Soft Comput (2007) 11:1165–1172
DOI 10.1007/s00500-007-0159-x

FOCUS

Applying genetic programming technique in classification trees

Chan-Sheng Kuo · Tzung-Pei Hong · Chuen-Lung Chen

Published online: 29 March 2007

Abstract Classification problems are often encountered in many applications. In the past, classification trees were often generated by decision-tree methods and commonly used to solve classification problems. In this paper, we have proposed an algorithm based on genetic programming to search for an appropriate classification tree according to some criteria. The classification tree obtained can be transferred into a rule set, which can then be fed into a knowledge base to support decision making and facilitate daily operations. Two new genetic operators, elimination and merge, are designed in the proposed approach to remove redundancy and subsumption, thus producing more accurate and concise decision rules than that without using them. Experimental results from the credit card data also show the feasibility of the proposed algorithm.

Keywords Genetic programming · Classification tree · Knowledge base · Genetic operator

1 Introduction

Classification is one of the important issues for knowledge discovery and business decision-making. It has been used in different applications including financial prediction. A good example is the credit risk assessment (Chen et al. 2006; Chen 2002; Kiang 2003), in which a bank decides whether to approve proposals of customers by examining their feature values. Classification problems can generally be divided into two main tasks—finding classifiers and classifying unknown data. Finding classifiers is usually done by adopting appropriate learning techniques on training data sets. Data sets and their features are thus examined to derive one or several good classifiers satisfying some criteria. The derived classifiers can then classify data and predicts their classes.

Many different methods have been proposed to solve classification problems in the literature. They included decision trees, logistic regression, discriminant analysis, neural network, and genetic algorithms, among others (Chtioui et al. 1997; Cox 1970; Fisher 1936; Quinlan 1993; Wang et al. 1998a; Wang et al. 1998b). Decision-tree methods, such as ID3, CART, C4.5 and C5.0 (Quinlan 1986; Quinlan 1993; Quinlan 1997), can be used to generate classification trees and classification rules. They are very popular for classification due to its understandability.

In many enterprises, the unlimited growth of data inevitably leads to a situation in which accessing desired information from a database becomes difficult. Knowledge management has thus become an important topic in some practical applications. The intent of knowledge management is to emphasize knowledge flows and the process of creation, sharing, and disseminating knowledge in an organization (Alavi and Leidner 2001). Developing a successful knowledge management system usually needs the construction of a complete and consistent knowledge base. Only a few workers will, however, contribute their knowledge into the repositories since they usually consider it as an extra burden (Kwan and Balasubramanian 2003). Designing an effective method to automatically generate classification rules fed into

C.-S. Kuo (✉) · C.-L. Chen
Department of Management Information Systems,
National Chengchi University, 116, Taipei, Taiwan, ROC
e-mail: cskuo@nccu.edu.tw

C.-L. Chen
e-mail: chencl@mis.nccu.edu.tw

T.-P. Hong
Department of Electrical Engineering,
National University of Kaohsiung,
811, Kaohsiung, Taiwan, ROC
e-mail: tphong@nuk.edu.tw

knowledge bases thus plays a critical role in knowledge management. It can facilitate enterprises for decision making or for daily operations.

Genetic programming is an evolutionary approach and can be used to discover some useful classification rules in financial areas (Chen 2002; Chen and Kuo 2002). In this paper, we propose a learning algorithm based on genetic programming to automatically find an appropriate classification tree, which will be transferred into a rule set composed of some classification rules. The rule set can then be used to help the establishment of a knowledge base in a knowledge management system for a practical application. Two new genetic operators, elimination and merge, are also designed in the proposed approach. Experimental results on a data set about credit risk assessment also show the feasibility of the proposed approach.

The remaining parts of this paper are organized as follows. Genetic programming is first briefly reviewed in Sect. 2. A GP-based framework for classification is designed in Sect. 3. An algorithm based on GP to learn appropriate classification trees is proposed in Sect. 4. The details of the proposed algorithm are described in Sect. 5. Experiments to demonstrate the performance of the proposed algorithm are stated in Sect. 6. Conclusions and future works are given in Sect. 7.

