Mathware & Soft Computing 15 (2008) 21-40

Optimization of Fuzzy Rule Sets Using a Bacterial Evolutionary Algorithm

M. Drobics[1] and J. Botzheim[2]
[1] eHealth systems / Biomedical Engineering
Austrian Research Centers GmbH - ARC
A-1220 Vienna, Austria
[2] Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
H-1117 Budapest, Hungary
mario.drobics@arcsmed.at, botzheim@tmit.bme.hu

Abstract

In this paper we present a novel approach where we first create a large set of (possibly) redundant rules using inductive rule learning and where we use a bacterial evolutionary algorithm to identify the best subset of rules in a subsequent step. This enables us to find an optimal rule set with respect to a freely definable global goal function, which gives us the possibility to integrate interpretability related quality criteria explicitly in the goal function and to consider the interplay of the overlapping fuzzy rules.

1 Introduction

Inductive learning is concerned with finding a function $f(\mathbf{x}), \mathcal{X} \mapsto \mathcal{Y}$ which best fits a given data set X. As fuzzy rule bases are capable of fulfilling requirements regarding interpretability and accuracy, they are often used in applications, where expert knowledge is not available, but knowledge of the resulting system is essential [9].

Most methods for learning fuzzy rule bases (or fuzzy regression/decision trees) choose a stepwise approach to construct the rule base. Therefore, decisions are based on local criteria like entropy gain, confidence and support, or improvement in goodness of fit (e.g. mean squared error for regression problems or entropy gain for classification problems). This approach has two shortcomings: Firstly, selecting accurate rules individually is not sufficient, as the interaction of the rules is very important for the overall performance of the rule base in the fuzzy case—especially for regression problems. Secondly, it is usually not possible to define the goal function freely. This becomes crucial, when global criteria—like interpretability measures [15]—are involved.

Recent approaches which try to use more problem specific selection criteria have been presented e.g. for regression trees, where a rough approximation of the expected output is used. As at the time a node is split no results from the subtrees are available, the quality of the split is measured by interpolating between the mean values of each subgroup [17, 12].

$$\mathrm{MSE}_{\mathrm{DT}}(P, z, X) = \frac{\sum_{\mathbf{x}\in X} \mu_X(\mathbf{x})(\tilde{z}_P(\mathbf{x}) - z(\mathbf{x}))^2}{|X|},$$
$$\tilde{z}_P(\mathbf{x}) = t(P(\mathbf{x}))\bar{z}(X|P) + t(\neg P(\mathbf{x}))\bar{z}(X|\neg P),$$

where X is the data set, P the predicate to be applied, z the actual goal function and $\bar{z}(X|P)$ the average of z in X, weighted according to P. $t(.)$ denotes the truth evaluation function. Although this approach is an improvement over traditional approaches and it might be adopted to other error measures easily, it still uses a step wise approach and is therefore restricted to an approximation of the final tree structure. Other approaches use subsequent optimization techniques like pruning [17] or meta-optimization techniques like boosting [19].

Currently, only a few approaches like genetic programming [20] are capable of optimizing a complete rule base. These approaches, however, are usually very complex and time consuming, as the search space is extremely large.

We overcome these limitations by splitting the construction of the rules and the construction of the final rule base. Namely we construct a large set of rules first, where all rules fulfill only minimal requirements in terms of confidence and support. Then we select a much smaller subset of these rules using a bacterial evolutionary algorithm (BEA). As in the BEA we can define the goal function freely, we finally obtain a rule set which perfectly fits the given requirements. Comparable approaches limited to classification problems using genetic algorithms have been presented in [14] and [22]. The main disadvantage of these approaches is their complexity, caused by the use of GAs and a binary position related coding. Furthermore, the key advantage of this approach—its ability to find a good *combination* of rules—is much more powerful when applied to regression learning.

Bacterial evolutionary algorithms are simpler then genetic algorithms and it is possible to reach lower error levels within a short time. They comprise of two operations inspired by the microbial evolution phenomenon. The bacterial mutation operation which optimizes the chromosome of one bacterium, and the gene transfer operation which transfers information between different bacteria within the population. BEA have already been successfully applied to rule learning [8] and feature selection [7].

In this paper we first define the underlying language and the rule induction method used. Then we introduce the bacterial evolutionary algorithm and we show how this method can be applied to the problem of rule selection. Afterward, some simulation results are presented to illustrate the potential of this new approach. Finally, the paper is closed with an outlook to future work.

2 Prerequisites

Before we discuss how to select an optimal subset of rules, we have to clarify some prerequisites. First, we have to define the language used to construct the fuzzy rules, i.e. how fuzzy sets and fuzzy predicates are composed. Then, we will define the structure of the rules base and discuss, how minimal requirements concerning quality and interpretability for these rules can be established. Finally, we will describe, how the initial set of rules is generated.

2.1 Defining the underlying language

To define the underlying language for our fuzzy models, we have to consider the different types of input attributes that can occur. Basically, we can distinguish between three types of attributes:

Boolean categorical attributes: The domain X_i is a unstructured finite set of labels, for instance, types of car engines (gasoline, Diesel, hydrogen, electric) or classes of animals (birds, fish, mammals, etc.). The attribute values x_r^i are single elements of the label set X_i.

Fuzzy categorical attributes: There is again a unstructured finite set of labels, but with possible overlaps. Therefore, values of such kinds of variables may be fuzzy sets on this set of labels. For example, assume that we are given a finite set consisting of different grape varieties. Then blended wines (cuvees) cannot be assigned to single categories crisply.

Numerical attributes: The underlying domain X_i is the set of real numbers or a subset thereof (e.g. an interval). The attribute values x_r^i are real numbers, e.g. pressures, temperatures, incomes, ratios, etc.

Note that Boolean categorical attributes are special cases of fuzzy categorical attributes, since any crisp label can be considered as a fuzzy set of labels, too.

Fuzzy predicates for categorical attributes, boolean or fuzzy, can be defined easily in a straight forward manner. Finding appropriate fuzzy predicates for numerical attributes, however, is a subtle problem which has to be thought out carefully.

