Genetic Programming Fuzzy Rule Extractor Using Class Preserving Representation

Jiri Kubalik[a,b] Leon Rothkrantz[b] Jiri Lazansky[a]

[a] CTU Prague, Technicka 2, 166 27 Prague 6, Czech Republic

[b] TU Delft, P.O. Box 356, 2600 AJ Delft, The Netherlands

Abstract

This paper describes a genetic programming approach to the construction of fuzzy classification system with if-then fuzzy rules. Recently many research studies were focusing on utilisation of evolutionary techniques for automatically extracting fuzzy rules from data. In this paper we present a method based on genetic programming with a special structure preserving representation and special rule base adjusting operators working on it. First results show that the new features added to standard GP extractor considerably improve both performance and comprehensibility of found fuzzy rules.

1. Introduction

One of the most challenging problems in machine learning is finding methods to transform implicit knowledge to explicit knowledge. Implicit knowledge, example of which are data sets can not be easily understood by humans. The opposite case is explicit knowledge represented in human more comprehensible way such as rules. The explicit knowledge description used here takes shape of a fuzzy rule set.

The fundamental difference between fuzzy logic and traditional logic is the ability to express vagueness and uncertainty in data by means of fuzzy sets. Fuzzy sets, introduced by Zadeh [2], allow to represent linguistic identifiers like “high”, “medium”, “about” that are closer to human reasoning. As such an employment of human domain expert in a process of constructing of proper fuzzy rules for a given problem was often necessary. Obviously a problem how to find the fuzzy rules automatically from the data without expert knowledge has become a new challenge.

Recently many research studies were focusing on utilisation of strong searching capacity of evolutionary techniques for (1) determining membership functions with a fixed a fixed number of fuzzy rules, (2) finding fuzzy rules with known membership functions and (3) finding both membership functions and rules simultaneously [5]. Especially genetic programming [1] appeared to be well suited for the problem of learning fuzzy rules since the rule set can be naturally represented in the form of parse tree on which GP operates.

In this paper we present a GP extractor with a special structure preserving representation and special rule base adjusting operators working on it.

2. Definition of Fuzzy Rule Classification System

Fuzzy controllers based on a utilisation of fuzzy sets are conceptually very simple. The core of the controller is usually defined as a set of fuzzy if-then rules whose antecedents and consequences are made up of linguistic variables and associated fuzzy membership functions. First the consequences from all fired rules are computed. Then the outputs are numerically aggregated by the fuzzy OR operator (typically the maximum value is taken) and finally defuzzified to yield a single real number output. For a more detailed introduction to fuzzy sets and fuzzy control see [3].

The goal of this work was to design a tool for extraction of fuzzy rules from classified data. So it is a task of supervised learning, where the training data set contains <input, output> records. Each record consists of n-sized input vector $\vec{x}$ and m-sized output vector $\vec{o}$. For the sake of easy manipulation with the data it is desirable that both input and output values reside within a unit hypercube. Thus all input values are normalised to [0, 1]. Output values are treated according to the type of data input-output projection. In the case of fuzzy projection the output values are normalised into [0, 1], resulting in the projection $[0, 1]^n \mapsto [0, 1]^m$. In the case of Boolean projection $[0, 1]^n \mapsto \{0, 1\}^m$, all of the output elements are set to 0 but one, which is 1.

The fuzzy rule set of the classification system that we use consists of k if-then fuzzy rules of the following type:

$Rule_i$: IF x_{i1} is A_{i1} AND … x_{ij} is A_{ij} THEN Class C_i with $CF=c_I$, where $i \in [1, k]$

Note that the number of rules k is not set in advance, instead it is let up to the extractor to find the best number of rules. The condition part of the rule is conjunction of antecedent terms x_{ij} is A_{ij}, where x_{ij} is some input variable of the n-dimensional input vector and A_{ij} is its associated linguistic value. For all input variables 5 linguistic values are defined {small, medium small, medium, medium large, large}, whose triangular membership functions are depicted in Figure 1. The fuzzy operator AND is defined in the way that it returns the minimum truth value of the involved terms.