2 Review of genetic programming

The method of genetic programming (GP) was firstly proposed by Koza (1992). It initially used some genetic operations as a basis for creating computer programs. Each computer program could be thought of as a potential solution to a problem. GP is an extension to genetic algorithms by using variable-length trees instead of fixed-sized chromosomes. An individual in GP is a tree structure consisting of functions and terminals suitable to the problem domain.

There are five major preparatory steps before applying GP to solving a problem (Koza 1992). They include the preparation of the following: (a) a set of terminals; (b) a set of functions; (c) a fitness measure; (d) parameters for controlling the process; (e) a method for terminating the process. The functions are selected from a possible pool of functions and the terminals are selected from a possible pool of terminals. There are several types of functions applied to GP, including arithmetic operations, mathematical functions, Boolean operations, conditional operations and any other user-specific functions.

For example in Fig. 1, the terminal set consists of variables (e.g. X, K and Y) and constants (e.g. 8 and 2) and the function set consists of some mathematic functions (e.g. +, * and −). The fitness values of individuals are calculated by an appropriate fitness function, with higher values representing better individuals in the population. The fitness function is usually designed corresponding to the problem to be solved. Control parameters generally include the population size, the probability of crossover and the probability of mutation. Termination criteria generally include a prespecified maximum number of generations and the total time of execution.

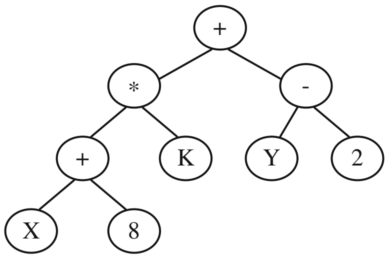

Fig. 1 A tree representation for the equation $(X + 8) * K + (Y - 2)$

There are three fundamental genetic operators, which are reproduction, crossover and mutation (Koza 1999). The reproduction operation selects a set of individuals from the population according to a selection method based on their fitness values. The individuals selected are reproduced into the new population in the next generation. Many selection methods have been proposed, which include fitness-proportionate reproduction, tournament selection and rank selection (Koza 1992). In this paper, the fitness-proportionate selection method is adopted. The crossover operation selects two individuals (parents) to operate and produces two new offspring. The mutation operation operates on single individuals to avoid local optimum. It usually replaces a selected subtree with another subtree.

The applications of GP to financial engineering cover various topics. Among them, financial forecasting and trading strategies are two of the most commonly seen research areas. For example, Nikolaev and Iba (2002) adopted the GP technique for learning polynomial autoregressive models, which were used to find trends in financial data series. Neely and his co-workers used GP as a search procedure to discover profitable trading rules in foreign exchange markets (Neely and Weller 1999, 2001; Neely et al. 1997). Other financial application areas of GP include option pricing, volatility modeling, and arbitrage, among others (Chen 2002; Chen and Kuo 2002). In addition to financial applications, genetic programming has also been applied to other application fields like economic models, symbolic regression, hand written digit recognition, and classification, etc (Chen 2002; Chen and Kuo 2002; Chien et al. 2002; Koza 1992; Parkins and Nandi 2004). This paper proposes a new learning algorithm based on genetic programming to automatically find an appropriate set of classification rules from a set of training examples for financial applications.

3 The GP-based learning framework

In this section, we state the GP-based framework used in this paper to generate a good final classification tree to be put in the knowledge base. The proposed framework is shown in Fig. 2.

In GP, each generation includes a population of individuals. The proposed framework in Fig. 2 thus maintains a population of classification trees with each representing a possible rule set. It then chooses suitable classification trees according to their fitness values to crossover or mutate, gradually creating good offspring trees. The framework constructs new classification trees by the proposed genetic operations and evaluates the trees according to a set of testing instances. The evolutionary process is repeated until the solutions converge. Finally, the best classification tree is transferred into a rule set, which can then be fed into a knowledge base to facilitate decision making or daily operations in some enterprises. Domain experts can thus reduce their efforts in building knowledge bases.

4 The proposed GP algorithm for classification

The proposed algorithm is stated in this section to show how a good classification tree can be found. Its flowchart is shown in Fig. 3.

