Taylor & Francis
Taylor & Francis Group

Fuzzy Information and Engineering

ISSN: 1616-8658 (Print) 1616-8666 (Online) Journal homepage: https://www.tandfonline.com/loi/tfie20

Evolution of Fuzzy Classifiers Using Genetic Programming

Durga Prasad Muni & Nikhil R. Pal

To cite this article: Durga Prasad Muni & Nikhil R. Pal (2012) Evolution of Fuzzy Classifiers Using Genetic Programming, Fuzzy Information and Engineering, 4:1, 29-49, DOI: 10.1007/s12543-012-0099-8

To link to this article: https://doi.org/10.1007/s12543-012-0099-8

© 2012 Taylor and Francis Group, LLC

Published online: 25 Jan 2019.

Submit your article to this journal

Article views: 14

View related articles

Citing articles: 1 View citing articles

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=tfie20

Fuzzy Inf. Eng. (2012) 1: 29-49
DOI 10.1007/s12543-012-0099-8

ORIGINAL ARTICLE

Evolution of Fuzzy Classifiers Using Genetic Programming

Durga Prasad Muni · Nikhil R. Pal

Received: 20 September 2011/ Revised: 20 January 2012/
Accepted: 15 February 2012/
© Springer-Verlag Berlin Heidelberg and Fuzzy Information and Engineering Branch of the Operations Research Society of China
2012

Abstract In this paper, we propose a genetic programming (GP) based approach to evolve fuzzy rule based classifiers. For a c-class problem, a classifier consists of c trees. Each tree, T_i, of the multi-tree classifier represents a set of rules for class i. During the evolutionary process, the inaccurate/inactive rules of the initial set of rules are removed by a cleaning scheme. This allows good rules to sustain and that eventually determines the number of rules. In the beginning, our GP scheme uses a randomly selected subset of features and then evolves the features to be used in each rule. The initial rules are constructed using prototypes, which are generated randomly as well as by the fuzzy k-means (FKM) algorithm. Besides, experiments are conducted in three different ways: Using only randomly generated rules, using a mixture of randomly generated rules and FKM prototype based rules, and with exclusively FKM prototype based rules. The performance of the classifiers is comparable irrespective of the type of initial rules. This emphasizes the novelty of the proposed evolutionary scheme. In this context, we propose a new mutation operation to alter the rule parameters. The GP scheme optimizes the structure of rules as well as the parameters involved. The method is validated on six benchmark data sets and the performance of the proposed scheme is found to be satisfactory.

Keywords Genetic programming · Fuzzy logic · Classification · Rule extraction · Evolutionary algorithms

1. Introduction

Evolutionary algorithms (EAs) [1-3] have been used to solve numerous searching and optimization problems. In EAs, usually a population of possible solutions to a given problem is evolved using the Darwinian principle of survival of the fittest.

Durga Prasad Muni (✉)
Infosys Limited, Bangalore, India
email: DurgaPrasad_Muni@infosys.com
Nikhil R. Pal
Indian Statistical Institute, Calcutta, India
email: nikhil@isical.ac.in

Genetic algorithms (GAs) [1] are the most popular EAs. Genetic programming (GP) [2] is a different type of EA where each solution is typically represented by a tree or a program. And hence, GP is usually used to find a required model (or a function) to solve a given problem. In a classifier design problem [4], each solution of the GP population is a possible classifier for the given classification problem. GP evolves a population of possible classifiers and provides better classifiers and their (mathematical) expressions [11-13]. So, the obtained classifiers can not only be directly employed to classify new samples, but also the expression of the classifiers (models) can be analyzed to understand the classifiers and the hidden relation in the data.

While designing models, we may incorporate fuzzy logic concepts to address vagueness and uncertainty in data. In the literature, several works are available, where artificial neural networks (ANNs) are combined with fuzzy logic to design models (e.g. classifiers) [5-9]. Pedrycz [10] introduced a 3-layer heterogeneous neural network using logic-based neurons to realize matching and aggregation. The neurons in the hidden layer detect individual decision regions in the feature space. Outputs from these neurons are then aggregated to make the final decision. All computations in this network are realized by logic operations.

Many authors have used genetic algorithms to design fuzzy rules [14,17-23] including fuzzy classifier rules [15,16]. Ishibuchi [24] presented a review on the use of multi-objective evolutionary algorithms to design fuzzy rule based systems. Using a set of benchmark data sets, the author has demonstrated the advantages of the multi-objective approaches over the single-objective methods.

A few attempts have also been made to evolve fuzzy classifier rules by using GP. Evolution (or design) of fuzzy rules using a technique like GP does not need the help of domain experts. In fuzzy classification, a classifier constitutes a set of fuzzy rules (or a rule base).

Tsakonas has employed GP in [25] to develop four different types of classifiers. The author has used four context-free grammars to describe decision trees, fuzzy rule-based systems, feed forward neural networks and fuzzy Petri-nets with GP. To evolve fuzzy rule-based classifiers using GP, the author has initialized a population of rule bases where each rule base represents a possible classifier. In [26], GP has been used to find suitable structure of fuzzy neural networks for classification. After developing the structure, authors have optimized parameters using gradient-based technique. GP operators have been incorporated with simulated annealing to obtain fuzzy rule based classifiers in [27]. In [28], a co-evolutionary approach has been used to discover fuzzy classification rules. A GP population involving fuzzy rules and a simple evolutionary algorithm involving membership function definitions are co-evolved to seek a well adapted fuzzy rule set and a set of membership function definitions. GP has been used in [29] for the generation of fuzzy rule bases in classification and diagnosis of medical data. In [30], authors have proposed a fuzzy rule-based classification system using GP. They followed the cooperative-competitive learning approach. Edmonds et al [31] used GP to generate fuzzy logic production rules. Chien et al [32] used GP to evolve discriminant functions as classifiers. The attributes were transformed into fuzzy attributes and then the classifiers were developed based on these fuzzy attributes.

A solution (individual) of the GP population can represent an individual fuzzy rule or a rule base (the classifier). In the former case, the individuals (rules) are evolved in a co-operative manner and the rules of the population act together as a classifier. In the latter, each individual is a rule base representing a classifier. These rule bases or classifiers are evolved in a competitive manner and the best classifier(s) is considered the final desired classifier. Here we have adopted the latter approach.

The performance of a fuzzy classifier depends on both its structure and parameters. In many approaches, the structure is assumed a-priori. GP has the ability not only to find the structure of the model (fuzzy classifier), but also the parameters it involves.

In this study, we consider fuzzy classifier rules [33] of the form:

R_i: **If** x_1 is A_{i1} **AND** x_2 is A_{i2} **AND**$\cdots$**AND** x_p is A_{ip} **then class is** j. Here A_{ij} is a fuzzy set used in the i-th rule and defined on the domain of attribute x_j and p is the number of features.

For classification of a pattern $\mathbf{x} \in \mathcal{R}^p$, the antecedents (if-clauses) of the fuzzy rules are evaluated using some T-norm. The degree to which the antecedent of a rule is satisfied is called the firing strength of the rule. These firing strengths are used to determine the class membership of $\mathbf{x}$. Usually, the class corresponding to the rule with the largest firing strength is assigned to $\mathbf{x}$.

Generally, all features are used to design a fuzzy rule. However, a rule may not need all features. In fact, some features may be redundant/irrelevant and some may even be derogatory. So a rule may be created by using a subset of features.