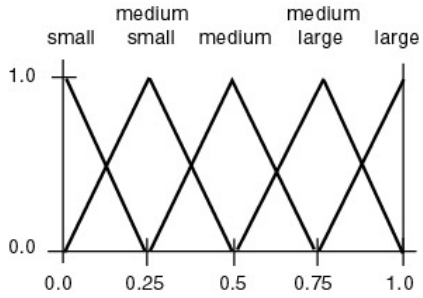

Figure 1: Membership functions.

The consequent part of the rule is determined by value of the consequent class C_i and the certainty factor CF. Linguistic variable C_i can stand for any class variable of the m-dimensional output $C_1 \ldots C_m$. For the class variables only one linguistic value, let say *High*, is defined. The value of C_i is computed using a simple inference method so that it is given the truth value generated by the premise. The certainty factor CF represents the

"fire power" of each rule so the truth value of C_i is multiplied by CF in order to get the final output value of the rule. Each rule affects just one element of the output vector $\vec{o}$. If multiple rules fire on the same consequent class a simple fuzzy OR operator is used to specify which rule outcome to accept (usually takes the maximum one). After outcomes of all rules are computed we get an m-sized real fuzzy output vector. In other words the fuzzy classifier carries out a projection of $[0, 1]^n \mapsto [0, 1]^m$. This projection is compatible with fuzzy data sets. If a Boolean projection is needed i.e. output $\{0, 1\}^m$ instead of $[0, 1]^m$, the real output of the fuzzy system is transformed to the Boolean. This is achieved by using the following "a winner takes it all" decision rule

$\forall i \in [1, m]$: If $o_i > o_j, \forall i \neq j$ then $o_i = 1$ else $o_i = 0$,

where o_i is an element of the output vector $\vec{o}$.

3. Design of Genetic Programming Extractor

Genetic programming (GP) is a powerful technique for automatically generating computer programs to perform a wide variety of tasks [1]. GP belongs to a broader class of so called evolutionary techniques that share a common idea of adopting mechanisms of survival of fittest and genetic heredity to evolve a desired solution to given problem. GP operates on population of candidate solutions, called individuals, each represented as a hierarchical parser tree. All individuals are evaluated i.e. assigned a fitness value expressing a performance measure of the represented solution. Such a population of diverse individuals is then being evolved generation by generation by means of reproduction and crossover and mutation operators. The process of evolving the population runs until some stopping criterion is fulfilled, usually it runs for some pre-specified number of generations. The best solution encounter during the whole run is then returned as the final solution.

When applying GP to any problem one has to go through the following subtasks:

- design of a proper representation of the solution of the given problem,
- definition of a quality measure for assessing the solutions,
- design of a proper recombination operators for generating new candidate solution,
- choosing of proper values for control parameters of GP.

3.1. Encoding of Fuzzy Rule Base

First the basic elements – terminals and functions – of trees must be defined. Functions require arguments and represent inner nodes of trees. Terminals require no arguments and represent leaves. We have chosen following terminal set and function set

T = {IS, CL, CF} and F = {OR, IF, AND}.

Terminals have the following meaning:

- **IS** is a constant that takes on a random combination of an antecedent linguistic variable and an associated fuzzy membership function. When presented an input vector IS node returns a truth value of its proposition.
- **CL** is an integer constant from the range [1, m] determining the consequent class to which the rule classifies i.e. the output vector element that is affected by this rule.

- **CF** is a real constant chosen from the range [0.1, 1.0]. We use a threshold 0.1 because values close to 0.0 mean the corresponding rule is unimportant.

Specification of function arguments and the action they carry out are as follow:

- **OR** function has just two arguments. Each of them may be either another OR function or IF function. The role of OR node is only to provide a means for structural binding of multiple rules in a tree. It does not involve any computation of the output. This is all done in IF nodes.
- **IF** function implements a single fuzzy rule and has the following arguments:
  1. the antecedent part, which may be represented by IS node or AND node,
  2. the consequent class node CL, and
  3. the certainty factor CF.