It is an iterative method including several evolving steps. It chooses suitable trees for genetic operations, thus gradually creating good offspring. Two new genetic operators, elimination and merge, are also designed in the proposed algorithm to remove redundancy and subsumption. The offspring then undergoes repetitive evolution until the termination criterion is met. The algorithm is described as follows.

The proposed algorithm:

Input: a set of instances to be classified.

Output: an appropriate classification tree.

Step 1: Create an initial population of n randomly generated classification trees.

Step 2: Evaluate the fitness value of each classification tree by an evaluation function and the set of instances.

Step 3: Select suitable classification trees according to the evaluation results to perform the following genetic operations:

- Step 3.1: Perform crossover operations on parent trees to generate offspring trees;
- Step 3.2: Perform mutation operations on parent trees to generate offspring trees;
- Step 3.3: Perform elimination operations on offspring trees to remove redundancy;
- Step 3.4: Perform merge operations on offspring trees to remove subsumption.

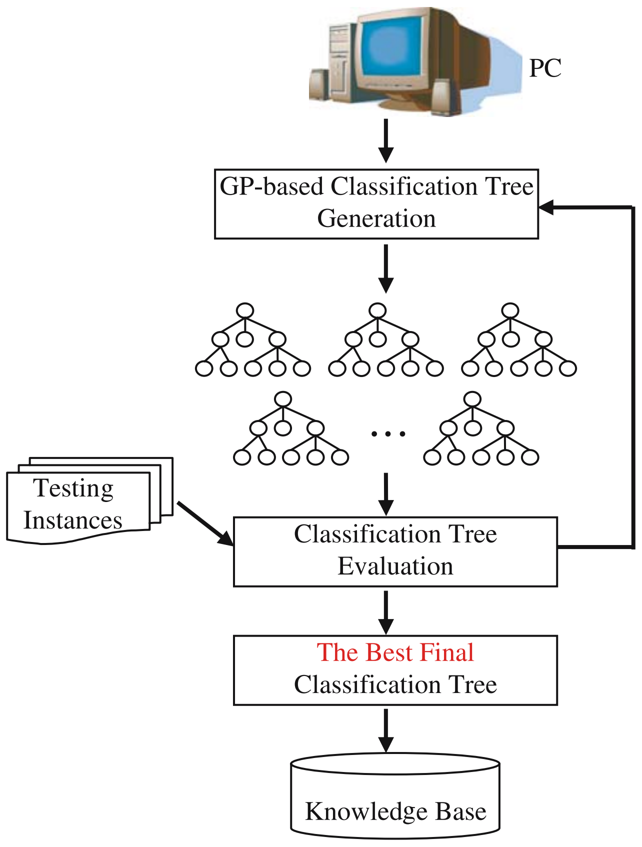
PC
GP-based Classification Tree Generation
Testing Instances
Classification Tree Evaluation
The Best Final Classification Tree
Knowledge Base

Fig. 2 GP-based framework for learning a good classification tree

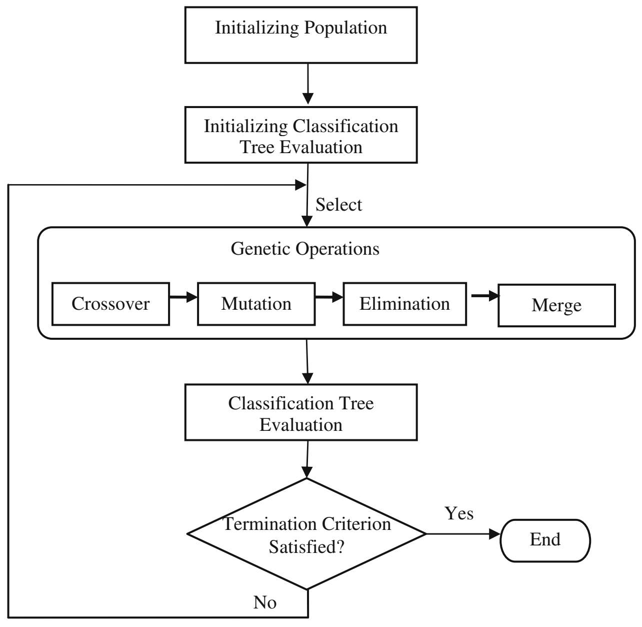
Initializing Population
Initializing Classification Tree Evaluation
Select
Genetic Operations
Crossover
Mutation
Elimination
Merge
Classification Tree Evaluation
Termination Criterion Satisfied?
Yes
End
No

Fig. 3 Flowchart of the proposed GP algorithm

Step 4: Evaluate the fitness value of each resulting offspring tree by the evaluation function and the set of instances.