2.1.1 Generating meaningful fuzzy sets

To generate k unevenly distributed fuzzy sets according to the distribution of numeric values in the data set $\mathcal{X}$, we developed an algorithm called *CompFS* [10]. First, the centers c_i $(1 \leq i \leq k)$ of the fuzzy sets are initialized according to the data distribution. By initializing the fuzzy set centers with the according quantiles ($q_i = \frac{i-0.5}{k}$), we create an equi-frequent binning of the data set. To overcome problems which can occur when using the quantiles, it is necessary to readjust the fuzzy set centers by using a simple k-means algorithm. Of course, equal frequencies can no longer be guaranteed. Usually a few iterations of this clustering step suffice. Finally, the fuzzy sets are computed around these centers. The resulting

families of fuzzy sets form a partition and are in a proper order—to ensure highest interpretability of the results [5].

Although *CompFS* is capable of computing the fuzzy sets automatically with respect to a given data distribution, it requires the actual number of fuzzy sets as an input. In our experiments it has, however, turned out that the actual choice of this number influences the performance of the resulting model only slightly as long as a sufficiently large number of fuzzy sets is created. This is achieved by using ordering-based predicates and by defining the underlying fuzzy sets according to the actual distribution of the data such that there is a good chance, that a sufficiently good split is found already with a low number of sets. A detailed discussion of our experiments on this topic can be found in [12].

2.1.2 Defining appropriate predicates

After defining the underlying fuzzy sets, it is necessary to define appropriate predicates using these fuzzy sets. A fuzzy predicate p on X is uniquely represented by its truth function

$$t(p) \ : \ X \to [0,1] \tag{1}$$

where $t(p(x))$ is interpreted as the degree to which the value x fulfills the predicate p.

Suppose that the r-th attribute is numerical. This means that $X_r \subseteq \mathbb{R}$ and the values in the r-th component are real numbers. We assume that, for attribute r, a family of N_r linguistic labels $M_{r,1}, \ldots, M_{r,N_r}$ is defined. Depending on the underlying context of the attribute under consideration, these labels can be natural language expressions like *very low*, *medium*, and *large*. To each label $M_{r,j}$, we assign a fuzzy set with membership function $\mu_{M_{r,j}} \in \mathcal{F}(X_r)$ $(j = 1, \ldots, N_r)$ using *CompFS*. Furthermore, we can define the complement and the smallest superset with non-decreasing / non-increasing membership function for these sets. These new fuzzy sets correspond to the linguistic expressions *is not*, *is at least*, and *is at most*, respectively.

Given a set of linguistic labels $M_{r,1}, \ldots, M_{r,N_r}$ and their corresponding semantics modeled by fuzzy sets, we can now define $4 \cdot N_r$ atomic fuzzy predicates. The degrees to which a sample $\mathbf{x} \in X_1 \times \cdots \times X_{n+m}$ fulfills these predicates can be computed as follows $(j = 1, \ldots, N_r)$:

$$\begin{aligned} t(\mathbf{x} \text{ is } M_{r,j}) &= \mu_{M_{r,j}}(x_r) \\ t(\mathbf{x} \text{ is not } M_{r,j}) &= 1 - \mu_{M_{r,j}}(x_r) \\ t(\mathbf{x} \text{ is at least } M_{r,j}) &= \sup\{\mu_{M_{r,j}}(u) \mid u \le x_r\} \\ t(\mathbf{x} \text{ is at most } M_{r,j}) &= \sup\{\mu_{M_{r,j}}(u) \mid u \ge x_r\} \end{aligned} \tag{2}$$

Although the two latter ordering-based predicates are not absolutely necessary, they help to improve compactness, expressiveness, and interpretability of the results [3, 4, 5, 6].

2.2 Rule Structure

Before we start to discuss how to generate rule bases for a given learning problem, we have to clarify the structure of the resulting rule base. For classification problems (i.e. where the output parameter is either boolean or fuzzy categorical) this can be done in a straight forward manner:

Given a set of goal predicates $C_g, g \in G$, rules of the form

$$A_i \to C_g, \qquad i \in I_g, g \in G,$$

are defined. Such a rule is as *linguistic description* of the rule consequence C_g, which can be read as: "For all samples fulfilling A_i, C_g holds". In this setting, A_i is a filter on the set of samples and C_g is a classification.

In case that the goal parameter is numeric (i.e. for regression learning problems) we have to compute the set of goal predicates $C_g, g \in G$ first. Then we can proceed as for the categorical case. If the desired result is a real value and not a fuzzy set, the resulting fuzzy set has to be defuzzified to obtain the final result. In our tests, we used center-of-gravity defuzzyfication for this purpose.

2.3 Performance Evaluation

Beside measuring the quality of prediction, comprehensibility of the resulting rule base plays an important role in fuzzy systems. Thus, it is necessary to establish numeric measures which allow to compare different rule sets with respect to their interpretability. Such measures typically cover two aspects of the rule base: its size (i.e. number of rules and/or predicates) and the conditioning of the underlying fuzzy partitions [15]. Depending on the application domain, other aspects might be included as well (e.g. a preference on certain variables). As in our approach the underlying language is defined a-priori to the actual rule learning, we will restrict ourself to the measurement of the size of the rule base.

When creating the rules involved in the final rule base, we have to specify which rules are of potential interest. For a single rule, we will now specify two minimal quality requirements, which are aimed to remove useless/uninteresting rules. To ease notations we will restrict ourself to a single consequence parameter C.

Let $A_i \to C, i \in I$ be a rule set for a given learning problem, and let X be our set of samples. The first requirement on a rule $A_i \to C$ is, that the antecedent should only be fulfilled if the consequence is also fulfilled, i.e. each rule should be confident.

Definition 2.1. Let $A \to C$ be an arbitrary rule, and be X a given set of samples. The *confidence* of $A \to C$ with respect to X is defined according to:

$$\mathrm{conf}_X(A \to C) := \frac{|A(X) \cap C(X)|}{|A(X)|}. \tag{3}$$

The second requirement is, that the whole rulebase covers all possible states ($\bigcup_{i \in I} A_i = C$). This motivates the definition of the *support* measure:

Definition 2.2. Let $A \rightarrow C$ be an arbitrary rule, and be X a given set of samples. The *support* of $A \rightarrow C$ with respect to X is then defined according to:

$$\text{supp}_X(A \rightarrow C) := \text{sim}_X(A, C) = \frac{|A(X) \cap C(X)|}{|A(X) \cup C(X)|} \tag{4}$$

These two measures will be used to ensure a minimal performance of the rules. Generally, though, these two criteria are contrary. The confidence measure prefers local, more accurate rules, while the support measure prefers rules which involve more samples and might have a higher rate of false classifications. It depends on the actual problem definition and the learning algorithm used, how this conflicting goals are balanced and how an optimal solution is found. This balance has to be specified in the actual goal function used to find the optimal ruleset.