  IF node calculates an output value of the rule and if the value is higher than the current value of the corresponding output vector element then replaces it.
- **AND** function is used to combine terms of the antecedent part of the rule. AND node has just two arguments (of type AND or IS) and returns the minimum of their values.

In many genetic programming applications the terminals and functions are defined in such a way that so-called *closure* property is guaranteed. This means that any node from the combined set, $F \cup T$, can be a child in a parser tree for any other node without violating any constraints imposed on the structure of the tree. Obviously this is not the case here. One can imagine how an arbitrary placement of functions and terminals in a tree could lead to severe syntactic violations. For example any change in the order of the arguments to some IF node would make that IF node non-interpretable.

In fact, there are strong constraints imposed on the structure of the tree – such as number and types of function node arguments and the specific arguments' order. So it must be ensured that any genetic operator working on tree structures (such as tree generator, crossover and mutation operators, etc.) will produce only legal tree i.e. tree that represents a well-defined fuzzy rule set. To achieve this a concept of strongly typed genetic programming (STGP) has been introduced [4]. In STGP each function is given input and output types and each terminal is assigned an output type. When operating on genetic tree structure, it is made sure that all outputs linked to input are of the same type.

F / T	Output	Input
OR	0	0, 0
IF	0	1, 2, 3
AND	1	1, 1
IS	1	None
CLASS	2	None
CF	3	None

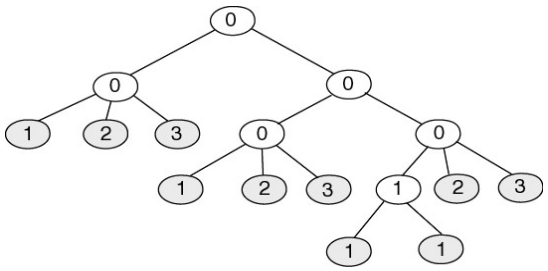

Figure 2: Strongly typed representation of rule set.

Figure 2 shows a definition of simple strongly typed representation. It uses 4 types {0, 1, 2, and 3}. For example function IF expects on its first input a node of type 1 (AND or IS) on its second input a node of type 2 (CL) and on the third input a node of

type 3 (CF). The output type of IF is 0 so it can be connected as an input node only to some OR node.

STGP guarantees that only legal trees are generated and that genetic operators maintain legal syntactic structure. On the other hand the semantic information encoded in the processed structures still remains hidden for genetic operators. Let us assume the crossover operator. For the sake of simplicity of the crossover operation it is usually implemented as a simple swapping of randomly chosen parts of two parental trees. While the only thing that is really checked is that the nodes, which are the roots of the swapped subtrees are of the same type. It is clear that using of such a simple recombination operator has an undesirable effect that it does mix irrelevant information – i.e. rules or parts of rules classifying to different classes are swapped.

Let us compare this situation to crossover operation in standard genetic algorithms (GAs). GAs use binary string representation called chromosome instead of trees to encode parameters of given problem. What happens there if a simple crossover based on swapping substrings between two parents is applied? The offspring are made up of parts taken either from one or the other parent. However, the bits representing certain parameter in parental chromosome come to the same positions into the offspring chromosomes so they do represent the same parameter there. It means that semantic meaning of swapped strings does not change through the action.

To achieve the same property of semantic-preserving operations in GP much more sophisticated operators should be designed or the representation should be adapted somehow. In this paper we present an example of the latter approach. We have proposed so-called class preserving representation. The main idea behind it is that all rules with the same consequent class CL are clustered into one compact subtree rather than they would be dispersed across the whole tree.

	ROOT	OR	IF	AND	IS	CL	CF
Input	1,…,1	1, 1	2, 3, 4	2, 2	None	None	None
Output	0	1	1	2	2	3	4

Table 1: Class preserving strongly typed representation.

Practically it looks so that there is a main root of the tree, which stands for the root of the whole rule set. This root has as many branches as is the number of output classes i.e. *m*. Each branch carries a subtree which represents subset of rules classifying to the class specified by the branch. So the rules appearing under the first root's branch have the consequent class 1, the rules under the second root's branch belong to class 2, etc. Note that this representation requires an additional type for the main root to distinguish it from the ordinary OR nodes. So the tree root is of type 0 and has as m inputs of type 1. The remaining data types are increased by 1, see Table 1.