Step 5: If the termination criterion is not satisfied, then go to Step 3; otherwise, do the next step.

Step 6: Select the best classification tree with the highest fitness value from the population as the final classification tree.

After Step 6, the final classification tree will then be transferred into a rule set, which can then be stored into a knowledge base. The two new genetic operators, elimination and merge, can reduce the tree complexity, but will take more execution time than using only the original operators. Trade-off thus exits between the tree concision and the time complexity. Some details about the execution of the algorithm are stated below.

5 Details of the algorithm

5.1 Initial population

Genetic programming requires a population of feasible solutions to be initialized and updated during the evolving process. Each individual within the population is a hierarchically structured tree consisting of functions and terminals. The functions used in this paper are as follows:

(a) Boolean operations: {And, Or, Not, $>$, $<$, $\geqq$, $\leqq$}, and
(b) Conditional operators: If-Then, If-Then-Else.

The initial population is randomly generated with some constraints for forming feasible classification trees. Below, a simple example is given to illustrate the concept.

Example 1 Assume in the application of applying for credit cards, two classes {disapproving, approving} represented as {R0, R1}, are to be distinguished by the three features {Job type, Education, Salary}. Also assume the feature of *Job type* has four possible values {laborer, low-level manager, middle-level manager, high-level manager} denoted {J0, J1, J2, J3}, the feature of *Education* has four possible values {High school, College, Master, Ph.D} denoted {E0, E1, E2, E3}, and the feature of *Salary* has four possible values {$<$200 thousand (NT dollars), 200$-$490 thousand, 500$-$800 thousand, >800 thousand} denoted {S0, S1, S2, S3}. To illustrate the tree representation more clearly, assume that a rule set RS_i only has the following two rules:

$Rule_1$: If (Job type = middle-level manager) and (Education = Master) and (Salary = 500 $-$ 800 thousand) then Class is approving;
$Rule_2$: If (Job type = laborer) and (Education = College) and (Salary = <200 thousand) then Class is disapproving.

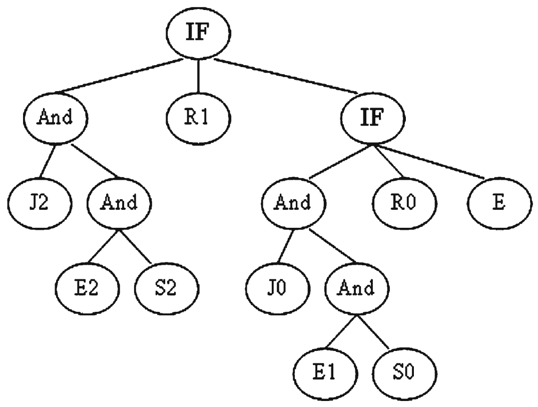

Fig. 4 A classification tree representation for the example

The two rules are equivalent to the following representation:

$Rule'_1$: If J2 and E2 and S2 then R1;
$Rule'_2$: If J0 and E1 and S0 then R0.

The two rules can be represented as a classification tree shown in Fig. 4. The internal nodes of the classification tree are labeled with operations or functions (e.g. And and IF); the leaf nodes are labeled with the feature values (e.g. J2, E2, S2, J0, E1, and S0) and the classes (e.g. R1 and R0). E represents the end of the rule set.