2.4 Rule Learning

To obtain the initial set of rules, we use a method which finds all rules fulfilling minimal requirements in terms of confidence and support called *FS-Miner* [11]. Although it is possible to remove rules covering the same range of the data space using a partial ordering structure, we do not use this mechanism as we want to obtain the most comprehensive set of rules. Of course, other rule learning methods might be used as well (e.g. association rule miners [1, 13]).

FS-Miner performs a single beam search to find all rules of interest. It starts with the most general predicate $\top$, the predicate that always gives a truth value of 1 as the only element in the set of candidates. Then all rules within the set of candidates are expanded with all available predicates by means of conjunction. From this set of candidates all rules with a support below the minimum support are removed. From the resulting set of candidates, all rules which have at least minimum confidence are added to the set of potentially maximal rules. Then overlapping rules might be removed from the set of candidates. The most promising rules remaining in the set of candidates are then expanded with a further predicate. This procedure is repeated until the set of candidates is empty, or the maximal rule length is reached. Finally, all non maximal rules are removed from the set of potentially maximal rules to obtain the set of all maximal rules.

3 Bacterial Evolutionary Algorithm

There are several optimization algorithms which were inspired by the processes of evolution. These processes can be easily applied in optimization problems where one individual corresponds to one solution of the problem. An individual can be represented by a sequence of numbers that can be bits as well. This sequence is called *chromosome*, which is nothing else than the individual itself. Bacterial evolutionary algorithms are a recent variant of genetic algorithms based on bacterial evolution rather than eukaryotic. Bacteria share chunks of their genes rather than perform neat crossover in chromosomes, which means bacterial genomes can grow or shrink. This mechanism is used both in the *bacterial mutation* and the

gene transfer operations. The latter substitutes the genetic algorithms crossover operation, so information can be transferred between different individuals. As in this approach many operations can be performed in parallel, it can be adopted to a parallel computing environment in a straightforward manner. In the following, we will restrict ourself to the case of rule selection, although the basic algorithm is applicable to a much broader range of applications.

3.1 The encoding method

In bacterial evolutionary algorithms, one bacterium $\xi_i, i \in I$ corresponds to one solution of the problem under investigation. For the task of selecting m_i rules from a set of n rules ($m_i \leq n$), the bacterium consists of a vector of rule indices $\xi_i = \{\xi_i^1, \ldots, \xi_i^{m_i}\}, 1 \leq \xi_i^k \leq n$ with ξ_i^k being the index of the k-th rule and $\xi_i^k \neq \xi_i^l$ for $k \neq l$. As applying the same rule multiple times is not reasonable, we ensure that each rule occurs only once in each bacterium. This is, however, only a matter of efficiency and not absolutely necessary as by preferring smaller rule sets redundant rules are likely to be removed anyway.

This encoding method, although more complex than a simple binary coding, has strong benefits. First of all this encoding supports the implicit definition of subgroups from not consecutive rules. When using a binary position coding (i.e. a 1 at the i-th position indicates the selection of the i-th rule), subgroups can only evolve amongst neighboring rules. Having subgroups of rules is, however, very important as these subgroups may contain interacting rules with a good overall performance. Furthermore, the evolutionary operations perform block operations which preserve these subgroups. Secondly, we have full control on the number of rules in the rule base. By specifying the length of elements inserted or deleted from the bacterium, we determine the overall number of rules involved. Thus, we can control the broadness of the search. When using a binary position coding, the number of rules is equivalent to the number of 1's, making it much harder to control the overall number of rules in a single step. Alternatively to an integer based encoding one might also use a binary integer coding, where each integer is encoded as a series of bits. Then, however, prevention for dealing with non existing rule numbers have to be made.

3.2 The evaluation function

Similar to genetic algorithms the fitness of a bacterium ξ_i is evaluated using an *evaluation function* $\phi(\xi_i)$. As this evaluation function is computed for all bacteria after each mutation, its efficiency has a major influence on the overall runtime performance of the algorithm.

When a regression problem is only optimized with respect to the overall error the problem occurs, that the number of rules will increase rapidly. Although this effect can be reduced by using a separate test data set, we might want to obtain the best result involving a certain number of rules. As, however, defining the size of the final rule base a-priory is quite restricting, we use a fuzzy predicate which is incorporated in the target function to limit the size of the rule base. Doing so, we

can compute the best result for a given threshold size while still allowing a certain amount of flexibility in the size of the rule base [22].

3.3 The evolutionary process

The basic algorithm consists of three steps [16, 8]. First, an initial population has to be created randomly. Then, bacterial mutation and gene transfer are applied, until a stopping criteria is fulfilled. The evolution cycle is summarized below:

Bacterial Evolutionary Algorithm

1. create initial population
2. do
3. apply bacterial mutation
4. apply gene transfer
5. loop until stopping condition is not fulfilled
6. return best bacterium

3.4 Generating the initial population

First, an initial bacterium population of N_{ind} bacteria $\{\xi_i, i \in I\}$ is created randomly ($I = \{1, \ldots, N_{\text{ind}}\}$). As we do not use a predefined bacteria size, the length of each bacterium is chosen randomly within the range of 2 and m. Figure 1 shows a bacterium ξ_i with $n = 50$ and length 5.

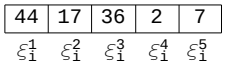

Figure 1: A single bacterium

3.5 Bacterial mutation

To find a global optimum, it is necessary to explore new regions of the search space not yet covered by the current population. This is achieved by adding new, randomly generated information to the bacteria using bacterial mutation.

Bacterial mutation is applied to all bacteria $\xi_i, i \in I$. First, N_{clones} copies (clones) of the bacterium are created. Then, a random segment of length l_c is mutated in each clone. After mutating the same segment in all clones, all the clones and the original bacterium are evaluated using the evaluation function ϕ. The bacterium with the best evaluation result transfers the mutated segment to the other individuals. This step is repeated until each segment of the bacterium has been mutated once. The mutation may not only change the content, but also the length. The length of the new elements is chosen randomly as $l_c \in \{l-l^\star, \ldots, l+l^\star\}$, where $l^\star \geq 0$ is a parameter specifying the maximal change in length. When changing a segment of a bacterium, we must take care that the new segment is

unique within the selected bacterium. At the end, the best bacterium is kept and the clones are discharged. Figure 2 shows an example mutation for $N_{\text{clones}} = 3$ and $l_c = 1$.