3.2. Fitness function

A fitness function plays a crucial role in every genetic programming application. Its specification strongly affects the whole evolution process since it is usually the only information about the solved problem that is provided to GP system. In most of the applications in the field of rule extraction two contradictory primary goals are followed. The first one is the ability of the rule set to correctly classify data whilst the second one

focuses on generalising power and comprehensibility of the rules. In other words as simple rule base as possible that captures the most significant knowledge about the data to sufficient extent is searched for. We have used following two fitness measures:

- **Boolean fitness** for the Boolean data sets. The fitness combines two objectives – minimisation of a classification error and reducing of a complexity of the rule base. The classification error (the first part of the formula below) is given as a number of wrong classifications against examples over the whole training set. The second part of the function adds a penalty for the size of the trees. Note that the optimal classification accuracy is the primary goal here. So the penalty is defined just to distinguish between individuals with the same classification rate. It holds that of two individuals the better classifying individual always wins regardless of their size.

$$f_{Boolean} = \sum_{i=0}^{all\_points} \left(o_i^{data\_set} \neq o_i^{GP} \right) + \left(1 - \frac{1}{size} \right)$$

- **Fuzzy fitness** for the fuzzy data sets. The fitness used for fuzzy data sets has strengthened the part calculating the classification error. Since both the outputs of the fuzzy rule set and the fuzzy data set are of the type $[0, 1]^m$ we can use besides the number of wrong classifications also the measure how far the wrong classification from the desired one was.

$$f_{Fuzzy} = \sum_{i=0}^{all\_points} \left(o_i^{data\_set} \neq o_i^{GP} \right) + \left(1 - \frac{1}{size} \right) + \sum_{i=0}^{all\_points} \left(o_{best,i}^{GP} - o_{desired,i}^{GP} \right)$$

  This is supported by the third term of the fitness, which calculates a distance of the actual output from the desired one. Note that it does not compute a distance between rule set and data set output vectors instead it takes a distance between a winner element of the actual output and the element of the actual output that was supposed to be a winner. This component is calculated only if the actual output element differs from the desired one. As an example let us assume the fuzzy rule set output is [0.2, 0.9, 0.4], and the desired winner class is 3 whilst the actual winner class is 2. So the distance between the actual output and the desired output is 0.5 (i.e. 0.9-0.4).

3.3. Genetic Operators

It is now easy to design simple crossover and mutation operators for this representation that would manipulate trees in an efficient way. It is also easy to maintain some additional useful properties of the trees. For example there should be at least one rule for every output class present in the rule base otherwise the non-presented class's points would not have any chance to be classified. This property can be simply guaranteed by not allowing less than *m* branches leading from the root.

We have used following genetic operators - an *m*-point crossover, subtree modifying mutation, and two special rule base adjusting operators.

The proposed *m*-point crossover works on two parental trees so that a simple 1-point x-over schema is applied to all *m* class subtrees bellow the root. First an arbitrary node in the current class subtree of the first parental tree is chosen. Then a node of the same in corresponding subtree of the second tree is found. Finally the subtrees rooted in the selected nodes are swapped between the both parents. In this way an effective exchange of relevant information at the level of CF leafs, antecedents and their subparts (IS and AND nodes), and single/multiple rules (If and OR nodes) is ensured.

Mutation operates on only one individual. It randomly chooses a node in the tree (excluding the root) and replaces the node and its subtree by a new one generated according to the type of the selected node as follows:

- **CF** value is changed up or down by a value from the range [0, 0.1] not exceeding the CF boundaries [0.1, 1.0].
- **IS** is changed so that its current fuzzy membership function is replaced by the nearest higher or lower one. For example "x1_medium_small" would be changed to "x1_small" or "x1_medium".
- **CL** does not undergo mutation.
- **OR, IF, AND** nodes and their subtrees are replaced by newly generated subtree of the same root type and of the maximum depth as the original subtree is.