5.2 The fitness function

It is important to define a suitable fitness function for GP to work. It is used to reward good individuals or punish bad ones. In order to develop a good classification tree (CT) from an initial population, parent classification trees with high fitness values are selected for mating. An evaluation function is defined and a set of training instances are used to qualify the classification trees. The training instances are the cases actually occurring in the application and can be gathered from real environments. Two important factors are used in evaluating a classification tree. They are accuracy and complexity of the resulting knowledge structure (Wang et al. 1998b). In our approach, the accuracy for a classification tree (CT) is evaluated by the training instances and is defined as follows:

$$\text{Accuracy (CT)} = \frac{\text{the total number of training instances correctly matched by CT}}{\text{the total number of training instances}}, \tag{1}$$

The complexity of a classification tree is the ratio of nodes used, which is defined as follows:

$$\text{Complexity (CT)} = \frac{\text{number of nodes within CT}}{\left[\sum_{i=1}^{j} (\text{number of nodes within initial } CT_i)\right] \Big/ j}, \tag{2}$$

where j is the number of individuals in a population. Accuracy and complexity are combined to represent the fitness value of a classification tree. The evaluation function is thus defined as follows:

$$\text{Fitness (CT)} = \frac{[\text{Accuracy(CT)}]}{[\text{Complexity(CT)}]^{\alpha}}, \tag{3}$$

where α is a control parameter representing a tradeoff between accuracy and complexity. The fitness function can also reduce the impact of noisy information that causes classification-tree overfitting (excessive complexity).

5.3 Genetic operators

Genetic operators are very important to the success of genetic programming applications. In this paper, two fundamental genetic operators, crossover and mutation, and two new operators, elimination and merge, are used in the proposed algorithm. The elimination and merge operators are designed to solve the problems such as redundancy and subsumption (Giarratano and Riley 1993).

5.3.1 Crossover

In the crossover operation, two parent trees are partially exchanged to form two new offspring trees. The crossover point may occur within a rule or on rule boundaries. The crossover point must match up semantically. The detailed procedure is shown below.

(1) Select a crossover point in one of the parents at random.
(2) If the chosen point occurs at a rule boundary ("IF" node), the crossover point in the other parent tree must also be at a rule boundary. Otherwise, the point may be within a rule. The crossover point for the other parent must also be within a rule.
(3) Exchange the two chosen parent subtrees, which consist of the entire set of nodes below the crossover point, to generate two new offspring trees.

Example 2 Continuing from Example 1, assume that the two parent trees for crossover are Parent 1 and Parent 2 shown in Fig. 5. If a crossover operation with crossover points at CP1 and CP2 is executed, the two new offspring trees (Offspring1 and Offspring2) are shown in the lower part of Fig. 5.

5.3.2 Mutation

The mutation operation is desired to help the evolving process escape from local optimum. It begins by selecting a parent tree at random from the population. A node within the tree is randomly selected. The mutation operation then replaces the selected node (including its child nodes) with a randomly generated subtree. An example of the mutation operation using the mentioned procedure is given in Fig. 6.

5.3.3 Elimination

The elimination operation is used to solve the redundancy problem, in which two or more rules with the same feature

