MINING MULTIPLE COMPREHENSIBLE CLASSIFICATION RULES USING GENETIC PROGRAMMING

K. C. Tan, A. Tay, T. H. Lee and C. M. Heng
Department of Electrical and Computer Engineering
National University of Singapore
10 Kent Ridge Crescent Singapore 119260

Abstract: **Genetic Programming (GP) has been emerged as a promising approach to deal with classification task in data mining. This work extends the tree representation of GP to evolve multiple comprehensible *IF-THEN* classification rules. In the paper, we introduce a concept mapping technique for fitness evaluation of individuals. A covering algorithm that employs an artificial immune system-like memory vector is utilized to produce multiple rules as well as to remove redundant rules. The proposed GP classifier is validated upon nine benchmark datasets and the simulation results confirm the viability and effectiveness of the GP approach for solving data mining problems in a wide spectrum of application domains.**

I. INTRODUCTION

Data mining is an automated process of extracting structured knowledge from databases, which is often referred to as a particular step in the overall process of discovering useful knowledge from data, called knowledge discovery from database (KDD) [6]. In recent years, there have been numerous attempts to apply evolutionary algorithms (EAs) in data mining to accomplish various tasks [1-4, 7, 10, 14, 17, 18]. One of the data mining tasks gaining significant attentions is the rules/knowledge extraction from databases. Along with other evolutionary algorithms, genetic programming (GP) has emerged as a promising technique to discover useful and interesting knowledge from the database. Genetic programming has been formulated originally as an autonomous method for breeding computer programs using tree structures [11].

GP starts with a population of randomly created individuals that represents the possible solutions to a given problem. The principle elements of the GP are a set of functions and terminals that are able to represent the solution of the problem. The function set may consist of arithmetic operators, mathematic functions, Boolean operators, conditional and branch operators etc., whereas the terminal set may consist of problem specified variables and/or constant values. For every generation, each individual will be evaluated for its fitness, and individuals of the next generation will be produced from the parents selected based on their fitness value. The population evolves through genetic reproduction, crossover and mutation over a number of generations until the termination criteria is met. At the end of the GP run, the best individual is presented as the solution to the problem. In this work, we extend the idea of booleanized attributes [3] and use the Michigan approach to encode the rules, where each individual encodes only one rule. With this configuration, a simpler structure that is easy for evaluation with good comprehensibility could be produced. The proposed GP classifier is tested on nine sets of data obtained from UCI Machine Learning Repository, which is a collection of widely used benchmark and real-world data for data mining and KDD community.

This paper studies the application of GP in data mining by evolving a set of comprehensive decision rules with high classification accuracy. A novel concept mapping technique is also introduced for efficient fitness evaluation. To evolve a set of classification rules that are compatible to each other, i.e., rules that can be used collectively as a whole to classify the instances without conflicting each other, a covering algorithm is utilized. The covering algorithm uses an immune system-like memory vector to hold discovered good rules such that redundant rules could be checked and removed. Therefore, the covering algorithm effectively enables the production of multiple rules for each class when necessary. The paper is organized as follows: In Section II, the classification task in data mining is introduced in general. Section III presents the proposed GP classifier system and discusses the approach of concept mapping for fitness evaluation and the covering algorithm. The problem sets used for validation are presented in Section IV and the simulation results are analyzed and summarized. Conclusions and future work are given in Section V.

II. CLASSIFICATION IN DATA MINING

Given a set of classified examples, the goal of classification is to find a logical description that correctly classifies novel cases. The three basic requirements of a classifier are accuracy, simplicity and efficiency [5]. In the classification task, the discovered knowledge is usually represented in the form of decision trees or *IF-THEN* prediction rules, which have the advantage of being a high-level and symbolic knowledge representation contributing towards the comprehensibility of the discovered knowledge. In this paper, knowledge is presented as multiple *IF-THEN* rules in a decision rules list. Such rules state that the presence of one or more items (antecedents) implies or predicts the presence of other items (consequents). A typical rule has the form of

Rule: *If* X_1 *and* X_2 *and* ... X_n *then* Y,

where X_i, $\forall\ i \in \{1,2,\ldots,n\}$ is the antecedent that leads to the prediction of consequent Y.