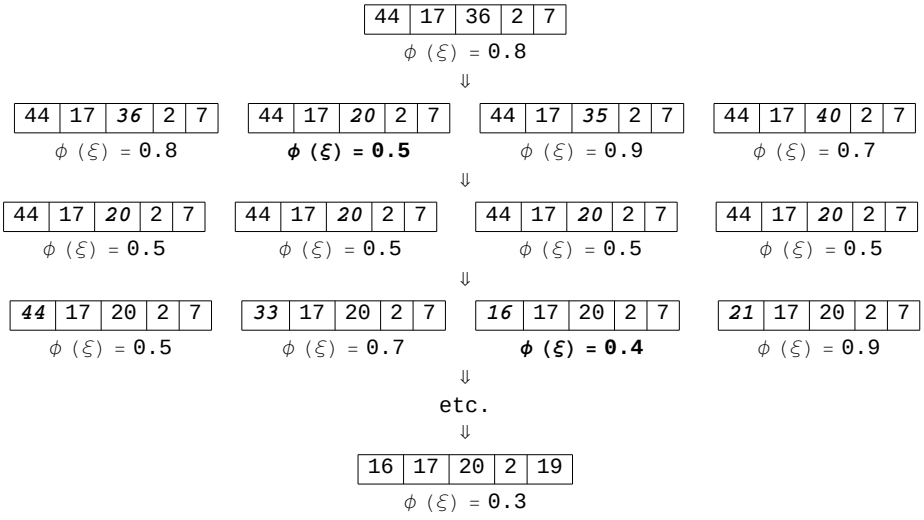

Figure 2: Bacterial mutation

3.6 Gene transfer

The bacterial mutation operator optimizes the bacteria in the population individually. To ensure that information from effective bacteria spreads over the whole population, gene transfer is applied.

First, the population must be sorted and divided into two halves according to their evaluation results. The bacteria with a higher evaluation are called superior half, the bacteria with a lower evaluation are referred to as inferior half. Then, one bacterium is randomly chosen from the superior half and another from the inferior half. These two bacteria are called the source bacterium, and the destination bacterium, respectively. A segment from the source bacterium is randomly chosen and this segment is used to overwrite a random segment of the destination bacterium, if the source segment is not already in the destination bacterium. These two segments may vary in size up to a given length. This ensures—together with the variable length in the bacterial mutation step—that the bacteria are automatically adjusted to the optimal length. Gene transfer is repeated N_{inf} times, where N_{inf} is the number of "infections" per generation. Figure 3 shows an example for the gene transfer operations ($N_{\text{ind}} = 4, N_{\text{inf}} = 3$).

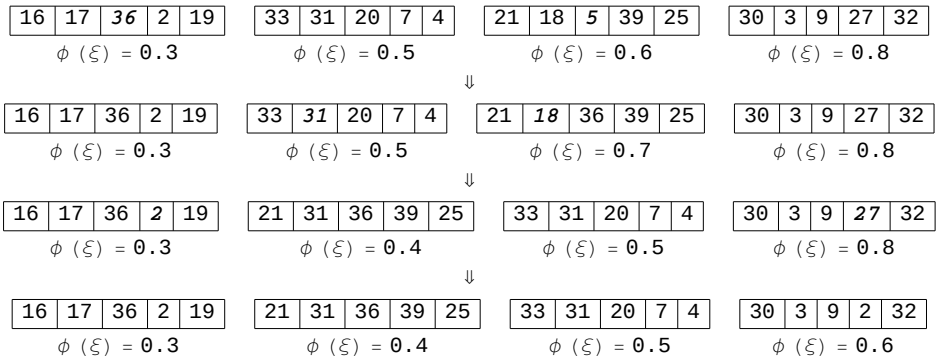

Figure 3: Gene transfer

3.7 Stopping condition

If the maximum number of generations N_{gen} is reached, the algorithm ends, otherwise it returns to the bacterial mutation step. Typically, a small number of generations (below 10) already leads to good results. If a target value for evaluation function exists, a threshold value might be defined alternatively.

4 Simulation results

4.1 Comparison to Genetic Algorithms

To compare the performance of bacterial evolutionary algorithms (BEA) to those of traditional genetic algorithms (GA), we applied both approaches to an artificial learning problem. Given the function (see Fig. 4)

$$f(x) = \frac{1}{2}(1 + \sin(10\pi x^2) + x), \qquad x \in [0, 1],$$

and using a simple binary encoding $x_i = \{x_i^1, \ldots, x_i^n\}, x_i^j \in \{0, 1\}$, the fitness function

$$\phi(x_i) = f\left(\frac{\sum_{j=1}^{n} x_i^j 2^{j-1}}{2^n - 1}\right)$$

was defined. For our tests we used bacteria with a fixed length of $n = 10$, allowing multiple occurrences of the same element within one bacterium. For comparison, we used the genetic algorithm package for Mathematica from Mats G. Bengtsson. It uses simple crossover, selection based on the fitness values, and supports synchronous as well as asynchronous update of the population.

To compare the performance of the two approaches, we generated for the BEA 50 generations of 4 bacteria with 2 clones, mutation length 2 and gene transfer length 2. For the GA we generated 50 generations of 50 genes with a mutation

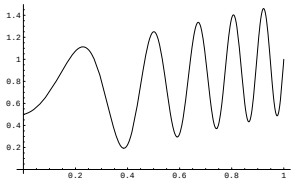

Figure 4: Signal $f(x)$ used for comparison

rate of 2%. The performance was measured as the average performance over all individuals. The performance evaluation is shown in Figure 6. While the BEA reached a high quality level within the first 10 iterations, the GA took significantly longer. All three algorithms made approximatelly 2500 calls to the fitness function.

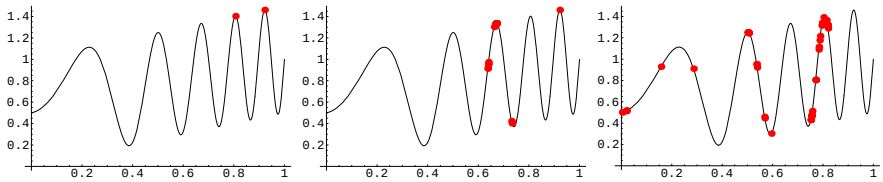

Figure 5: Resulting populations for BEA and GA sync., and GA async.