We have used GP to evolve fuzzy classifier rules for multi-class problems. Our GP methodology generates the required fuzzy classifier by determining: (i) number of rules, (ii) features(clauses) used in a rule, and (iii) values of the parameters involved. For a c class problem, an individual/classifier C_l consists of c trees $\{T^l{}_1, T^l{}_2, \cdots, T^l{}_c\}$. Each tree $T_k{}^l$ contains a set of $b_k{}^l$ rules representing class k.

2. Procedure

The fuzzy sets A_{ik} of a fuzzy rule can be determined based on prototypes [33]. A prototype represents a cluster of patterns. If $\mathbf{m}_i \in \mathcal{R}^p$ represents a prototype and if $\mathbf{x} \in \mathcal{R}^p$ represents an arbitrary pattern then a fuzzy rule can be written as:

R_i: If $\mathbf{x}$ is CLOSE TO $\mathbf{m}_i$ then class is k.

This rule can further be written in a more understandable form as below:

R_i: If x_1 is CLOSE TO m_{i1} AND $\cdots$ AND x_p is CLOSE TO m_{ip} then class is k.

Note that, the two forms of the rules are not necessarily the same. We can model the fuzzy set "CLOSE TO m_{ij}" using different functions such as triangular, trapezoidal and Gaussian membership functions. Here, we have used Gaussian membership function to represent a fuzzy set. The Gaussian membership function is given by:

$$\mu_{CLOSETO}(x_j, m_{ij}) = e^{-[x_j - m_{ij}]^2/\sigma_{ij}{}^2}. \tag{1}$$

In (1), m_{ij} is the j_{th} component of prototype $\mathbf{m}_i$ and σ_{ij} is the spread of the membership function. Since the membership values depend on the values of m_{ij} and σ_{ij}, it is necessary to find appropriate values of these parameters. At first, we take some initial values for these parameters to initialize the population of classifiers. The evolutionary process then evolves the classifiers along with the parameters and provides us with useful classifiers.

To compute firing strength of the rule, in this investigation, the MIN operator is used as the T-norm. So, the minimum of all membership values in a rule gives the firing strength of the rule.

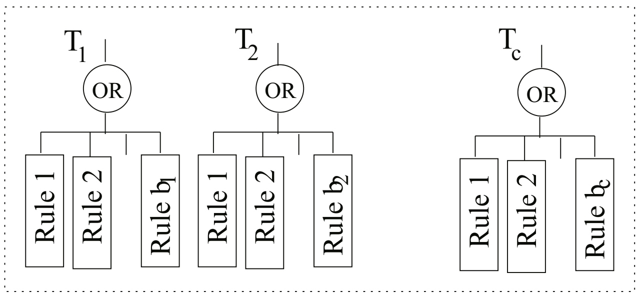

Fig. 1 Representation of a fuzzy classifier

As mentioned earlier, in our GP methodology, for a c-class problem, a classifier consists of c trees. Fig.1 represents a classifier. Each tree T_k has an "OR" node as the root and a set of b_k rules, $\{R_i{}^k\}, i = 1, 2, \cdots, b_k$ as children to the root. The set of rules $\{R_i{}^k\}$ of tree T_k represents the class k. So, we do not include consequent part (then-clause) in the rules.

A typical rule $R_i{}^k$ (by using features x_1, x_3, x_4 and x_5) may be written as:

$R_i{}^k$: x_1 is CLOSE TO m_{i1} AND x_3 is CLOSE TO m_{i3} AND x_4 is CLOSE TO m_{i4} AND x_5 is CLOSE TO m_{i5}.

The fuzzy set CLOSE TO m_{ij} is defined by a Gaussian function with two parameters m_{ij} and σ_{ij}.

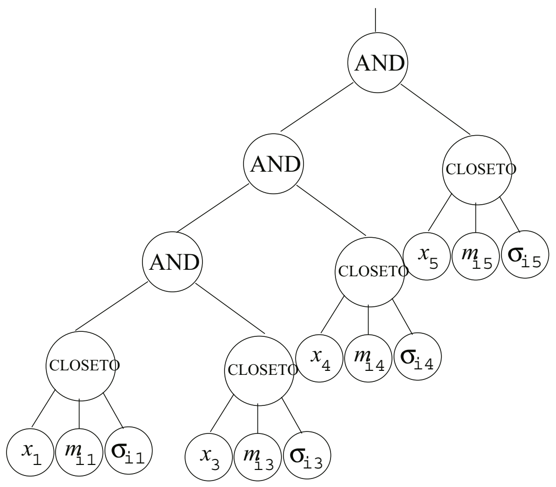

Fig. 2 A typical tree representation of fuzzy rule

In our GP approach, the above rule is represented as a tree as shown in Fig.2. The root node of a rule ${R_i}^k$ is an "AND" node. The right children of an "AND" node is a "CLOSE TO" node and the left children is either an "AND" node or a "CLOSE TO" node. A CLOSE TO node has three children: a feature x_j, m_{ij} and σ_{ij}.

The procedure for evolution of the GP-fuzzy classifier is discussed in the following sections.

2.1. Initialization

To start the evolution, at first a population of P classifiers is randomly generated. For a c class classification problem, a classifier C^l consists of c trees. To create a tree ${T_k}^l$ an "OR" node with b_k-arity (children) is taken. Each children of the "OR" node is a rule ${R_i}^k, i \in \{1, 2, \cdots, b_k\}$. We assume a range for the initial height of the trees. To determine the height of the rule ${R_i}^k$, a randomly selected value d_i in the range of initial height of the trees is taken. Then the tree structure of the rule ${R_i}^k$ is generated with height $d_i - 1$ (excluding the OR node) using a randomly chosen feature subset, S_i, of size $d_i - 2$ and ($\mathbf{m_i}$, $\sigma_\mathbf{i}$).

We produce the prototypes ($\mathbf{m_i}, \sigma_\mathbf{i}$) either randomly or by using a clustering algorithm; here we use the popular fuzzy K-means (FKM) clustering algorithm [34]. Both procedures are given below.

2.1.1. Random Prototypes

We randomly choose a value in the domain of the j^{th} feature as the value of μ_{ij}, i.e., $\mu_{ij} \in [{f^j}_{min}, {f^j}_{max}]$. ${f^j}_{min}$ and ${f^j}_{max}$ are the minimum and maximum values of the j^{th} feature in the training set. To obtain a value for σ_{ij}, we randomly take a value $r \in [\beta_1, \beta_2]$. Then we multiply r by the range of the j^{th} feature, W_j=${f^j}_{max}$-${f^j}_{min}$. Instead of taking $\beta_1 = 0$, we have taken a small value $\beta_1 = 10^{-4}$. This prevents generation of almost zero value of σ_{ij}. Similarly, we have taken $\beta_2 = 0.25$ that prevents from producing a large σ_{ij}. Note that, our goal is to represent a cluster of points by a rule so that membership values of patterns near to the prototype are high and away from the prototype are low. But if we take a very large value of σ_{ij}, then the Gaussian function will cover a large area and the difference between membership values of points near to and far from the prototype may not be strong enough to discriminate.