0-7803-7282-4/02/$10.00 ©2002 IEEE

One reason of using classification rules instead of decision tree is that each rule can be seen as an independent piece of knowledge. New rules can be added to an existing rule set without disturbing those already there. Multiple rules can be combined together to form a set of decision rules. This set of decision rules is usually listed according to the "quality" of the rules, with the best rule listed first. When the decision rules list is use to predict a new instance, the best rule will be considered first. If the rule does not match the instance, i.e., the antecedents of the rule do not satisfy the value of the attributes in the instance, then the next rule will be considered. At the case where none of the rules in the decision list satisfies the new instance, the predetermined default prediction will be used. The default prediction class is the largest class in the training set in this case (the class with the largest number of instances). The basic structure of the decision rules list could be built as follows:

IF (antecedents)$_1$ *THEN* class$_1$
ELSE IF (antecedents)$_2$ *THEN* class$_2$
... *ELSE* default class

The discovered decision rules can be evaluated according to several criteria, such as classification accuracy rate on unlabeled instances (testing set), degree of confidence in the prediction, comprehensibility, and interestingness. Classification accuracy is the major metric to evaluate the performance of a classifier. Comprehensibility measures how clear and easy a rule is for human to understand and take action on it accordingly. Rules that are incomprehensible to human are often useless in the sense of data mining and knowledge discovery because such rules are not beneficial to the user. Interestingness measures surprisingness (unexpectedness), usefulness and novelty of the discovered knowledge [8]. As compared to classification accuracy and comprehensibility, the interestingness of a rule is still more difficult to measure. This paper only considers the predictive accuracy of the decision rules and comprehensibility of the rules.

III. THE GENETIC PROGRAMMING CLASSIFIER

A. Flowchart of GP Classifier

The GP used in this paper is based upon a modified version of steady state GP [1]. Fig. 1 depicts the complete learning flow of the GP Classifier. The GP starts with an initial population. The initial population is created with 'ramped-half-and-half' method [11]. The elitism preserving strategy is used to ensure that some best individuals of current generation will survive in the next generation according to the elitism rate, *e*. The tournament selection scheme with tournament size of 2 is implemented to reduce the selection pressure and for good convergence. After the genetic operations (i.e. crossover, mutation and reproduction) are applied to the individuals, the genetically modified individuals together with the preserved elitist will form the new population. These individuals will pass through a covering algorithm to penalize the redundant individuals and to promote the individuals that cover the solution space well. The working principle of the token competition will be discussed later in Section 3.5. In addition to passing the outcome of token competition directly into the new generation, individuals that are able to seize any token (instance) will be added into a rule vector as shown in Fig. 1.

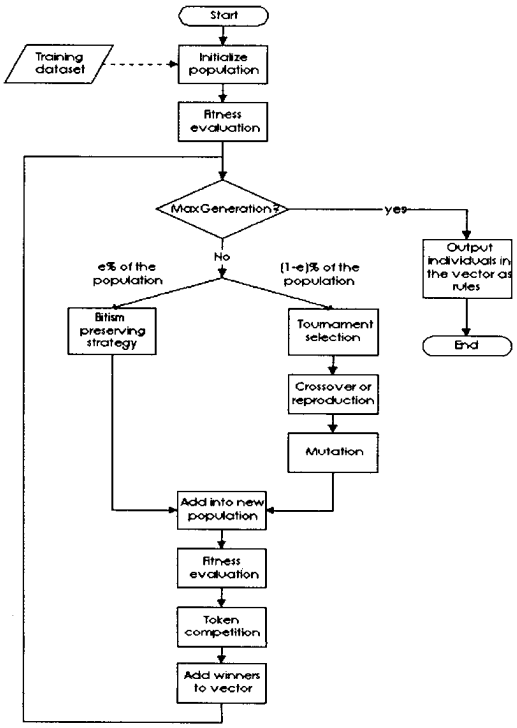

Fig. 1 Flowchart of the proposed genetic programming classifier

The rule vector is similar to the memory-cell concept in artificial immune system [9], which is used to maintain a memory on previously found good solutions. The rule vector is maintained such that no redundant rules may exist and previously encountered good rules are memorized for subsequent competition. All individuals in the rule vector and current population will participate in the token competition performed to ensure that there will be no redundant rule in the rule vector. The complete GP process is performed for every class of the dataset, that is, for an *n* class problem, there will be *n* GP run. The rule vector of every GP run is combined into a global decision list vector. At the end of the training process, the global decision list vector will be presented as the final solution to a problem. The GP-based classifier developed in this work, namely, Genetic Programming Classifier (GPc) is capable of evolving a classification rules list, which is implemented in Java™ and is based on the Groovy Java Genetic Programming (gjprog) package. The GPc is then integrated into the *Waikato Environment for Knowledge Analysis* (WEKA) [16], a Java-based public domain data mining package, which is used as the experiment and analysis environment in this work.