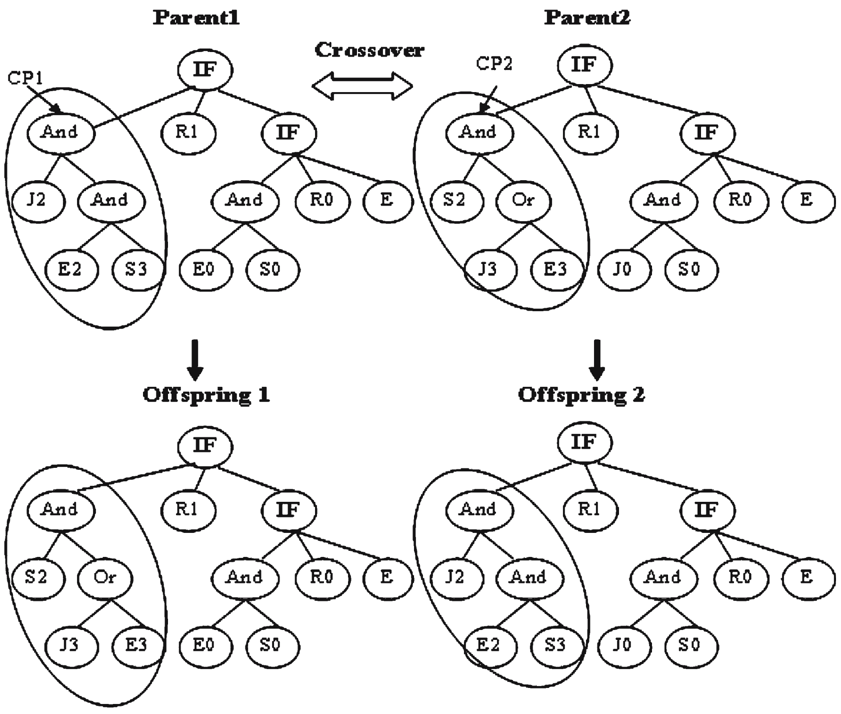

Fig. 5 An example of crossover

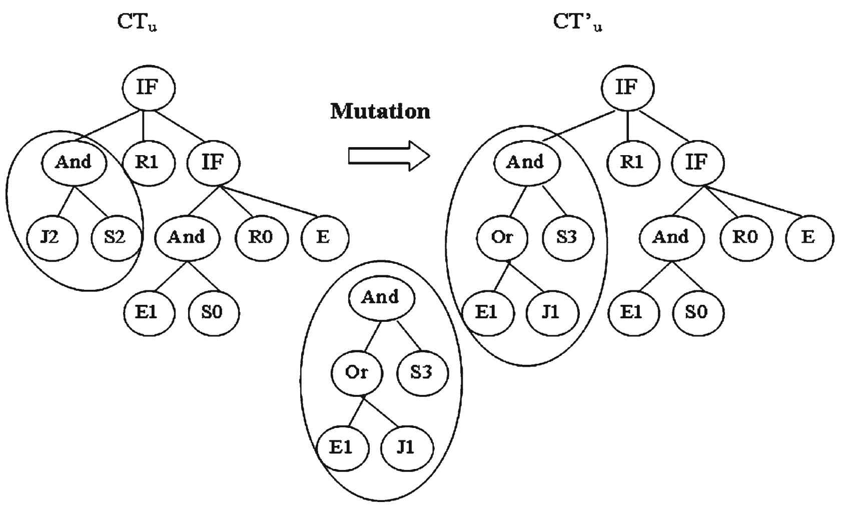

Fig. 6 An example of mutation

values and class appear in a classification tree. The elimination operation will remove redundant rules and can thus reduce the tree complexity. Let the classification tree to be processed be CT_r with k rules. The steps for the elimination operation are shown as follows.

(1) Set $i = 1$ and $j = i + 1$, where i and j are used to represent the numbers of the current two selected rules for checking.
(2) Compare $Rule_{ri}$ with $Rule_{rj}$ for their feature values and classes.
(3) If the two rules have the same values, they are redundant. Remove the deeper redundant rule from the classification tree CT_r.
(4) Set $j = j + 1$.
(5) If $j > k$, then do the next step; otherwise, go to step (2).
(6) Set $j = i + 2$ and $i = i + 1$.
(7) If $i = k$, then exit the searching process; otherwise, go to step (2).

Below, an example is given to illustrate how the elimination operation deals with redundancy.

Example 3 Assume the classification tree CT_r in Fig. 7 includes two redundant rules $Rule_{ri}$ and $Rule_{rj}$. To solve the redundancy problem, the elimination operation is used on the tree CT_r, with the deeper redundant rule $Rule_{rj}$ removed. The resulting classification tree CT'_r is shown in the right part of Fig. 7.

5.3.4 Merge

Rule subsumption means one rule subsumes the other in a classification tree. The subsumed rule may be removed from the tree to reduce the tree complexity. The merge operation is thus designed to solve the subsumption problem in a classification tree. It is a little like the elimination operation. Let the classification tree to be processed be CT_s with q rules. The steps for the merge operation are shown as follows.

