class=1
6. IF (0.55<=X6<=0.88) AND (0.07<=X7<=0.58) AND (0.58<=X8<=0.97) THEN Class=1
7. IF (0.72<=X6<=0.84) THEN Class=1
8. IF (0.81<=X2<=0.99) THEN Class=1
9. IF (0.57<=X2<=0.71) AND (0.31<=X3<=0.82) AND (0.48<=X5<=0.64) AND (0.17<=X8<=0.57)
   THEN Class=1
10.IF (0.18<=X1<=0.79) AND (0.31<=X3<=0.60) AND (0.90<=X5<=0.91) AND (0.43<=X6<=0.69)
   AND (0.57<=X8<=0.63) THEN Class=1
11.IF (0.06<=X1<=0.96) AND (0.20<=X4<=0.78) AND (0.15<=X6<=0.58) AND (0.41<=X7<=0.76)
   AND (0.15<=X8<=0.32) THEN Class=1
12.IF (0.04<=X4<=0.89) AND (0.27<=X5<=0.45) AND (0.45<=X7<=0.78) AND THEN Class=1
13.IF (0.57<=X3<=0.61) AND (0.18<=X6<=0.22) AND (0.16<=X7<=0.16) AND (0.73<=X8<=0.90)
   THEN Class=1
14.IF (0.24<=X4<=0.81) AND (0.06<=X7<=0.69) AND (0.38<=X8<=0.63) THEN Class=1
15.IF (0.30<=X2<=0.76) (0.02<=X5<=0.46) AND (0.55<=X6<=0.77) AND (0.64<=X7<=0.97) THEN
   Class=1
```

(Rule set for module 1 with feature selection – features 4, 5 and 8 are removed)

```
1. IF (0.16<=X3<=0.45) AND (0.79<=X7<=0.96) THEN Class=1
2. IF (0.50<=X2<=0.98) AND (0.81<=X3<=0.86) THEN Class=1
3. IF (0.64<=X2<=0.72) AND (0.54<=X7<=0.92) THEN Class=1
4. IF (0.84<=X3<=1.01) AND (0.55<=X6<=0.89) THEN Class=1
5. IF (0.71<=X6<=0.96) THEN Class=1
6. IF (0.00<=X1<=0.19) AND (0.46<=X2<=0.83) AND (0.47<=X6<=0.55) AND (0.25<=X7<=0.81)
   THEN Class=1
7. IF (0.17<=X1<=0.70) AND (0.24<=X3<=0.34) AND (0.07<=X6<=0.31) AND (0.34<=X7<=0.84)
   THEN Class=1
8. IF (0.81<=X2<=1.02) THEN Class=1
9. IF (0.51<=X1<=0.94) AND (0.45<=X2<=0.96) AND (0.47<=X6<=0.85) AND (0.16<=X7<=0.58)
   THEN Class=1
10.IF (0.04<=X1<=0.39) AND (0.24<=X2<=0.50) AND (0.19<=X6<=0.51) AND (0.81<=X7<=0.83)
   THEN Class=1
11.IF (0.64<=X3<=0.81) AND (0.44<=X7<=0.65) THEN Class=1
12.IF (0.93<=X1<=0.97) AND (0.04<=X2<=0.94) AND (0.53<=X3<=0.71) AND (0.00<=X7<=0.77)
   THEN Class=1
13.IF (0.12<=X1<=0.42) AND (0.46<=X7<=0.65) THEN Class=1
14.IF (0.84<=X1<=0.93) AND (0.21<=X2<=0.27) AND (0.67<=X7<=0.81) THEN Class=1
15.IF (0.46<=X1<=0.69) AND (0.68<=X2<=0.92) AND (0.20<=X3<=0.99) THEN Class=1
```

Acknowledgements

The first author is grateful to the Singapore Millennium Foundation for the scholarship awarded.

References

Anand, R., Mehrotra, K., Mohan, C.K., and Ranka, S. 1995. Efficient classification for multiclass problems using modular neural networks. IEEE Transactions on Neural Networks, 6 (1), pp. 117-124.

Battiti, R. 1994. Using mutual information for selecting features in supervised neural net learning. IEEE Transactions on Neural Networks, 5 (4), pp. 537-550.