B. Function and Terminal sets

For the GP classifier to be capable of evolving a classification rule, two Boolean operators, AND and NOT, are used. These two functions are sufficient to build a basic classification rule in the form of, e.g.,

> *If* antecedent$_1$ AND (NOT antecedent$_2$) AND ... *then* consequent.

The classification rules that are built up of AND and NOT can then combined to form the decision rules list (the OR effect). This can be effectively realized with a covering algorithm, such as token competition. The terminal set contains all possible attribute-value pairs of a given dataset. That is, the terminal set (T) is defined as,

$$T = \{\text{attr}_1\text{-value}_{11}, \text{attr}_1\text{-value}_{12}, \ldots \text{attr}_1\text{-value}_{1n}, \text{attr}_2\text{-value}_{21}, \text{attr}_2\text{-value}_{22}, \ldots, \text{attr}_2\text{-value}_{2n}, \ldots, \text{attr}_m\text{-value}_{m1}, \text{attr}_m\text{-value}_{m2}, \ldots, \text{attr}_m\text{-value}_{mn}\}$$

where m is the number of attributes in the dataset and n is the number of values for each respective attribute. To avoid redundant and conflicting nodes exist in the same tree, these terminals are built into a table and only one *attribute-value* pair can be selected from each attribute entry for a tree structure. Each individual is encoded as a single rule. The population is structured such that all individuals in a particular GP run predict the same class (there will be n GP run for n classes). With this structure, there is no encoding needed for the class value.

C. Fitness Function

The fitness function evaluates the quality of each rule or individual, which is based on the evaluation function previously proposed by [12]. The fitness function is defined as,

$$fitness = \frac{tp}{(tp+w_1 fn)} \times \frac{tn}{(tn+w_2 fp)} \quad (1)$$

The *tp*, *fp*, *tn* and *fn* stands for true positive, false positive, true negative and false negative, respectively. These are four possible outcomes or concepts resulted from classifying an instance. The true positive and true negative are correct classifications, while false positive and false negative are incorrect classifications. Consider a two-class case with classes yes and no, for example,

True positive (*tp*): the rule predicts that the class is yes (positive) and the class of the given instance is indeed yes (positive).

False positive (*fp*): the rule predicts that the class is yes (positive) but the class of the given instance is in fact no (negative).

True negative (*tn*): the rule predicts that the class is no (negative) and the class of the given instance is indeed no (negative).

False negative (*fn*): the rule predicts that the class is no (negative) but the class of the given instance is in fact yes (positive).

These observations can be presented as a confusion matrix shown in Table 1. The fitness function has a value between 0 and 1. The weights (w_1 and w_2) enable the dependency of fitness function on different concepts (i.e., *tp*, *fp*, *tn* and *fn*) to be controlled effectively, especially when the classifier is used to deal with datasets that have variety of characteristics from different domains, as in this experiment. With $w_1 = 1.0$ and $w_2 = 1.0$, the first part of the fitness function is known as *sensitivity* and the second part is known as *specificity*. In medical domain, these two parameters are used as performance measures of a classifier test [12]. The weights are controlled such that decreasing w_1 or increasing w_2 will usually improve the prediction accuracy but increase the tendency of over-fitting (i.e., increase in number of rules). The range of 0.2–1 for w_1 and the range of 1–20 for w_2 are found to be reasonable settings for most cases.

TABLE 1 CONFUSION MATRIX OF CONCEPTS

		Predicted class	
		Yes	No
Actual class	Yes	*tp*	*fn*
	No	*fp*	*tn*

D. The Concept Map

For rule evaluation, the use of concept map is proposed in this paper. For every GP run, once the concept table of a terminal on the training cases is built, the concept table is stored in the memory for future reference. This reduces the need to rescan the training set and thus makes the fitness evaluation more efficient.