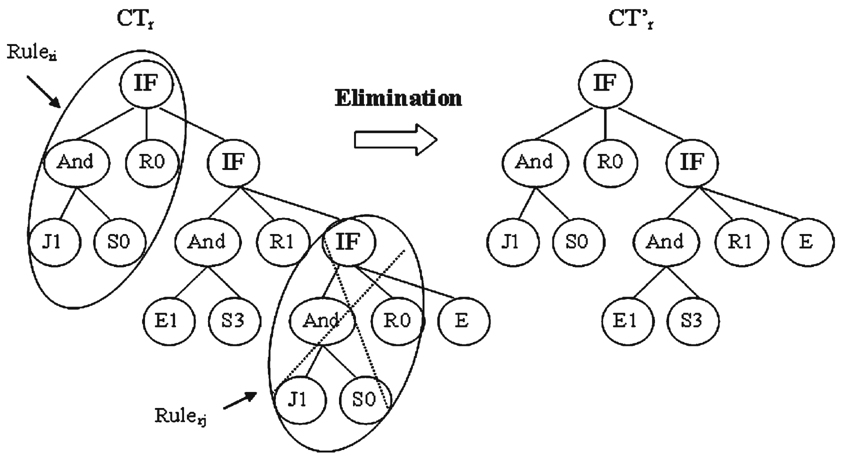

Fig. 7 An example of elimination

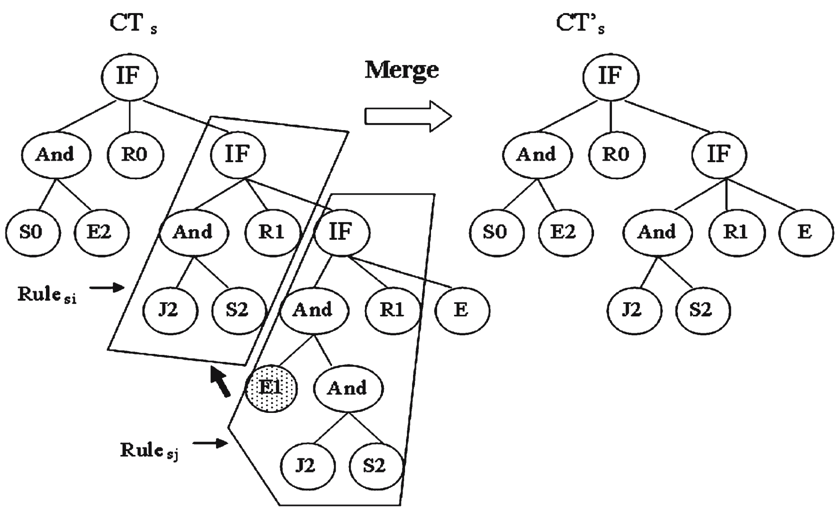

Fig. 8 An example of merge

(1) Set $i = 1$ and $j = i + 1$, where i and j are used to represent the numbers of the current two selected rules for checking.
(2) Compare $Rule_{si}$ with $Rule_{sj}$ for their feature values and classes.
(3) If the two rules have the same classes and the feature values of a rule is a subset of those in the other one, remove the rule with more features from the classification tree CT_s.
(4) Set $j = j + 1$.
(5) If $j > q$, then do the next step; otherwise, go to step (2).
(6) Set $j = i + 2$ and $i = i + 1$.
(7) If $i = q$, then exit the searching process; otherwise, go to step (2).

Below, an example is given to illustrate how the merge operation deals with subsumption.

Example 4 Assume in Fig. 8, $Rule_{si}$ and $Rule_{sj}$ are included in the classification tree CT_s. There is a subsumption relation between $Rule_{si}$ and $Rule_{sj}$. To solve the subsumption problem, the merge operator is used on the tree CT_s. $Rule_{sj}$ is thus removed from the tree since it has more feature values than $Rule_{si}$. The resulting classification tree CT_s' is shown in the right part of Fig. 8.

6 Experimental results

In this section, a set of data for credit-card application was used to test the performance of the proposed approach. There were 687 cases obtained from a bank in Taipei, Taiwan. The cases consisted of fourteen features and two classes (disapproving and approving). The goal was to find a good classification tree which could be converted into a set of rules to help identify one of the two classes for a credit-card applicant.

Table 1 The accuracy of the proposed method, traditional GP and C5.0

Method	Accuracy (%)
Our approach	78.63
Traditional GP	76.21
C5.0	75.39

The 687 cases were first divided into two groups, a training set and a test set. The training set was used to evaluate the fitness values of classification trees during the evolving process and the test set acted as input events to test the resulting rule set. In each run, 70% of the credit card cases were selected at random for training, and the remaining 30% of the cases were used for testing. The percentage of correct predictions was recorded.