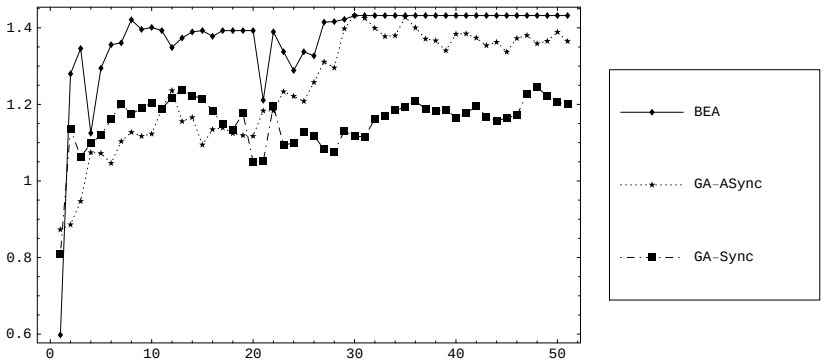

Figure 6: Comparison of BEA and GA

We can see, that the asynchronous genetic algorithm took longer to reach a good overall result, while the synchronous GA failed to reach a higher fitness level. This is mainly due to the large number of individuals in different regions of the search space. The bacterial evolutionary algorithm has a somehow different search strategy, as it explores new regions by creating mutations and selects the best

no. of generations	5
no. of individuals	4
no. of clones	5
mutation length	3 ± 2
no. of gene transfers	20
gene transfer length	2 ± 2
max. length	20

Table 1: Parameter setting for the BEA

individuals and transfers their information to the other bacteria. Thus, a higher increase of the fitness functions can be reached within shorter time. By increasing the size of the population, the diversification can be increased to handle more complex problems, too.

4.2 Regression Learning

4.2.1 Performance Comparison

To compare the performance of our approach with other methods, we used seven data sets from the UCI repository [2]. We compared the results obtained with a fuzzy rule base learner *FS-FOIL*, a fuzzy decision tree learner *FS-ID3*, and a fuzzy regression tree learner *FS-LiRT*—all using the same sets of predicates as used in the BEA rules, where for each attribute, a partition into five fuzzy sets was created automatically. Furthermore, we used three methods from the WEKA 3-4 toolkit [21], namely *M5-Prime* [18], *M5-Rules*, and *SMOreg*. *M5-Prime* and *M5-Rules* generate decision trees and decision rules to solve the regression learning problem. *SMOreg* is an implementation of Alex J.Smola and Bernhard Scholkopf sequential minimal optimization algorithm for training a support vector regression using polynomial or RBF kernels. The methods from the WEKA toolkit do not use a predefined set of predicates/decisions. Furthermore, *FS-FOIL* and *FS-ID3* use predefined linguistic output classes, while all other methods compute individual numeric output values for each rule/leaf. For all these methods we disabled local linear models to ensure equal expressiveness of the underlying language. All tests have been carried out using 5-fold cross validation with identical subsets for all test runs. The results of these tests are shown in Tab. 2 where the average normalized mean error, the average ratio of null predictions, and the average number of rules are printed for each test run. For the BEA we used the parameter setting shown in Table 1.

The evaluation function was defined as follows: For a given data set $S^\star \subset S$ we compute the error estimate of a rule set ξ_i as the normalized mean squared error of the corresponding output function f. To ensure that we do not obtain infinitely large rule sets we define a fuzzy predicate $\mathrm{LE}(s, m)$ according to:

$$\mathrm{LE}(s, \overline{m}) = \begin{cases} 1 & s <= \overline{m} \\ e^{-\frac{1}{2}\left(\frac{\overline{m}-s}{\overline{m}/2}\right)^2} & \text{otherwise} \end{cases}, \tag{5}$$

with $s = \mathrm{MS}(f)$ being the actual model size, and $\overline{m}$ the desired maximum number of rules. The overall error measure $\phi(f, S^\star)$ is then computed as:

$$\phi(f, S^\star) = \frac{\mathrm{mseN}(f, S^\star)}{(1 - \mathrm{RoNP}(f, S^\star))^4 * \mathrm{LE}(\mathrm{MS}(f), \overline{m})}, \tag{6}$$

where

$$\mathrm{mseN}(f, S^\star) = \frac{\sum_{\mathbf{x} \in S^\star} (z(\mathbf{x}) - f(\mathbf{x}))^2}{(\max_{\mathbf{x} \in S^\star} z(\mathbf{x}) - \min_{\mathbf{x} \in S^\star} z(\mathbf{x}))^2},$$

and $\mathrm{RoNP}(f)$ is the ratio of null predictions.

The results obtained using the BEA are in four of seven cases equal or better than those of all other methods. The trade-off, however, is a decreased coverage (i.e. a higher ratio of null predictions). This is caused by the design of the evaluation function ϕ in Equ. 6, where the ratio of null predictions is a divisor of the normalized mean squared error. A high ratio of null predictions as for the servo data set indicates, that a large portion of the data (approx. 30%), shows an "untypical" behaviour and should be analyzed further. Using a different design of the evaluation function could, however, be used to obtain a rule base with a higher coverage, but also a higher MSE.

NMSE **RoNP** **Size**	*FS-FOIL*	*FS-ID3*	*FS-LIRT*	*M5P*	*M5Ru*	*SMOreg*	*BEA*
autoMpg	0.019 0.357 16.6	0.017 0 36.	0.014 0 22.8	0.012 0 16.	0.013 0 11.8	0.009 0 –	0.016 0.096 21.2
autoPrice	0.018 0.071 20.8	0.021 0 34.	0.017 0 12.	0.023 0 8.6	0.023 0 7.4	0.014 0 –	0.014 0.013 20.2
bodyFat	0.003 0.012 7.	0.006 0 5.	0.008 0 3.6	0.007 0 18.6	0.01 0 17.6	0.002 0 –	0.008 0.032 18.6
cloud	0.042 0.038 28.8	0.031 0 25.4	0.044 0 8.2	0.039 0 7.2	0.044 0 6.2	0.013 0 –	0.025 0.048 17.6
housing	0.015 0.154 21.8	0.014 0 21.	0.012 0 17.	0.012 0 18.2	0.016 0 13.	0.013 0 –	0.011 0.048 19.8
servo	0.017 0.364 19.2	0.023 0 44.	0.012 0 9.6	0.028 0 10.6	0.042 0 6.8	0.033 0 –	0.006 0.309 23.
veteran	0.116 0.185 22.	0.084 0 65.4	0.161 0 19.6	0.053 0 1.8	0.055 0 1.8	0.053 0 –	0.051 0.185 20.6

Table 2: Comparison of regression results for 7 UCI data sets

4.2.2 Housing data

Let us take a closer look at the results obtained for the housing data set. Initially 120, 98, 429, 232, and 50 rules (929 in total) were created for each of the five goal classes using *FS-Miner*. The partitioning of the input domains and the definition of the corresponding fuzzy predicates were done using *CompFS*. The partitioning of the goal attribute is shown Fig. 7.