2.1.2. Fuzzy K-means Based Prototypes

We run fuzzy K-means algorithm to find K prototypes from the training data for each class. Then each training pattern is assigned to the prototype for which it has the highest membership value. It may happen that no pattern is assigned to a prototype. In this case, the prototype is discarded. This provides $g_h \leq K$ prototypes for a particular class h. For the i_{th} prototype, the mean vector ($\mathbf{m}_i$) and the vector of standard deviations ($\sigma_\mathbf{i}$) of the patterns that are associated with the prototype are used to initialize a rule. The j_{th} component of σ_i is computed as the standard deviation of the j_{th} component of all training patterns that are associated with the i_{th} prototype.

The type of the prototype to be used in a rule ${R_i}^k$ is decided probabilistically. The probability to select a random prototype is p_{rp} and the probability to select an FKM prototype is $1 - p_{rp}$. The structure as given in Fig.2 is maintained while generating the rule. The CLOSE TO nodes, each having a feature x_j from the chosen feature subset S_i and the corresponding parameters (m_{ij}, σ_{ij}), are used to generate the rule in the descending order of index j from the right top most to the left bottom (as shown in Fig.2). This makes it easy to maintain the validity of rules during the genetic operations.

In the example shown in Fig.2, the chosen feature subset is $\{x_1, x_3, x_4, x_5\}$. The right top most children (to the root node AND) is a CLOSE TO node having x_5 and prototype (m_{i5}, σ_{i5}) as children. The left children to the root is an AND node. Again the right children to this AND node is the CLOSE TO node having the next feature (in decreasing order of index) x_4 of the feature subset and the corresponding prototype (m_{i4}, σ_{i4}). The left children to this second AND node is also an AND node. This process is repeated till all but one feature is left in the feature subset. Now instead of taking an AND node, a CLOSE TO node is created using the last feature (having the lowest index) of the chosen feature subset. Here the last feature is x_1.

2.2. Fitness Measure

To assess how good the solutions are, each solution is evaluated and a fitness value is assigned to it. We evaluate a classifier (solution), C^l, of the GP population by using it to classify all training samples. For a training sample $\mathbf{x}$, the firing strength of each rule $R_i{}^k$ of tree $T_k{}^l$ is computed. The maximum firing strength is taken as the output $(\mu_k{}^l)$ of the tree $T_k{}^l$. If output $\mu_g{}^l$ is the highest among outputs $\{\mu_k{}^l\}$ of all trees and $\mathbf{x} \in$ *class* g, then we say that $\mathbf{x}$ is correctly classified by the classifier C^l. The number of correctly classified training samples $n^l{}_{rc}$ by the classifier C^l is computed. If N_{tr} is the total number of training samples, then the fitness value of the classifier is defined as

$$f^l = \frac{n^l{}_{rc}}{N_{tr}}. \tag{2}$$

2.3. Reproduction

Reproduction copies a few good solutions of the GP population to the next generation. This allows us to retain a few good solutions and prevents the population from losing them in the variation process (e.g. crossover, mutation).

2.4. Crossover

The good solutions are recombined to produce new solutions. It is expected that the new solutions will drive toward better and desired solutions. This recombination or crossover operation plays a vital role in the evolutionary process. For the crossover operation, we select two classifiers C^{p_1} and C^{p_2} from the GP population with probability of selection proportional to fitness. A tree $T_g{}^{p_1}$ of the first parent is randomly selected. Then an AND node of this tree $T_g{}^{p_1}$ is randomly chosen. After that, an AND node of $T_g{}^{p_2}$ of the second parent is randomly chosen in such a way that after swapping the subtrees with these AND nodes as root nodes will produce valid rules. Now the subtrees under these two AND nodes are swapped. In addition to this, we swap trees ${T_k}^{p_1}$ of C^{p_1} with ${T_k}^{p_2}$ of C^{p_2} for all $k > g$. This above-mentioned crossover operation is repeated with probability p_c. Note that, such a crossover produces one new rule in each of the two classifiers involved and also swaps two sets of rules between the two classifiers. Figures 3-6 illustrate the crossover operation. In the example, a subtree of rule 2 in tree T_1 of the first parent is selected for swapping with a subtree of rule 1 in tree T_1 of the second parent. After swapping subtrees, the trees following tree T_1 of both parents are swapped.

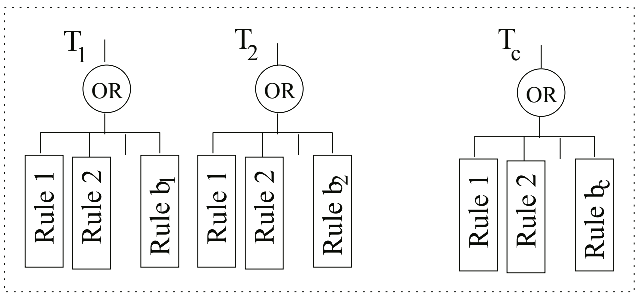

Fig. 3 Parent 1 for crossover

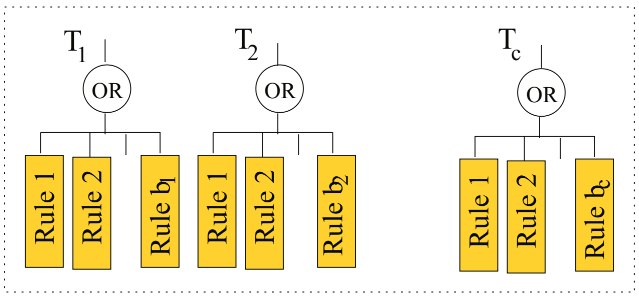

Fig. 4 Parent 2 for crossover

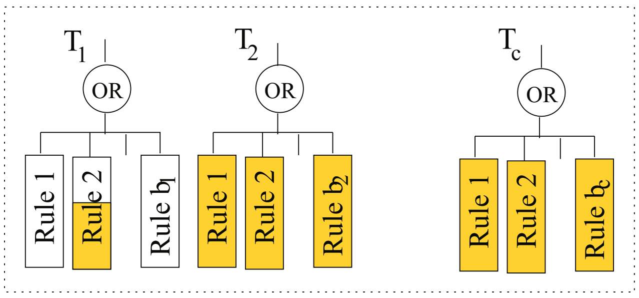

Fig. 5 Offspring 1 after crossover

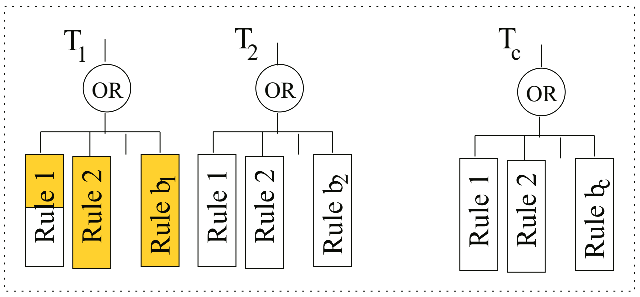

Fig. 6 Offspring 2 after crossover

2.5. Mutation

Mutation is a process to alter a single solution. To obtain the desired model (fuzzy classifier), we need to alter both the structure and parameters. To facilitate this, we use two mutation operations: *Sub-tree* mutation and *parameter* mutation. At first a classifier of the GP population is randomly chosen. Then a tree of the classifier is randomly selected for the mutation operation. After that, the type of mutation operation is decided with probability. Both mutation operations are given below:

1) *Sub-tree Mutation*: This alters the structure including the parameters of a rule. This subtree mutation is the standard GP mutation operation. First, we have tried the standard GP mutation operation by randomly selecting a subtree for mutation. However, we have observed that it is no more effective than mutation restricted to choosing only subtrees with AND node as the root. So, instead of randomly selecting a subtree, we select a subtree with an AND node as root for mutation. After selecting a subtree with an AND node as the root, we find the lowest index, L_I and the highest index H_I of the features used by the subtree. Then a valid rule is randomly generated using a randomly selected subset of the features $\{x_{L_I}, x_{L_i+1}, \cdots, x_{H_I}\}$. The chosen subtree of the tree is replaced with the generated rule. For example, in Fig.2, if the left child of the root node is selected for sub-tree mutation, then $L_I = 1$ and $H_I = 4$. So a randomly generated subtree using a subset of $\{x_1, x_2, x_3, x_4\}$ is used to replace the subtree rooted at the left child of the root node.