In the experiments, the rates for crossover, mutation, elimination and merge operations were set at 0.9, 0.02, 0.8, and 0.8, respectively. The parameter α in the fitness function was set at 0.125. Table 1 compares the accuracy of the proposed approach with those by traditional GP (without elimination and merge operations) (Koza 1999) and by C5.0 (Quinlan 1997). The proposed approach obtained an accuracy rate of 78.63% after 500 generations. It can easily be seen that the proposed approach had a higher accuracy than traditional GP and C5.0.

7 Conclusions and future works

In this paper, we have proposed a learning algorithm based on genetic programming to search for an appropriate classification tree according to the criteria of accuracy and complexity. The proposed approach tries to find a good classification tree by genetic operators and improves its accuracy via the fitness function. Experimental results have also showed that the proposed approach could find a classification tree for credit-card application with a higher accuracy than those by traditional GP and C5.0. The final classification tree obtained can be transferred into a rule set, which can be further fed into a knowledge base to support decision-making processes and to facilitate daily operations.

The proposed approach has also designed two new genetic operators, elimination and merge, to take rule relationship into consideration, thus reducing the tree complexity and making the results closer to those desired. This, however, takes more execution time than using only the original operators. In the future, we will study the choice of fitness functions in order to further raise the accuracy. Some fuzzy concepts may also be adopted to deal with fuzzy knowledge.

References

Alavi M, Leidner DE (2001) Review: knowledge management and knowledge management systems: conceptual foundations and research issues. MIS Q 25(1):107–136

Chen G, Liu H, Yu L, Wei Q, Zhang X (2006) A new approach to classification based on association rule mining. Decis Support Syst 42:674–689

Chen SH (2002) Genetic algorithms and genetic programming in computational finance. Kluwer, Dordrecht

Chen SH, Kuo TW (2002) Evolutionary computation in economics and finance: a bibliography. Evolutionary computation in economics and finance. Physica-Verlag, Heidelberg, New York, pp 419–455

Chien BC, Lin JY, Hong TP (2002) Learning discriminant functions with fuzzy attributes for classification using genetic programming. Expert Syst Appl 23:31–31

Chtioui Y, Bertrand D, Devaux M, Barba D (1997) Comparison of multilayer perceptron and probabilistic neural networks in artificial vision application to the discrimination of seeds. J Chemom 11:111–129

Cox DR (1970) The analysis of binary data. Chapman & Hall, London

Fisher RA (1936) The use of multiple measurements in taxonomic problems. Ann Eugen 7:179–188

Giarratano J, Riley G (1993) Expert system principles and programming. PWS, Boston

Kiang MY (2003) A comparative assessment of classification methods. Decis Support Syst 35:441–454

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

Koza JR (1999) Genetic programming III: Darwinian invention and problem solving, Morgan Kaufmann Publishers, San Mateo

Kwan MM, Balasubramanian P (2003) KnowledgeScope: managing knowledge in context. Decis Support Syst 35:467–486

Neely CJ, Weller PA (1999) Technical trading rules in the European monetary system. J Int Money Finance 18:429–458

Neely CJ, Weller PA (2001) Technical analysis and Central Bank intervention. J Int Money Finance 20:949–970

Neely CJ, Weller PA, Dittmar, R (1997) Is technical analysis in foreign exchange market profitable? A genetic programming approach. J Financ Quant Anal 32(4):405–426

Nikolaev N, Iba H (2002) Genetic programming of polynomial models for financial forecasting. Genetic algorithms and genetic programming in computational finance. Kluwer, Dordrecht, pp 103–123

Parkins AD, Nandi AK (2004) Genetic programming techniques for hand written digit recognition. Signal Processing 84:2345–2365

Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106

Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers, San Mateo

Quinlan JR (1997) C5.0 and see 5: illustrative examples. RuleQuest Res. http://www.rulequest.com

Wang CH, Hong TP, Tseng SS (1998a) Integrating fuzzy knowledge by genetic algorithms. IEEE Trans Evol Comput 2(4):138–149

Wang CH, Hong TP, Tseng SS, Liao CM (1998b) Automatically integrating multiple rule sets in a distributed-knowledge environment. IEEE Trans Syst Man Cybern 28(3):471–476