The derivation of concept map is as follows: The problem space is restricted to 2-class problem. Therefore, an n-class problem must be solved as n 2-class problem. As described earlier in the section, the whole population in a single GP run represents one predicted class. For example, assume a six-class dataset with class {1, 2, 3, 4, 5, 6}. When the GP is run for class 1, positive instances are instances with class 1, and negative instances are instances with class other than 1, i.e., {2, 3, 4, 5, 6}. With this setup, 2 properties can be observed:

(1) An antecedent with positive concept (*tp* or *fp*) when '*AND*' together with another antecedent with negative concept (*tn* or *fn*) will always produce a negative concept.
(2) Two antecedents with same concept on an instance when '*AND*' together will learn the same concept again.

Additionally, a rule will never have two antecedents that when used separately, will predict a given instance with different classes. That is, for the antecedents {*A*, *B*}, the following will never happen and hence can be set as *don't care* (*X*):

A (tp) AND B (tn) *A (fp) AND B (fn)*

A (tp) AND B (fp) *A (tn) AND B (fn)*

The *NOT* function is reflected with very simple property as well. A *NOT* to an antecedent will cause a positive instance become negative instance ($Xp \leftrightarrow Xn$) and true concept become false concept ($tX \leftrightarrow fX$), and vice versa. These observations are summarized into Table 2 named *concept maps*, which are used to program the execution of function *AND* and *NOT*, respectively. Therefore the *AND* and *NOT* functions implemented in the paper are different from conventional logical *AND* and *NOT*.

TABLE 2: THE CONCEPT MAP (A) *AND* (B) *NOT*

(A)

	tp	*fp*	*tn*	*Fn*
tp	*tp*	*X*	*X*	*fn*
fp		*fp*	*tn*	*X*
tn			*tn*	*X*
fn				*fn*

(B)

tp	*fn*
fp	*tn*
tn	*fp*
fn	*tp*

E. Covering Algorithm

The token competition (TC) technique proposed by Wong and Leung (2000) is employed as an alternative niche approach to promote the diversity and to evolve multiple rules. Multiple rules that cover the same instances of training set increase the tendency of premature convergence. Most of the time, only a few of these rules are useful and cover most of the instances while most others are redundant. The TC is an effective way to eliminate redundant rules and its working principle is as follows: In token competition, instance in the training set is called *token* or resource. Instances that are covered by an individual are the *territory* of that individual. Territories may overlap and all individuals should try their best to prevent others from sharing their territory. Individuals of current generation and existing individuals in the rule vector will compete for tokens to survive, following the token competition algorithm. Individuals that seize tokens less than the total number of tokens in its territory will be penalized. Individuals that are failed to seize any token will soon extinct from the population. That is, the more tokens an individual can seize, the more likely it will survive. The penalization is done by adjusting the fitness according to eqn. 2. Fig. 2 shows the pseudo code of the modified token competition used in this paper.

$$adjusted\_fitness = fitness \times \frac{token\_count}{ideal\_count} \qquad (2)$$

The *ideal_count* is the maximum number of token in an individual's territory, that is, the total instances in the training set that match the individual. However, because of the existence of competition, an individual usually will not be able to seize all available tokens. The actual number of tokens that are successfully taken by an individual is reflected by its *token_count* value. At the end of the token competition, all individual will be re-evaluated according to the *token_count* to *ideal_count* ratio as shown in eqn. 2. As such, individuals that are able to get tokens up to their respective ideal count will maintain their original fitness. For individuals that are only successful in seizing part of the tokens, the *token_count* to *ideal_count* ratio will penalize them accordingly. Those individuals that success in the token competition will be added into the rule vector and wait for new individuals from the next generation for another round of competition. Note that the TC algorithm allows any individuals that are able to get at least a token to survive to the next generation and in the rule vector. This in many cases will cause the size of the rule vector to increase significantly. This is an unwanted situation because it will decrease the effectiveness of token competition. As such, the token competition is set so that an individual would only survive if its ratio of *token_count* to the total number of training instances (N) is greater than the predefined minimum support. That is, if $\frac{token\_count}{N} < min\_support$, then the individual should be considered as bad individual and thus be removed.

```
Set the flag of all instances to -1
Set ideal_count and token_count of all individuals to 0
For all individuals in the population and individuals in the rule vector, do
   For all instances in the training set, do
      If instance is in the territory of individual
         ideal_count ← ideal_count + 1
         If this instance haven't been seized by any individual
            Flag as token
         Else
            Challenge the owner of the instance by fitness
            Flag as token if fitter, otherwise leave it alone
   Repeat for the next instance
Repeat for the next individual
Calculate the token_count of each individual by scanning the flags
Adjust the fitness of all individuals
```