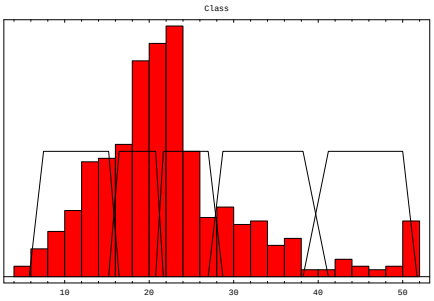

Figure 7: Partitioning of the class parameter in the housing data set

Afterward we applied the BEA to obtain the final rule set. In the first of five trial runs 4, 4, 9, 2, and 1 rules (total 20 rules) involving only two or three predicates have been selected. The resulting rule base is shown in Fig. 8. The additional columns show the number of correctly classified samples (tt), the number of misclassified samples (tf), the number of unclassified samples of the goal class (rt), together with the corresponding confidence and support. Computation took about 90 seconds for each validation cycle, involving about 800 evaluations of the error measure function.

4.3 Classification

4.3.1 Performance Comparison

We also applied the BEA to different classification problems using seven data sets from the UCI repository [2]. The results were compared to five other methods, namely *FS-FOIL*, *FS-ID3*, *J48*, a naive-Bayes classifier, and the *SMO* algorithm for training a support vector classifier using polynomial kernels. *FS-FOIL* and *FS-ID3* use the same set of predicates as the BEA rules, while *J48*, the naive-Bayes classifier, and the *SMO* algorithm were taken from the WEKA 3.4 toolkit [21]. All test were made using 5-fold cross validation with identical splits for all test runs, except for the *SMO* algorithm, which was run separately using 10-fold cross validation.

The error measure was defined in a similar manner as for the regression problem: Given data set $S^\star \subset S$ we compute the error estimate of a rule set ξ_i as the reciprocal of the percentage of correctly classified samples *(FC)* from the corresponding output function f. Again, we use the fuzzy predicate $\mathrm{LE}(s, m)$ to restrict

Class	tt	tf	rt	conf	supp	Condition
class_Is_0_VL	73.38	23.26	31.54	0.76	0.15	LSTAT_IsAtLeast_H && CRIM_IsAtLeast_L
	82.24	28.48	22.78	0.75	0.16	LSTAT_IsAtLeast_H && PTRATIO_IsAtLeast_H
	9.36	2.32	95.34	0.82	0.02	CRIM_IsAtLeast_H && RM_IsAtLeast_H
	26.98	4.97	77.55	0.85	0.05	TAX_Is_VH && B_Is_VL
class_Is_1_L	69.5	43.28	76.85	0.63	0.14	RM_IsAtMost_L && NOX_IsAtMost_M
	26.72	10.26	118.2	0.73	0.05	RM_Is_L && DIS_IsAtLeast_H
	22.43	10.03	122.66	0.71	0.04	LSTAT_Is_M && TAX_Is_H
	61.41	39.59	85.01	0.62	0.12	LSTAT_Is_M && INDUS_IsAtLeast_L
class_Is_2_M	58.82	32.67	90.87	0.66	0.12	RM_Is_M && NOX_IsAtMost_M
	60.26	36.72	89.35	0.63	0.12	RM_Is_M && CRIM_Is_VL
	23.3	11.28	124.78	0.68	0.05	LSTAT_IsAtMost_L && RM_IsAtMost_H && ZN_Is_L
	8.72	4.03	139.09	0.7	0.02	TAX_Is_L && CHAS_Is_1
	26.52	11.25	122.11	0.72	0.05	LSTAT_Is_L && TAX_Is_L && NOX_IsAtMost_L
	5.62	2.65	142.57	0.73	0.01	LSTAT_Is_L && CRIM_Is_L
	8.81	6.08	139.48	0.62	0.02	LSTAT_IsAtLeast_M && AGE_Is_L && B_Is_VH
	54.85	28.96	93.47	0.66	0.11	TAX_Is_L && AGE_IsAtMost_M && RM_IsAtMost_H
	50.	19.18	99.22	0.74	0.1	RM_Is_M && PTRATIO_IsAtMost_M && DIS_IsAtLeast_L
class_Is_3_H	13.31	2.07	60.12	0.89	0.03	ZN_IsAtLeast_L && RM_Is_H && TAX_Is_VL
	31.61	16.75	42.21	0.66	0.06	LSTAT_Is_VL && RM_Is_H && RAD_IsAtMost_H
class_Is_4_VH	9.33	5.69	22.37	0.62	0.02	RM_IsAtLeast_H && PTRATIO_Is_VL && DIS_IsAtMost_L

Figure 8: Result obtained for the housing data set

the size of the rule base. The overall error measure $\phi(f, S^\star)$ is then computed as:

$$\phi(f, S^\star) = 1 - \mathrm{FC}(f, S^\star)(1 - \mathrm{RoNP}(f, S^\star))^4 \mathrm{LE}(\mathrm{MS}(f), \overline{m}). \tag{7}$$

The results of the comparison are shown in Tab. 3.

Not surprisingly, the *SMO* method gave in all but one case the best results as it can cope with much more complex decision boundaries then the (fuzzy) rule based approaches. The BEA, however, obtained (slightly) better results than the other methods in four out of seven cases. This illustrates, that the method is capable of finding at least equally good solutions as other rule based methods, even when a very straight forward error measure is used. Much more interesting, however, is the result for the *balance-scale* data set, where the BEA found a solution having a comparable fraction of correct classifications, but with a significantly lower number of rules. Here, the combination of predictive accuracy and model complexity has resulted in a better overall performance of the BEA. On the other hand, the number of rules which have been allowed without a "penalty" have been also used for the *iris* data set, although the larger number of rules does not increase the performance of the rule base.