2) *Parameter Mutation*: We have introduced this mutation operation to alter values of parameters (m_{ij}, σ_{ij}). We randomly select a rule (of the selected tree) for the mutation. Then value of each (m_{ij} and σ_{ij}) of this rule is altered as follows:

Variation of m_{ij}: Let a_j and b_j be the smallest and highest values of the j_{th} feature, respectively. Then, define $W_j = b_j - a_j$. Now we generate a random value $\delta \in [0, W_j/100]$ and another random value $r \in [0, 1]$. If $r < 0.5$, then we add δ to m_{ij}. Otherwise, we subtract δ from it. We do not accept the variation if the altered value falls outside the domain of the feature.

Variation of σ_{ij}: A random value $\rho \in [0, 0.1 \times \sigma_{ij}]$ is generated. Now another random value $r \in [0, 1]$ is generated. If $r < 0.5$, then we add ρ to σ_{ij}. Otherwise, we subtract ρ from it. However, if the altered value is very small ($< W_j * 10^{-3}$), then we do not accept the variation.

We do sub-tree mutation with probability 0.8 and parameter mutation with probability 0.2. Before choosing these values, we have experimented on three sets of values: (0.9, 0.1), (0.8, 0.2) and (0.7, 0.3). Out of these three cases, the performance is the best for (0.8, 0.2) case.

After mutating the classifier, we evaluate it. If the fitness of the mutated classifier is higher or equal to that of original classifier, then we accept the mutation. Otherwise, we accept the mutated classifier with probability 0.5.

After every generation, we expect the new best classifier to have a higher fitness than that of previous best classifier. But sometimes the increase in fitness may be negligible or zero. If the variation is not sufficient, then we need to give sufficient perturbation to the GP population. This is accomplished by using mutation operation with a higher probability. Mutation may improve, degrade or may not even affect the performance. Since we use a directed mutation operation which accepts poor mutated solution with a probability, the chance of obtaining poor solutions is comparatively low. Thus, if the improvement in fitness over a set of successive generations is not satisfactory, we increase the probability of mutation. To do this, we maintain the so far best classifier and proceed as follows. Let B_t be the best classifier evolved in generation t with fitness f_{B_t} and SB_{t-1} is the so far best classifier with fitness $f_{SB_{t-1}}$ up to generation $t-1$. Now, if $f_{B_t} > f_{SB_{t-1}}$, then $SB_t \leftarrow B_t$; if $f_{B_t} = f_{SB_{t-1}}$ and the total number of nodes in B_t is smaller than that in SB_{t-1}, then $SB_t \leftarrow B_t$. Otherwise, $SB_t \leftarrow SB_{t-1}$. In this way, we maintain the so-far best classifier. Now, we sum the absolute difference in fitness of the so far best classifiers in 10 consecutive generations. If this sum is less than a predefined value, d_{th}, then we increase the probability of mutation operation p_m by γ_m. Now the question arises, what would be the values of these parameters?

Crossover operation plays a vital role to transfer good characteristics of the current individuals to the next generation by combining two or more good solutions. Hence, a very high probability (say, 0.8) is given to crossover operation and low probability to mutation and reproduction operations. Let p_c, p_m and p_r be the probabilities of crossover, mutation and reproduction operations respectively. We have taken $p_c = 0.75$, $p_m = 0.15$ and $p_r = 0.1$ at the beginning. However, as mentioned above, we increase p_m during the evolutionary process when required. In our experiments, we have observed that the initial fitness value (${f^b}_{ini}$) of the best classifier for different data sets varied from about 0.5 to 0.98. Thus, on average, we assume ${f^b}_{ini} = 0.75$. We need to continue the evolution till a classifier could classify all training samples (${f^b}_{fin} = 1$) or a pre-defined number of generations, generation = 50 (= M), is reached. This means that if ${f^b}_{ini} = 0.75$ and ${f^b}_{fin} = 1$, then we have to increase the fitness (of the best classifier) by 0.25 in at most 50 generations. This indicates that in every 10 generations, the increment in fitness value of the best classifier is expected to be $\delta f^b = 0.05$. So, we use $d_{th} = 0.05$. The increment in p_m should be small such that at the end of the evolution, p_m is not very high. Let us assume the maximum allowed value of p_m is 0.25. In other words, we may increase p_m at most by 0.1 in 50 generations. If we divide this increment over five steps (10 generations each), then the increment for each step is 0.02. Hence we use $\gamma_m = 0.02$. These choices seem plausible, but not necessarily the best choices. The most desirable values for these parameters may be chosen using a validation data.

2.6. Cleaning of Poor Rules

All rules of a tree may not be useful. Some of them may be poor/inaccurate and some may be inactive (not used in decision making). Inaccurate rules are primarily responsible for misclassification. By keeping only few good rules, we can give more chance to these good rules to participate in genetic operations. This increases the chance of obtaining better rules with fewer generations. Removing poor rules not only can improve the performance, but also can reduce computational time to a great extent. Hence, during evolution, we remove the inaccurate and inactive fuzzy rules. In this experiment, we evolve GP up to 50 generations. Note that, an attempt to remove poor rules at an early (premature) stage of evolution may have a derogatory effect on the final rule base. Hence, the first round of cleaning is done if and when the average fitness of the population reaches 0.7 (i.e., 70 % accuracy) during the first 25 generations. Then at gen=25, we again apply our rule cleaning scheme. After gen=25, we allow the population to evolve without any further pruning.

To decide on the rules to be removed, we evaluate each rule R_i^k and assign a fitness value h_{ik} to it as described below:

If rule R_i^k has the highest firing strength among all rules of tree T_k for a training sample $\mathbf{x}$, then we increase h_{ik} by 1 if $\mathbf{x} \in$ class k; otherwise, decrease h_{ik} by 1 if $\mathbf{x} \notin$ class k. If $h_{ik} > 0$, then we keep the rule R_i^k; otherwise we remove it. In this pruning scheme, if a rule makes more misclassification than correct classification, we remove it. However, we make sure that pruning process will retain at least two rules for every class.

2.7. Termination of GP and Validation

The GP is terminated when all N training samples are classified correctly by a classifier of the GP population or a predefined number, M, of generations are completed. The best classifier of the population is the required classifier, CF. The classifier CF is validated by allowing it to classify test samples. The test accuracies are discussed in the following section.