Fig. 2 Pseudo code of Token Competition

IV. CASE STUDY

A. Problem Sets

To evaluate the performance of the proposed GP classifier, the system is applied to some selected artificial as well as real-world datasets from the UCI machine learning repository. Table 3 summarizes the datasets used in this work.

TABLE 3 PROBLEM COMPLEXITY OF THE DATASETS TESTED

Dataset	Attribute	class	Instances	missing
Weather	4	2	14	no
Contact-lenses	4	3	24	no
Zoo	17	7	101	no
Breast-Cancer	9	2	286	yes
Monks-1	6	2	432	no
Monks-2	6	2	432	no
Monks-3	6	2	432	no
Mushroom	22	2	8124	yes
Nursery	8	5	12960	no

All experiments are carried out using split test with 2/3 of dataset as training set, and 1/3 as test set. Numeric attributes are discretized to nominal values before feeding to the GP classifier. All datasets are experimented with 30 independent runs using the GP classifier (except only 15 runs for the problem of nursery). Table 4 lists the common parameter settings used for all problems. Table 5 lists the GP parameters that are set according to the problem complexity. The fitness

of an individual is computed using the complete training set as fitness cases. The decision rules list obtained by the GP classifier is evaluated using the test set. It can be observed that the parameter settings vary greatly with the problem. Most of the experiments are done with most common settings of GP since tuning GP parameters for every problem is particularly difficult as the problem domains vary greatly with different set of data.

TABLE 4 COMMON PARAMETER SETTING FOR GP CLASSIFIER

Parameter	Setting
Function set	AND, NOT
Terminal set	All possible attribute-value pairs
Tree initialisation method	Ramped-half-and-half
Fitness cases	All instances of training set
Maximum crossover depth	17
Mutation probability	0.1
Crossover probability	0.9
Reproduction probability	0.1
Elitism rate	0.05
Minimum support	0.01

TABLE 5 SETTINGS BASED ON PROBLEM COMPLEXITY

Dataset	Population size	Maximum generation	Initial tree depth	w_1	w_2
Weather	10	10	2-3	0.5	2
Contact-lenses	25	25	2-3	0.5	2
Zoo	25	25	2-6	1.0	1
Breast-Cancer	50	50	2-6	1.0	1
Monks-1	50	50	2-6	0.5	10
Monks-2	50	25	2-6	0.2	12
Monks-3	50	25	2-6	0.5	2
Mushroom	50	50	2-6	0.5	8
Nursery	100	50	2-8	0.5	20

B. Results and Analysis

Table 6 shows the classification accuracies, average number of rules and typical run time for each problem. For clean data, where there is no missing value and have explicit knowledge (such as the *zoo* dataset), the GP classifier can evolve the expected rules easily with high classification accuracy. Decision rules list with perfect accuracy can be achieved easily for these datasets. The GPc is found robust to the existence of missing values in the datasets (*breast-cancer* and *mushroom* datasets). The missing value is simply assigned as negative concept and without any special handling technique, the GPc is able to tolerate the missing values with some degradation to the classification accuracy. All datasets except *zoo* dataset found number of rules greater than their respective number of class to represent the dataset. Without the covering algorithm, GPc will produce only one rule per class, which is insufficient for good classification in most cases. Generally, the accuracy increases as the number of rules grows. However, having too many rules to represent a dataset is not preferable in a practical extent. Therefore, the number of rules is restricted with the trade off on accuracy. One notable result seems contrary to this aspect is that for *monk2* problem, the classification accuracy (58.85%) is obtained with a relative large number of rules. This is mainly because the hidden knowledge of this problem is difficult to be represented using the usual 'if-and-then' rules. Our approach tends to evolve a large number of rules (average of 34) to represent the hidden fact of the problem. However, because of the minimum support setting used to limit the number of rules the system should hold, the rules evolved are not enough to represent the whole dataset. The best accuracy (75.46%) for this problem is achieved with 67 rules by setting the minimum support to 0. The results on *breast-cancer* dataset are particularly focused because relatively low accuracy had been achieved for this problem. This can be considered as a hard problem and should challenge any classification approach as most of the previous studies reported relatively high classification error. The default accuracy (classification without rule, which always classify instances as major class) is 67%. The average accuracy obtained here is 70.65% with 12 rules, and the best accuracy is 73.47%, as can be seen from Table 6. Although the accuracies are low, these results are comparable to the results of some previous studies (65 – 78%) obtained from the 'past usage' section in the dataset.