4.3.2 Hepatitis Analysis

This test was carried out to illustrate the ability of the proposed approach to solve specific optimization problems. For this study we selected 562 case records of adult hepatitis patients and 231 case records of non-infected patients. The patients received stationary treatment at the Vienna General Hospital (AKH-Wien) between 1976 and 1986. Each patients' clinical diagnoses was serologically verified and thus considered to be a gold standard.

Frac.Corr. **Size**	*FS-FOIL*	*FS-ID3*	*J48*	*Bayes*	*SMO*	*BEA*
balance-scale	0.63 13.	0.781 54.	0.787 35.6	0.878 –	0.877 –	0.771 14.
breast-cancer	0.698 23.2	0.733 18.6	0.73 5.6	– –	0.696 –	0.744 16.2
diabetes	0.741 9.2	0.731 3.6	0.733 22.2	0.599 –	0.773 –	0.749 14.
hepatitis	0.742 11.4	0.832 12.8	0.819 6.8	– –	0.852 –	0.806 14.4
iris	0.947 4.6	0.947 5.	0.94 4.8	0.667 –	0.96 –	0.92 13.
pima	0.741 9.2	0.731 3.6	0.733 22.2	0.599 –	0.773 –	0.749 14.
wine	0.914 8.	0.92 6.6	0.926 5.2	0.651 –	0.983 –	0.949 15.4

Table 3: Comparison of classification results for 7 UCI data sets

For each patient a detailed laboratory analysis was carried out including 11 laboratory measurement and two personal attributes of the patient (gender and age).

For the current experiment, we did not distinguish between the different types of hepatitis, but only between infected (positive) and non-infected (negative) cases to obtain a binary decision problem. From a medical point of view, the goal in this setting is then to optimize the performance of the predictor with respect to specificity and sensitivity under the constraint that the number of rules is limited.

Formally speaking, we compute the error estimate of a rule set $\xi_i, i \in I$ for a given data set $S^\star \subset S$ as the harmonic mean of sensitivity and specificity. To ensure that we do not obtain infinitely large rule sets we define a fuzzy predicate "s is not greater than m" $\mathrm{LE}(s, m)$ according to Equ. 5. The overall error measure $F(f, S^\star)$ is then computed as:

$$F(f, S^\star) = \frac{1}{\mathrm{LE}(\mathrm{MS}(f), \overline{m})} \left(1 - 2\frac{\mathrm{sens}(f, S^\star) + \mathrm{spec}(f, S^\star)}{\mathrm{sens}(f, S^\star) \cdot \mathrm{spec}(f, S^\star)}\right),$$

Other error measure can be used as well, depending on the problem under investigation. For the remaining we will, however, stick to this rather simple error measure to ease interpretation.

We created a rule base of 466 rules having a minimum support of 0.01 and a minimum confidence of 0.6. We then used the BEA to select the optimal subset of these rules using the parameter set shown in Tab. 1.

We also tried out other parameter setting (higher number of generations, individuals, clones), but with no significant increase in the quality of the obtained rule sets. A resulting rule base is shown in Fig. 9. In each iteration, the evaluation function was evaluated approximately 200 times.

We compared the results to those of three different learning algorithms which involve the same set of predicates. This enables us to eliminate influences caused from different definitions of the underlying fuzzy predicates. While *FS-ID3* is a fuzzy decision tree learning method, *FS-FOIL* and *FS-Miner* are rule learning algorithms. We compared the results of the three algorithms using 5-fold cross validation with similar splits for all algorithms. The means of the according quality measures are shown in Tab. 4.

Class		Condition
HEP2_Is_NEG	⇐	ALKPHOS_IsAtLeast_M && GPT_IsAtMost_L && GAMMA_GT_IsAtMost_L && GAMMA_IsAtMost_L && LDH_IsAtLeast_L
		SERBILI_Is_VL && ALKPHOS_Is_VL && BETA_Is_M
		ALKPHOS_IsAtMost_L && LDH_Is_VL && ALBUMIN_IsAtLeast_M
		ALKPHOS_IsAtMost_L && LDH_Is_VL && SEX_Is_W
		ALKPHOS_IsAtLeast_M && GAMMA_GT_Is_VL && GPT_Is_VL && SEX_Is_W && GAMMA_IsAtMost_M
		GPT_Is_VL && ALKPHOS_Is_VL && SERBILI_IsAtMost_L
HEP2_Is_POS	⇐	ALKPHOS_IsAtLeast_M
		LDH_IsAtLeast_M
		GOT_IsAtLeast_M
		ALKPHOS_IsAtLeast_H
		ALKPHOS_Is_VL && AGE_IsAtLeast_L && LDH_IsAtLeast_L && ALPHA2_IsAtMost_L
		ALKPHOS_Is_VL && GAMMA_GT_IsAtMost_H && SEX_Is_M && AGE_IsAtLeast_L && ALPHA1_IsAtMost_H
		GAMMA_GT_IsAtLeast_L && ALBUMIN_IsAtMost_H
		ALKPHOS_Is_VL && GAMMA_GT_IsAtMost_H && AGE_IsAtMost_L && ALPHA2_IsAtMost_L && ALPHA1_IsAtMost_H
		SERBILI_IsAtLeast_L && LDH_IsAtLeast_L
		ALKPHOS_Is_VL && GAMMA_GT_IsAtMost_H && SEX_Is_M && ALPHA2_IsAtMost_L && ALBUMIN_IsAtMost_H
		ALKPHOS_Is_VL && GAMMA_GT_IsAtMost_H && SEX_Is_M && AGE_IsAtLeast_L && ALPHA2_IsAtMost_M && GPT_IsAtMost_H
		ALKPHOS_Is_VL && GAMMA_GT_IsAtMost_H && SEX_Is_M && AGE_IsAtLeast_L && GPT_IsAtMost_H

Figure 9: Rule set obtained using Bacterial Evolutionary Algorithm

>> 5x Cross.Val. <<	**BEA**	**FS-ID3**	**FS-FOIL**	**FS-MINER**
F	0.234	0.28	0.473	1.
Sensitivity	0.759	0.791	0.755	0.861
Specificity	0.781	0.695	0.424	0
PosVals	0.927	0.87	0.915	0.891
NegVals	0.606	0.59	0.651	0
FractionCorrect	0.798	0.766	0.844	0.891
ChiSquared	49.305	34.349	39.533	0
PLevel	0	0	0	0
MutualInformation	0.259	0.167	0.237	0
NormalizedMutualInformation	0.297	0.189	0.304	0
ModelSize	18.2	19.8	7.2	36.
RatioOfNullPredictions	0.038	0	0.219	0.32