The role of the last two operators is to refine a rule set. They use usefulness information about each individual rule calculated over the whole training data set. It shows for each rule (1) how many times it determines the correct output and (2) how many times it causes wrong classification and which is the class of which the data were most frequently misclassified. The tree is traversed and every rule (IF node) is checked for it usefulness and either *delete_rule* or *change_class* operators are applied as follows:

- *delete_rule* is applied to any rule which does not determine the correct output of any data from the training data set. Obviously such a rule is completely useless and is removed from the rule set with the exception if it is the only one rule classifying to given class. This operator decreases size of the rule set.
- *change_class* operator works so that if the rule is better suited for classification to another class than its current class is then the rule's definition is changed – i.e. the rule is moved to the subtree of the better class. This operator increases classification accuracy of the rule set. Note that this operator can be applied to at most one rule of the whole rule base at a time because change in one rule may cause change in usefulness statistics of the other rules.

Alike such operators can be designed for any type of rule extraction problem – one just has to define some heuristic according to which the rules should be deleted or assigned a proper class.

3.4. Configuration of GP

The rest of the parameters used in the GP system are: *PopSize*=500, *Generations*=100, P_{cross}=0.8, P_{mut}=0.1, P_{rep}=0.1, tournament selection strategy with n=5, maximum number of nodes in the tree is 100. The starting population of trees is initialised with a growing strategy. The following populations are created so that $P_{cross} \times PopSize$ individuals are made by crossover, $P_{mut} \times PopSize$ individuals are mutated individuals from the old population and the rest of individuals come unchanged from the old population.

4. Experiments

The proposed rule extractor were tested and compared with an extractor using a simple strongly typed representation, simple 1-point crossover a mutation preserving the structural constraints. Following data sets were used for those experiments:

- *Iris plants Boolean data set* – 4 input attributes, 3 output classes, data set size 150,

- *Iris plants Fuzzy data set* – 4 input attributes, 3 output classes, data set size 256,
- *Wine Boolean data set* - 13 input attributes, 3 output classes, data set size 178,
- *Breast cancer Boolean data set* - 9 input attr., 2 output classes, data set size 286.

	Iris Boolean		Iris fuzzy		Wine		Breast cancer	
	Simple	C-P	Simple	C-P	Simple	C-P	Simple	C-P
Misclassifications	3	3	**0.7**	**0.0**	**2.0**	**0.0**	**61.6**	**58.0**
Rules	4	4	4.2	4	8.3	7.4	8.7	9.6
Generation	**25**	**7**	**52**	**17**	52	39	62	61

Table 2: Comparison of simple GP extractor (Simple) and GP extractor using class preserving representation (C-P) and *delete_rule* and *change_class* operators. The numbers represent average values obtained over sets of 10 independent experiments.

One can see that the new GP extractor using the class preserving representation and the special genetic operators performed better than the simple GP extractor on all tested data sets. The proposed extractor found rule sets with better misclassification rate on the Iris fuzzy, Wine and Breast cancer data sets. It also appeared to be much faster in finding the best rule sets than the simple GP extractor as can be clearly seen on both Iris data sets results. In the case of the Wine data set all three observed output parameters were in favor of our proposed GP extractor.

5. Conclusions

This paper presents a modification of a simple genetic programming fuzzy rule extractor. It is based on class preserving strongly typed representation, which is further supplied with special rule set refining genetic operators – *delete_rule* and *change_class*. First experiments show that the proposed rule extractor is capable of finding of better rule sets than the simple GP extractor in terms of the classification accuracy and the size of the rule sets.

References

[1] J. Koza. *Genetic Programming*. MIT Press, 1992.

[2] L. A. Zadeh. Fuzzy Sets. *Journal of Information and Control*, v8, 338-353, 1965.

[3] M. Jamshidi, N. Vadiee and T. Ross (Eds.). *Fuzzy Logic and Control: Software and hardware applications*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[4] David J. Montana. Strongly typed genetic programming. Technical Report 7866, Bolt Beranek and Newman, Inc., March 25, 1994.

[5] Sushmita Mitra and Yoichi Hayashi. Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework. *IEEE Trans. Neural Networks*, vol. 11, 748-768, 2000.