C. Performance Comparisons

The performance of the proposed GPc model was compared to other classification methods. The results are shown in Table 7 for the same nine datasets. The first seven classifiers, which are Decision Table, K-nearest neighborhood (IBk), ID3, J48, Naï ve Bayes, One rule (OneR) and multi-layer perceptron neural network with back-propagation (NN) are included in the WEKA package for comparison. The J48 is a classifier that generates an unpruned or a pruned C4.5 decision tree [15]. All these classifiers are applied with their default settings as provided in the WEKA.

TABLE 6 CLASSIFICATION ACCURACY RATES AND NUMBER OF RULES OBTAINED BY GPC

Dataset	Test instances	Correctly classified instances	Accuracy (%)			Run time (s)	Number of rules
			Best	Avg	Stddev		
Weather	14	14	100.00	98.57	1.01	26	5
Contact	24	22	100.00	92.08	5.60	30	7
Zoo	35	33	100.00	94.29	4.03	264	7
Breast-Cancer	98	69	73.47	70.65	1.99	116	12
Monk1	147	120	100.00	81.63	12.99	147	13
Monk2	147	87	75.46	58.85	11.99	89	34
Monk3	147	145	100.00	98.78	0.86	69	6
Mushroom	2713	2574	97.21	94.86	1.66	255	8
Nursery	4407	3687	86.79	83.67	2.21	1847	20

TABLE 7 PERFORMANCE COMPARISON OF GPC TO OTHER CLASSIFICATION METHODS

Dataset	Decision Table	IBk	ID3	J48	Naï ve Bayes	OneR	NN	GA	GbGP	GPc
Weather	85.71	**100**	**100**	**100**	92.86	71.43	**100**	-	-	**100**
Contact	87.5	**100**	**100**	91.67	95.83	70.83	**100**	-	-	**100**
Zoo	91.43	97.14	**100**	97.14	97.14	34.28	97.14	82.4	-	**100**
Breast-Cancer	**73.47**	69.39	mv	73.47	69.39	68.37	71.43	71.03	-	**73.47**
Monk1	**100**	97.96	87.07	**100**	77.55	77.55	**100**	-	**100**	**100**
Monk2	66.67	57.82	35.37	66.67	61.22	66.67	**100**	-	60	75.46
Monk3	**100**	**100**	**100**	**100**	98.64	84.35	**100**	-	95.4	**100**
Mushroom	**100**	**100**	mv	**100**	95.69	98.44	**100**	-	-	97.21
Nursery	94.35	96.60	96.39	96.19	91.04	70.37	**99.93**	-	-	86.79

mv – missing values error (the classifier is not able to process missing values)

The results from Genetic Algorithm (GA) and grammar-based genetic programming (GbGP) are obtained from [13] and [18], respectively. Table 7 does not include the complete results for the GA and GbGP as they were not provided in their respective publication. The ID3 has two "mv" because this algorithm was unable to handle dataset with missing values. The best classification accuracies for each dataset are highlighted in bold font. It can be seen that the OneR method performed well on 2-class problems. However, for dataset with larger number of class to be predicted, the classification accuracy dropped significantly, as can be seen for the *zoo* dataset. The NN classifier shows impressive results on all the datasets, but generally, the training time is much longer than all other methods and there is no comprehensible rules generated by the NN compared. The results for GPc shown in Table 7 are the best results obtained for each dataset. As can be seen, the proposed GP classifier performs comparatively well to other classification algorithms on the datasets tested.