Table 4: Comparison of results

We can see, that FS-Miner failed to produce reasonable results, as only positive cases have been predicted. This results in a high sensitivity, but no specificity of the resulting rule base. FS-ID3 has a slightly better performance than the BEA with respect to sensitivity, but with a significantly lower performance regarding specificity. FS-FOIL, finally, is comparable to the BEA in terms of sensitivity, but fails in terms of specificity. The measures *PosVals*, *NegVals*, and *FractionCorrect* show comparable results for all three approaches. Only the χ^2 statistics and the mutual information measure, computed on the predicted and the original output values, are significantly better for the BEA. The main problem of FS-FOIL is its high ratio of negative null predictions (*nullNeg*). This all together leads to a significantly better result of the bacterial evolutionary algorithms with respect to the F measure, the actual goal function of this optimization process.

5 Conclusions

In this paper we have shown how bacterial evolutionary algorithms can be applied to identify the optimal subset of rules for a given learning problem. While traditional top-down approaches usually lack an explicit goal function, separating the rule induction and the rule set definition process enables us to find the optimal combination of rules with respect to a freely definable global goal function. A global goal function has two important benefits: On the one hand side it allows to incorporate global quality criteria like interpretability measures, while on the other hand the interaction of overlapping fuzzy rules can be considered, too. We have shown, that although the underlying language (i.e. the predicates used) is rather simple, we are capable of finding equally good or even better solutions than with traditional top-down approaches.

Although the bacterial evolutionary algorithm (BEA) is quite efficient, the main drawback of this approach is its high computational effort, caused by the complexity of evaluating the according fitness function. Future work will therefore be concerned with implementing a parallel version of the BEA. We hope, that using a parallel implementation we can also solve large real-world applications within reasonable time. Furthermore, we want to target special application areas like medical reasoning, by investigating domain-specific evaluation functions.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. on Very Large Data Bases*, pages 487–499. Morgan Kaufmann, 1994.

[2] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. Univ. of California, Irvine, Dept. of Information and Computer Sciences, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[3] U. Bodenhofer. The construction of ordering-based modifiers. In G. Brewka, R. Der, S. Gottwald, and A. Schierwagen, editors, *Fuzzy-Neuro Systems '99*, pages 55–62. Leipziger Universitätsverlag, 1999.

[4] U. Bodenhofer. *A Similarity-Based Generalization of Fuzzy Orderings*, volume C 26 of *Schriftenreihe der Johannes-Kepler-Universität Linz*. Universitätsverlag Rudolf Trauner, 1999.

[5] U. Bodenhofer and P. Bauer. A formal model of interpretability of linguistic variables. In J. Casillas, O. Cordón, F. Herrera, and L. Magdalena, editors, *Interpretability Issues in Fuzzy Modeling*, volume 128 of *Studies in Fuzziness and Soft Computing*, pages 524–545. Springer, Berlin, 2003.

[6] U. Bodenhofer, M. De Cock, and E. E. Kerre. Openings and closures of fuzzy preorderings: Theoretical basics and applications to fuzzy rule-based systems. *Int. J. General Systems*, 32(4):343–360, 2003.

[7] J. Botzheim, M. Drobics, and L. T. Kóczy. Feature selection using bacterial optimization. In *Proc. 10th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 797–804, Perugia, July 2004.

[8] J. Botzheim, B. Hámori, L. T. Kóczy, and A. E. Ruano. Bacterial algorithm applied for fuzzy rule extraction. In *Proc. Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 1021–1026, Annecy, FR, 2002.

[9] J. Casillas, O. Cordón, F. Herrera, and L. Magdalena, editors. *Interpretability Issues in Fuzzy Modeling*, volume 128 of *Studies in Fuzziness and Soft Computing*. Springer, Berlin, 2003.

[10] M. Drobics. Choosing the best predicates for data-driven fuzzy modeling. In *Proc. 13th IEEE Int. Conf. on Fuzzy Systems*, pages 245–249, Budapest, July 2004.

[11] M. Drobics. *Data Analysis Using Fuzzy Expressions*, volume C 48 of *Schriftenreihe der Johannes-Kepler-Universität Linz*. Universitätsverlag Rudolf Trauner, 2005.

[12] M. Drobics and J. Himmelbauer. Creating comprehensible regression models—inductive learning and optimization of fuzzy regression trees using comprehensible fuzzy predicates. *Soft Computing*, 11:421–438, 2007.

[13] T.-P. Hong, K.-Y. Lin, and S.-L. Wang. Fuzzy data mining for interesting generalized association rules. *Fuzzy Sets and Systems*, 138(2):255–269, 2003.

[14] H. Ishibuchi and T. Yamamoto. Fuzzy rule selection by data mining criteria and genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conf.*, pages 399–406, 2002.

[15] D. Nauck. Measuring interpretability in rule-based classification systems. In *Proc. 12th IEEE Int. Conf. on Fuzzy Systems*, pages 196–201, St. Louis, USA, 2003.

[16] N. E. Nawa and T. Furuhashi. Fuzzy system parameters discovery by bacterial evolutionary algorithm. *IEEE Trans. Fuzzy Syst.*, 7:608–616, 1999.

[17] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138(2):221–254, 2003.

[18] J. R. Quinlan. Learning with Continuous Classes. In *Proc. 5th Austr. Joint Conf. on Artificial Intelligence*, pages 343–348, 1992.

[19] G. Ridgeway, D. Madigan, and T. Richardson. Boosting methodology for regression problems. In *Proc. Artificial Intelligence and Statistics*, pages 152–161, 1999.

[20] M. Setnes and H. Roubos. GA-fuzzy modeling and classification: Complexity and performance. *IEEE Trans. Fuzzy Systems*, 8(5):509–522, October 2000.

[21] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, 2000.

[22] Y. Yi and E. Hüllermeier. Learning complexity-bounded rule-based classifiers by combining association analysis and genetic algorithms. In *Proc. Joint 4th Int. Conf. in Fuzzy Logic and Technology and 11th French Days on Fuzzy Logic and Applications*, pages 47–52, Barcelona, September 2005.