Blake, C.L. and Merz, C.J. 1998. UCI Repository of machine learning databases (http://www.ics.uci.edu/~mlearn/MLRepository.html). Irvine, CA: University of California, Department of Information and Computer Science.

Brameier, M. and Banzhaf, W. 2001. A comparison of linear genetic programming and neural networks. IEEE Trans. on Evolutionary Computation 5 (1), pp. 17–26.

Corcoran, A.L. and Sen, S. 1994. Using real-valued genetic algorithm to evolve rule sets for classification. Proceedings of the 1st IEEE Conference on Evolutionary Computation, Orlando, US, pp. 120-124.

Duda, R.O., Hart, P.E., and Stork, D.G. 2000, Pattern Classification, New York: Wiley, 2nd Edition.

Falco, I.D., Cioppa, A.D., and Tarantino, E. 2002. Discovering interesting classification rules with genetic programming. Applied Soft Computing, 1, pp. 257-269.

Fukunaga, K. 1990, Introduction to Statistical Pattern Recognition, 2nd ed., Boston: Academic Press.

Gonzalez, A. and Perez, R. 2001. Selection of relevant features in a fuzzy genetic learning algorithm. IEEE Transactions on Systems, Man and Cybernetics, Part B, 31 (3), pp. 417-425.

Guan, S.U. and Li, S.C. 2002. Parallel growing and training of neural networks using output parallelism. IEEE Transactions on Neural Networks, 13 (3), pp. 542-550.

Guan, S.U. and Li, P. 2003. Feature selection for modular neural network classifiers. Journal of Intelligent Systems, 12 (3), pp. 113-139.

Ishibuchi, H., Nakashima, T., and Murata, T. 1999. Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. IEEE Transactions on Systems, Man and Cybernetics, Part B, 29 (5), pp. 601-618.

Jenkins, R.E. and Yuhas, B.P. 1993. A simplified neural network solution through problem decomposition: the case of the truck backer-upper. IEEE Transactions on Neural Networks, 4 (4), pp. 718-720.

Kwak, N. and Choi, C.H. 2002. Input feature selection for classification problems. IEEE Transactions on Neural Networks, 13 (1), pp. 143-159.

Lerner, B., Levinstein, M., Rosenberg, B., Guterman, H., Dinstein, L., and Romem, Y. 1994. Feature selection and chromosome classification using a multilayer perceptron neural network. IEEE International Conference on Neural Networks, vol. 6, pp. 3540-3545.

Lu, B.L. and Ito, M. 1999. Task decomposition and module combination based on class relations: a modular neural network for pattern classification. IEEE Transactions on Neural Networks, 10 (5), pp. 1244-1256.

Merelo, J.J., Prieto, A., and Moran, F. 2001. Optimization of classifiers using genetic algorithms. In: Patel, M., Honavar, V., Balakrishnan, K. (Eds.), Advances in the Evolutionary Synthesis of Intelligent Agents. MIT Press, Cambridge.

Michalewicz, Z. 1996. Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed. Springer-Verlag, New York.

Pal, S.K., De, R.K., and Basak, J. 2000. Unsupervised feature evaluation: a neuro-fuzzy approach, IEEE Transactions on Neural Networks, 11 (2), pp. 366 –376.

Prechelt, L. 1994. PROBEN1: A set of neural network benchmark problems and benchmarking rules, Technical Report 21/94, Department of Informatics, University of Karlsruhe, Germany.

Setiono, R. and Liu, H. 1997. Neural network feature selector. IEEE Transactions on Neural Networks, 8 (3), pp. 654-662.

Setnes, M. and Roubos, H. 2000. GA-Fuzzy modeling and classification: complexity and performance. IEEE Transactions on Fuzzy Systems 8 (5), pp. 509-522.

Sherrah, J., Bogner, R.E., and Bouzerdoum, A. 1996. Automatic selection of features for classification using genetic programming. Proc. of the IEEE Australian and New Zealand Conference.

Sherrah, J., Bogner, R.E., and Bouzerdoum, A. 1997. The evolutionary pre–processor: automatic feature extraction for supervised classification using genetic programming. Proc. of the Second Annual Genetic Programming Conference.

Verikas, A. and Bacauskiene M. 2002, Feature selection with neural networks. Pattern Recognition Letters 23, pp.1323-1335.