V. CONCLUSIONS AND FUTURE WORK

This paper has presented a GP classifier that implements the technique of concept mapping with enhanced representation of booleanized attributes and token competition to evolve multiple rules per class in data mining. The GP classifier has been applied to several selected datasets from *UCI Machine Learning Repository*, which shows that GP with simple tree structure (contains only *AND* and *NOT* function) is sufficient to represent solutions for classification tasks. The proposed GP classifier also demonstrates its capability of discovering comprehensible rules from datasets without background knowledge on that dataset. This feature is particularly important in learning the preliminary information for further guidance in the data mining process. Despite a few low performances on certain datasets, the GP classifier has been demonstrated to be a very promising approach in data mining. Future work includes further validations of the GP classifier upon a variety of datasets and refinements to the proposed GP classifier for better classification performance and accuracy.

REFERENCES

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications.* San Francisco, CA: Morgan Kaufmann, 1998.

[2] M. Brameier and W. Banzhaf. A Comparison of Linear Genetic Programming Neural Networks in Medical Data Mining. *IEEE Transaction on Evolutionary Computation*, 5, pp.17-26, 2001.

[3] C. C Bojarczuk, H.S. Lopes, and A.A. Freitas. Genetic Programming for Knowledge Discovery in Chest-Pain Diagnosis. *IEEE Engineering in Medicine and Biology*, July/August, pp. 38-44, 2000.

[4] R. Cattral, F. Oppacher, and D. Deugo. Rule Acquisition with a Genetic Algorithm. In *Proceedings of the 1999 Congress on Evolutionary Computation*, 1, pp. 125-129, 1999.

[5] P. Clark, and T. Niblett. The CN2 induction algorithm. *Machine Learning Journal*, 3, pp.261-283, 1989.

[6] U. Fayyad. Data Mining and Knowledge Discovery in Databases: implications for Scientific Databases. In *Proceedings of the ninth International Conference on Scientific and Statistical Database Management*, pp. 2-11, 1997.

[7] M. V. Fidelis, H. S. Lopes, and A. A. Freitas. Discovering Comprehensible Classification Rules with a Genetic Algorithm. In *Proc. Of the 2000 Congress on Evolutionary Computation*, 1, pp. 805-810, 2000.

[8] A. A. Freitas. On Rule Interestingness Measures. Knowledge-Based Systems, 12, pp. 309-315, 1999.

[9] J. E. Hunt, and D. E. Cooke. Learning Using an Artificial Immune System. *Journal of Network and Computer Application*, 19, pp. 189-212. Academic Press, 1996.

[10] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal. Application of Genetic Programming for Multicategory Pattern Classification. *IEEE Transaction on Evolutionary Computation*, 4(3), pp. 242-258, 2000.

[11] J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press, 1992.

[12] H. S. Lopes, M. S. Coutihno, and W.C. Lima. An Evolutionary Approach to Simulate Cognitive Feedback Learning In Medical Domain. In E. Sanchez, T. Shibata and L.A. Zadeh (Eds.), *Genetic Algorithm and Fuzzy Logic Systems: Soft Computing Perspectives*, pp. 193-207. Singapore: World Scientific, 1997.

[13] A. Papagelis and D. Kalles. GATree: Genetically Evolved Decision Trees. In: *Proc. of the 12th IEEE International Conference on Tools with Artificial Intelligence*, pp. 203-206, 2000.

[14] A. R. Polo and M. Hasse. A Genetic Classifier Tool. In *Proceedings of the 20th International Conference of the Chilean Computer Science Society*, pp. 14-23, 2000.

[15] J. R. Quinlan. *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann, 1992.

[16] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* CA: Morgan Kaufmann Publishers, 1999.

[17] M. L. Wong, W. Lam, K. S. Leung, P.S. Ngan, and J.C.Y. Cheng. Discovering Knowledge From Medical Databases Using Evolutionary Algorithms. *IEEE Engineering in Medicine and Biology Magazine*, 19(4), pp. 45-55, 2000.

[18] M. L. Wong and K. S. Leung. *Data Mining Using Grammar Based Genetic Programming and Applications.* London: Kluwer Academic Publishers, 2000.