3. Experimental Results

We have used six benchmark data sets to validate our methodology. In addition to the well-known Iris [35] data set, four data sets are taken from the UCI Machine Learning Repository [36]. These data sets are Wisconsin breast cancer (WBC), BUPA, Vehicle and Pima Indians Diabetes. We also used a large remote sensing data RS [37]. Table 1 summarizes these data sets.

3.1. Data Sets

1) Iris: It is a 4-dimensional data having 150 points. There are three classes with each class having 50 points.

2) BUPA : This data set on liver disorder has 345 points in six dimensions distributed into two classes.

3) Diabetes: A two-class data set with 768 instances in 8-dimension.

4) WBC: The Wisconsin breast cancer data set is in 9-dimension and has 699 samples distributed in two classes (malignant and benign). Here we have ignored the instances with missing values.

5) RS: It is a Landsat-TM3 satellite image of size 512×512 (= 262144 pixels) pixels captured by seven sensors/channels [37]. The 512×512 ground truth data provide the actual distribution of classes of objects captured in the image. Thus each pixel is represented by a seven dimensional intensity vector. The class distribution of the samples is given in Table 2. Following the protocols used in other studies, we use only 200 random points from each of the eight classes as training samples.

6) Vehicle: This 18-dimensional data set has 846 samples distributed in four classes. We have normalized the feature values for this data set.

Table 1: Data sets.

Name of data set	No. of classes	No. of features	Size of data set
Iris	3	4	150 (50+50+50)
BUPA	2	6	345 (145+200)
Diabetes	2	8	768 (500+268)
WBC	2	9	683 (444+239)
RS	8	7	262144 (see Table 2)
Vehicle	4	18	846 (212 + 217 + 218 + 199)

Table 2: Different classes and their frequencies for RS data.

Classes	Frequencies
Forest	176987
Water	23070
Agriculture	26986
Bare ground	7400
Grass	12518
Urban area	11636
Shadow	3197
Clouds	350
Total	262144

3.2. GP Parameters

The GP parameters that are common for all data sets are given in Table 3 and those that differ are given in Table 4.

Table 3: Common GP parameters for all data sets.

Parameters	Values
Probability of crossover operation, p_c	0.75
Probability of reproduction operation, p_r	0.10
Probability of mutation operation, p_m	0.15
Tournament size, τ	5
Total number of generations the GP is evolved, M	50

Table 4: Other GP parameters.

Data set	Iris	BUPA	Diabetes	WBC	RS	Vehicle
Population size	200	200	200	200	300	300
Initial height of a tree	4-6	4-8	4-8	4-8	4-8	6-12
Maximum height of a tree	6	8	10	11	9	17

The vehicle data is in a higher dimension than the other data sets used here and also it is a complex data set. The RS data has many classes and it is also a complex data. Hence we have taken a larger population for the vehicle and RS data. The height of a tree depends on the number of features. If there are F features, then the height of the tree will be $F + 2$. So, in case of Iris data, if a rule consists of all 4 features, then the height of the tree will be 6. Similarly, for Diabetes and WBC, the maximum possible heights of the trees are 10 (8+2) and 11 (9+2), respectively. As the dimension (number of features) increases, the chances of redundant features also increase, hence, we restrict rules not to use all features. Thus, for Vehicle data a rule is allowed to use up to 15 features. Note that, these numbers are chosen in an ad hoc manner because our objective in this paper is not to do feature analysis. For Iris and BUPA data sets, the initial height of a tree is 4-6. This means, we generate rules containing 2 ($= 4 - 2$) to 4 ($= 6 - 2$) features. We have kept the same (4-8) initial height for Diabetes, WBC and RS data sets as they have almost the same dimension.

3.3. Results

We have used the (modified) lilgp [38] to develop our programs and we have conducted three experiments. In the first and second experiments, except for RS and Diabetes data sets, we have used 10-fold cross-validation to estimate the accuracy.

For RS data, we have run GP 10 times using 200 points from each class for training and remaining points for test. Diabetes data has been used in the third experiment to compare with an existing GP based method. For other data sets, in both experiments, GP is run 10 times each time involving a 10-fold cross-validation (total 10 × 10 = 100 runs). Here, performance of the fuzzy rules using 5 different proportions of FKM prototypes and Random prototypes is studied. In the first case (1:0), only the FKM prototypes are used in the rules and in the last case (0:1), only random prototypes are used in the rules. In the intermediate cases $(1 - p_{rp} : p_{rp})$, rules use FKM prototypes with probability $1 - p_{rp}$ and random prototypes with probability p_{rp}.

Table 5: Test accuracy with initial 10 rules per class.

FKM : Rand	Iris	BUPA	WBC	RS	Vehicle
1 : 0	93.45 ±0.79	67.38 ±1.79	97.34 ±0.38	75.23 ±0.89	68.48 ±0.76
0.75 : 0.25	94.10 ±0.69	65.96 ±1.43	97.40 ±0.52	79.41 ±0.72	70.02 ±0.89
0.50 : 0.50	94.43 ±0.53	65.14 ±1.68	97.47 ±0.31	81.33 ±0.58	70.34 ±0.85
0.25 : 0.75	96.53 ±0.49	65.09 ±1.46	97.59 ±0.41	80.09 ±0.64	71.56 ±0.68
0 : 1	96.33 ±1.08	68.83 ±1.98	96.88 ±0.55	76.57 ±0.78	69.08 ±0.79

Table 6: Average number of rules with initial 10 rules per class.

FKM : Rand	Iris	BUPA	WBC	RS	Vehicle
1 : 0	7.2	7.4	6.8	8.7	8.9
0.75 : 0.25	6.2	6.8	5.4	7.3	6.6
0.50 : 0.50	4.3	4.9	5.1	6.4	5.0
0.25 : 0.75	3.3	3.7	4.1	4.8	3.4
0 : 1	2.3	2.8	2.7	2.9	2.5

In the first and second experiments, we have initialized classifiers with $b_k = 10$ and $b_k = 5$ rules per tree T_k, respectively. For the first case (1:0), when we use only FKM prototypes, we try to find b_k FKM prototypes. But for some data sets, the FKM resulted in fewer prototypes. If the total number of FKM prototypes, g_k, is less than b_k for class k, then some FKM prototypes will be used more than once in tree T_k. In the intermediate cases (when both FKM and random prototypes are used), the FKM algorithm is used to obtain $(1 - p_{rp}) * b_k$ prototypes. For example, in Case 2 of the first experiment, we try to obtain 0.75 * 10 ≈ 8 FKM prototypes.

Tables 5 and 7 show the average test accuracies along with standard deviation for

Iris, BUPA, WBC, RS, and Vehicle data sets. The performance of the classifiers is comparable irrespective of the type prototypes that are used to generate the initial rules. From Table 5 we find that usually the combined use of both types of prototypes yields better performance than that by using only one type of prototype. Also, we note that the performance of classifiers by using only random prototypes is very good. It is interesting to note that for Iris data in Table 5 the accuracy is 96.33% using only random prototypes while the accuracy is 93.45% using only the FKM prototypes. And in Table 7, it is 95.80% using only random prototypes as against 93.30% using only FKM prototypes. Similarly, for BUPA data, random generated rules outperform FKM prototype based rules. In Table 7, the average test accuracy is 69.20% using only random prototypes.

Tables 6 and 8 show the average number of rules of a tree over all trees in the final GP population. Table 6 reveals that when we use only random prototypes, we obtain only 2-3 rules although we start with 10 rules. The reason may be that when the prototypes are generated randomly, the proportion of bad prototypes is high. Our pruning scheme removes the unwanted rules arising from bad prototypes. This results in trees with fewer rules. On the other hand, when only FKM prototypes are used, almost all rules become important because the associated prototypes are placed in dense areas of the data. Since we do not try to optimize the number of rules, we may end up with more rules to model some areas of the input space.

Table 7: Test accuracy with initial 5 rules per class.

FKM:Rand	Iris	BUPA	WBC	RS	Vehicle
1 : 0	93.30 ±0.71	64.68 ±1.36	97.22 ±0.40	78.14 ±0.74	70.38 ±1.14
0.75 : 0.25	95.65 ±0.68	66.13 ±1.17	97.18 ±0.24	77.37 ±0.82	69.83±1.23
0.50 : 0.50	95.93 ±0.70	69.32 ±1.24	97.26 ±0.36	80.47 ±0.68	70.61±0.97
0.25 : 0.75	97.13 ±0.65	68.22 ±1.65	97.58±0.45	78.93 ±0.59	69.08±1.31
0 : 1	95.80 ±0.63	69.20 ±1.36	97.50 ±0.29	76.82 ±0.92	65.22±1.53

Table 8: Average number of rules with initial 5 rules per class.

FKM : Rand	Iris	BUPA	WBC	RS	Vehicle
1 : 0	3.3	3.9	2.3	4.4	4.2
0.75 : 0.25	3.1	3.0	2.7	3.4	3.8
0.50 : 0.50	2.8	2.9	2.3	3.6	3.3
0.25 : 0.75	2.3	2.2	2.3	2.7	2.8
0 : 1	2.1	2.2	2.4	2.1	2.3

It can be observed that the overall performance of classifiers with 5 initial rules per class is slightly better than that using initial 10 rules per class. Two possible reasons for these are: More rules per class increases the chance of having more poor rules and hence more chance for poor rules to get involved in genetic operation. As a result, useful rules may get less chance to evolve. Also more rules may lead to memorization of the training data resulting in poor test performance.

In the second experiment, for Iris data we have obtained 97.13% accuracy when the composition of FKM and random prototypes is 0.25:0.75. Again, for WBC data the average test accuracy is 97.58% with 0.25:0.75 composition of the two types of prototypes. In [28], the GP is used to evolve fuzzy classification rules. The reported accuracy for Iris data using 10-fold validation is 95.3%.

In first experiment, with RS data, when the proportion is 0.50:0.50, we obtain the average accuracy of 81.33%. In case of Vehicle data, the average accuracy is 71.56% when the composition is 0.25:0.75.

Lim et al [39] have presented a comparison of 33 classification algorithms by using a large number of benchmark data sets. Table 9 summarizes comparison of mean error rate of our methodology with the results reported in the literature using 10-fold cross-validation mechanism for BUPA, WBC and Vehicle data [11, 39].

Table 9: Comparison of mean error rate m_{err} with other approaches [11, 39].

Data set	BUPA	WBC	Vehicle
Our method	0.307-0.353	0.024-0.031	0.284-0.348
Twenty-two tree based methods	0.279-0.432	0.031-0.085	0.206-0.487
Nine statistical approaches	0.280-0.401	0.034-0.048	0.145-0.224
Two NN classifiers	0.329,0.330	0.028,0.034	0.372,0.374

To visualize how the learning (evolution) converges toward better solutions, we have plotted the best fitness and the average fitness of the solutions with generation for two data sets: Iris and WBC. For each data set we consider two extreme cases, using only FKM prototypes and using only random prototypes. Fig.7 and Fig.8 depict variation of the best and average fitness of a typical run for Iris data when only FKM and only random prototypes are used, respectively. Although the variation of average fitness is similar in the two cases, the change in the best fitness is more smooth in case of random prototype. Similarly, Fig.9 and 10 display the variation of the best fitness and average fitness of a typical run for WBC when only FKM and only random prototypes are used, respectively. The characteristic of Fig.7 and 8 is also present here. This demonstrates the consistency of our algorithm. As the FKM prototypes quantize the data much better than the random prototypes, we obtain better classifiers with higher fitness value at the beginning. Figures 7-10 correspond to the cases with initially 5 rules per class.

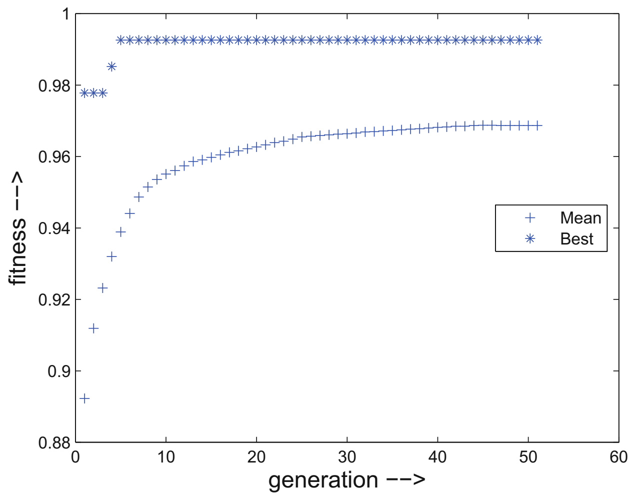

Fig. 7 Best fitness and mean fitness over generations for IRIS: all FKM prototypes

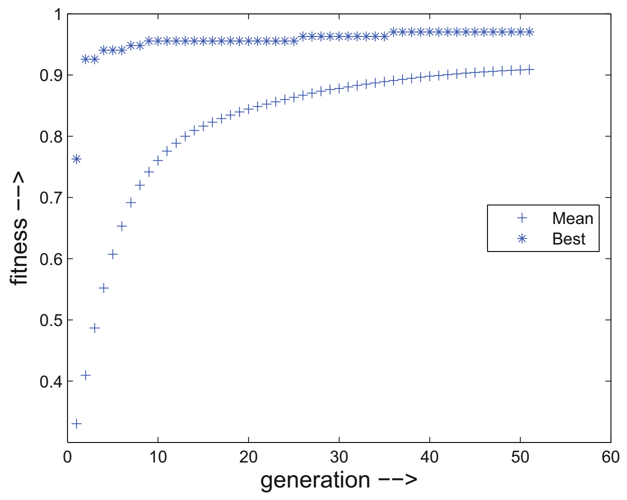

Fig. 8 Best fitness and mean fitness over generations for IRIS: all random prototypes

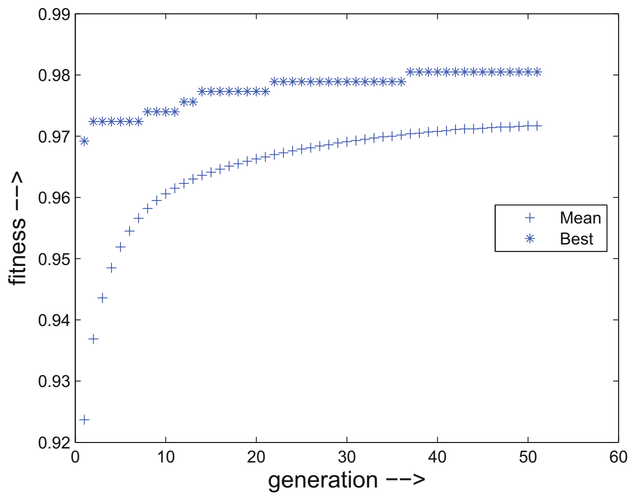

Fig. 9 Best fitness and mean fitness over generations for WBC: all FKM prototypes

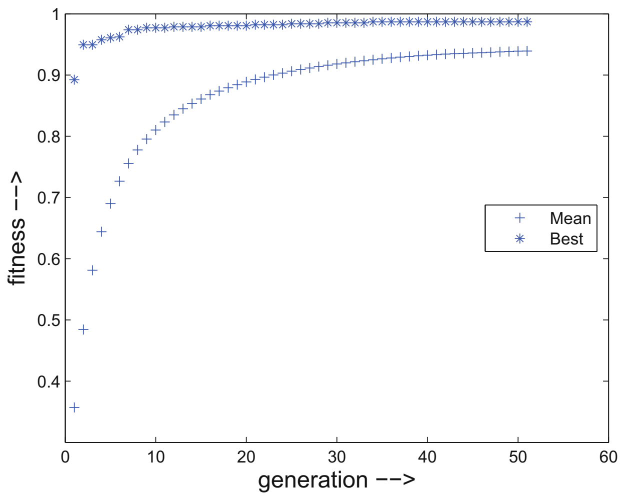

Fig. 10 Best fitness and mean fitness over generations for WBC: all random prototypes

As an illustration, we show the expressions corresponding to the three trees of the best classifier for the Iris data in a typical run:

Tree1: (OR (AND (CT x_3 0.18770 1.42674) (CT x_4 0.05320 0.20993)) (AND (CT x_2 0.27560 3.69521) (CT x_4 0.12370 0.27334)))

Tree2: (OR (AND (CT x_2 0.26417 2.59364) (CT x_4 0.14816 1.19223)) (AND (AND (AND (CT x_1 0.35443 6.33675) (CT x_2 0.28237 2.93938)) (CT x_3 0.22653

4.59761)) (CT x_4 0.13607 1.44758)) (AND (AND (CT x_1 0.87542 4.58146) (CT x_2 0.27787 3.46339)) (CT x_3 0.47887 4.53061)))

Tree3: (OR (AND (CT x_2 0.26858 3.13959) (CT x_4 0.14736 2.03583)) (AND (CT x_3 0.51499 5.96831) (CT x_4 0.54374 1.20857)))

In the above rules, CT stands for "CLOSE TO" and ($CT\ x\ y\ z$) represents a clause "x CLOSE TO z" which is modeled by a Gaussian membership function with center at z and spread y. The above expressions can be simplified as follows:
Class 1:

R1: If x_3 CLOSE TO$_{\sigma_{13}=0.19}$ 1.43 AND x_4 CLOSE TO$_{\sigma_{14}=0.05}$ 0.21 **then** *Class 1*

R2: If x_2 CLOSE TO$_{\sigma_{22}=0.28}$ 3.70 AND x_4 CLOSE TO$_{\sigma_{24}=0.12}$ 0.27 **then** *Class 1*

Class 2:

R1: If x_2 CLOSE TO$_{\sigma_{12}=0.26}$ 2.59 AND x_4 CLOSE TO$_{\sigma_{14}=0.15}$ 1.19 **then** *Class 2*

R2: If x_1 CLOSE TO$_{\sigma_{21}=0.35}$ 6.34 AND x_2 CLOSE TO$_{\sigma_{22}=0.28}$ 2.94 AND x_3 CLOSE TO$_{\sigma_{23}=0.23}$ 4.60 AND x_4 CLOSE TO$_{\sigma_{24}=0.14}$ 1.45 **then** *Class 2*

R3: If x_1 CLOSE TO$_{\sigma_{31}=0.88}$ 4.58 AND x_2 CLOSE TO$_{\sigma_{32}=0.28}$ 3.46 AND x_3 CLOSE TO$_{\sigma_{33}=0.48}$ 4.53 **then** *Class 2*

Class 3:

R1: If x_2 CLOSE TO$_{\sigma_{12}=0.27}$ 3.14 AND x_4 CLOSE TO$_{\sigma_{14}=0.15}$ 2.04 **then** *Class 3*

R2: If x_3 CLOSE TO$_{\sigma_{23}=0.52}$ 5.97 AND x_4 CLOSE TO$_{\sigma_{24}=0.54}$ 1.21 **then** *Class 3*

In the above rules, the parameters are rounded to two digits. When we apply the above classifier on all points of Iris data set, it could correctly classify all but one point.

In [25], GP is used to develop fuzzy classifier rules. For comparison with this GP method, named G^3P, we have conducted the third experiment. For G^3P, authors have partitioned the data sets including Diabetes and WBC into 3 sets: Training set with 50% samples, validation set with 25% samples and test set with 25% samples. Since we have not used validation set, we have considered 75% samples for training and 25% samples for test. We have selected samples for the test in the same manner as described in [25]. Then we run GP with different number of rules using different types of prototypes. For each case, GP is run 20 times and the test errors are calculated. Tables 10 and 11 show the best test error, average and standard deviation of errors over 20 GP runs for WBC and Diabetes data sets, respectively. In the first case, trees are generated with 2 rules per class by using only FKM prototypes. In the second case, a tree is initialized with 5 rules by using FKM prototypes with probability 0.25 and random prototypes with probability 0.75. In the third case, GP evolution is started with 10 rules per tree with only random prototypes. For the WBC data, with different combinations of FKM and random prototypes, we have obtained the best error rates in the range 1.17 %-1.75% and average error rates in 2.3%-3.66%. The reported best and average error rates obtained using G^3P were 2.29% and 4.39%, respectively. For the Diabetes data, the best error varies between 19.27% to 20.83% and average error is in 23.16%-23.30% as shown in Table 11. In [25], the best error reported using G^3P is 21.98% and average error is 26.47%. For both data sets, our GP method performed marginally better than the G^3P method.

Table 10: Test error for the WBC data set.

FKM : Rand	Initial No. of rules	Best error rate	Average error rate	Standard deviation
1 : 0	2	1.75	2.81	0.438
0.25 : 0.75	5	1.17	2.3	0.45
0 : 1	10	1.75	3.66	0.89

Table 11: Test error for the Diabetes data set.

FKM : Rand	Initial No. of rules	Best error rate	Average error rate	Standard deviation
1 : 0	2	19.27	23.16	1.71
0.25 : 0.75	5	20.83	23.30	1.71
0 : 1	10	19.79	23.23	1.72

Table 12: Reported test error in another GP work [25].

Data sets	Best error rate	Average error rate	Standard deviation
WBC	2.29	4.39	1.42
Diabetes	21.98	26.47	3.40

4. Conclusion

We have used genetic programming to extract fuzzy rules for designing classifiers. For a c-class problem, a classifier is represented by c trees. Each tree T_k represents a set of rules for class k. Our GP methodology evolves the required fuzzy classifier by determining: 1) number of rules, 2) features required to define the antecedent claus of a rule, and 3) values of the parameters to define the membership functions.

The initial rule base is obtained using randomly generated prototypes, FKM prototypes and a mixture of the two. In this context, we have proposed a new mutation operation to alter the parameters of the rules. We have also introduced a scheme to remove inaccurate and inactive rules. The method is validated on six benchmark data sets. It is observed that the performance of the evolved classifiers is not affected much by the type of prototypes used to define the initial rules and sometimes the performance of the classifiers is found to be better when we use a mixture of FKM and random prototypes to initialize the rule base. It is interesting to note that of the two extreme cases, use of only random prototypes is found to work better than the use of only FKM prototypes in most cases.

There are many avenues where further investigation is needed. For example, we can explore the utility of such a method for function approximation type system identification task. Our method requires a few algorithmic parameters to be set. We have chosen these parameters in an ad hoc manner based on a few trials. All these parameters may be chosen by using a validation set.

Acknowledgments

We thank the anonymous reviewers for their insightful comments, which have helped us to improve our manuscript.

References

1. Goldberg D E (2002) Genetic algorithms in search, optimization, and machine learning. Pearson Education
2. Koza J R (1992) Genetic programming: On the programming of computers by means of natural selection. MIT Press
3. Bruke E K, Kendall G (Ed.) (2005) Search methodologies: Introductory tutorials in optimization and decision support techniques. Springer
4. Duda R O, Hart P E, Stork D G (2001) Pattern classification. Wiley, 2nd Edition.
5. Nauck D, Kruse R (1997) A neuro-fuzzy method to learn fuzzy classification rules from data. Fuzzy Sets and Systems 89: 277-288
6. Chakraborty D, Pal N R (2004) A neuro-fuzzy scheme for simultaneous feature selection and fuzzy rule-based classification. IEEE Trans. Neural Networks 15(1): 110-123
7. Pal K, Pal N R, Keller J M (1998) Some neural net realizations of fuzzy reasoning. Int. Journal of Intell. Systems 13: 859-886
8. Pedrycz W (1993) Fuzzy neural networks and neurocomputations. Fuzzy Sets and Systems 56(1): 1-28
9. Oh S K, Pedrycz W, Park H S (2003) Hybrid identification in fuzzy-neural networks. Fuzzy Sets and Systems 138(2): 399-426
10. Pedrycz W (1992) Fuzzy neural network with reference neurons as pattern classifiers. IEEE Transactions on Neural Networks 3(5): 770-775
11. Muni D P, Pal N R, Das J (2004) A novel approach for designing classifiers using genetic programming. IEEE Trans. Evolut. Comput. 8(2): 183-196
12. Muni D P, Pal N R, Das J.(2006) Genetic programming for simultaneous feature selection and classifier design. IEEE Trans. SMC - B, 36(1): 106-117
13. Folino G, Pizzuti C, Spezzano G (2006) GP ensembles for large-scale data classification. IEEE Trans. on Evolutionary Computation 10(5): 604-616
14. Carse B, Fogarty T C, Munro A (1996) Evolving fuzzy rule based controllers using genetic algorithms. Fuzzy Sets and Systems 80: 273-293
15. Ishibuchi H, Nakashima T, Murata T (1999) Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. IEEE Trans. on SMC-B 29: 601-618
16. Ishibuchi H, Nozaki K, Yamamoto N, Tanaka H(1995) Selecting fuzzy if-then rules for classification problems using genetic algorithms. IEEE Trans. Fuzzy Syst. 3(3): 260-270

17. Russo M (2000) Genetic fuzzy learning. IEEE Trans. on Evolutionary Computation 4(3): 259-273
18. Liao G C, Tsao T P (2006) Application of a fuzzy neural network combined with a chaos genetic algorithm and simulated annealing to short-term load forecasting. IEEE Trans. on Evol. Compt. 10(3): 330-340
19. Casillas J, Carse B, Bull L (2007) Fuzzy-XCS: A Michigan genetic fuzzy systems. IEEE Trans. Fuzzy Systems 15(4): 536-550
20. Johnson R W, Melich M E, Michalewicz Z, Schmidt M (2005) Co-evolutionary optimization of fuzzy logic intelligence for strategic decision support. IEEE Trans. on Evol. Compt. 9(6): 682-694
21. Hoffmann F, Schauten D, Holemann S (2007) Incremental evolutionary design of TSK fuzzy controllers. IEEE Trans. Fuzzy Systems 15(4): 563-577
22. Cordon O, Gomide F, Herrera F, Hoffmann F, Magdalena L (2004) Ten years of genetic fuzzy systems: Current framework and new trends. Fuzzy Sets and Systems 141: 5-31
23. Herrera F (2008) Genetic fuzzy systems: Taxonomy, current research trends and prospects. Evolutionary Intelligence 1: 27-46
24. Ishibuchi H (2007) Multi-objective genetic fuzzy systems: Review and future research directions. Proc. of 2007 IEEE international Conference on Fuzzy Systems: 913-918
25. Tsakonas A (2006) A comparison of classification accuracy of four genetic programming-evolved intelligent structures. Information Sciences 176: 691-724
26. Pedrycz W, Reformat M (2003) Evolutionary fuzzy modeling. IEEE Trans. on Fuzzy Systems 11(5): 652-665
27. Sanchez L, Couso I, Corrales J A (2001) Combining GP operators with SA search to evolve fuzzy rule based classifiers. Information Sciences 136: 175-191
28. Mendes R R F, Voznika F B, Freitas A A, Nievola J C (2001) Discovering fuzzy classification rules with genetic programming and co-evolution. Lecture Notes in Computer Science 2168: 314-325
29. Dounias G, Tsakonas A, Jantzen J, Axer H, Bjerregaard B, Keyserlingk D (2002) Genetic programming for the generation of crisp and fuzzy rule bases in classification and diagnosis of medical data. Proc. 1st Int. NAISO Congr. Neuro Fuzzy Technologies, Canada, Academic Press, [CD-ROM]
30. Berlanga F J, Rivera A J, Jesus M J del, Herrera F (2010) GP-COACH: Genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems. Information Sciences 180: 1183-1200
31. Edmonds A N, Burkhardt D, Adjei O (1995) Genetic programming of fuzzy logic production rules. IEEE International Conference on Evolutionary Computation 2: 765-770
32. Chien B C, Lin J, Hong T P (2002) Learning discriminant functions with fuzzy attributes for classification using genetic programming. Expert Systems with Applications 23: 31-37
33. Pal N R, Kumar V, Mandal G K (2002) Fuzzy logic approaches to structure preserving dimensionality reduction. IEEE Trans. on Fuzzy Systems 10: 277-286
34. Bezdek J C, Keller J, Krisnapuram R, Pal N R (1999) Fuzzy models and algorithms for pattern recognition and image processing. Kluwer Academic Publishers
35. Anderson E (1935) The irises of the gaspe peninsula. Bulletin of the Amer. Iris Soc. 59: 2-5
36. Blake C L, Merz C J (1998) UCI Repository of machine learning databases. Univ. of California, Dept. of Inform. Comput. Science
37. Kumar A S, Chowdhury S, Mazumder K L (1999) Combination of neural and statistical approaches for classifying space-borne multi-spectral data. Proc. ICAPRDT, Calcutta, India: 871
38. Zongker D, Punch W F (1995) lil-gp 1.0 user's manual. Tech. Rep., MSU Genetic Algorithms and Research Application Group (GARAGe), [Online]. Available: http://garage.cps.msu.edu
39. Lim T S, Loh W Y, Shih Y S (2000) A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. Machine Learning 40: 203-229