Selecting the Optimal Rule Set Using a Bacterial Evolutionary Algorithm

Mario Drobics
eHealth systems
Biomedical Engineering
Austrian Research Centers GmbH
A-1220 Vienna, Austria
mario.drobics@arcsmed.at

János Botzheim
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
H-1117 Budapest, Hungary
botzheim@tmit.bme.hu

Klaus-Peter Adlassnig
Section on Medical Expert and Knowledge-Based Systems
Core Unit for Medical Statistics and Informatics
Medical University Vienna
A-1090 Vienna, Austria
klaus-peter.adlassnig@meduniwien.ac.at

Abstract

In many regression learning algorithms for fuzzy rule bases it is not possible to define the error measure to be optimized freely. A possible alternative is the usage of global optimization algorithms like genetic programming approaches. These approaches, however, are very slow because of the high complexity of the search space. 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. The evolutionary algorithm tries to find an optimal rule set with respect to a freely definable goal function.

Keywords: Inductive Learning, Rule Selection, Interpretability, Bacterial Evolutionary Algorithm.

1 Introduction

Regression learning is concerned with finding a function $f(\mathbf{x}), \mathbb{R}^n \mapsto \mathbb{R}$ 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 control applications, where expert knowledge is not available, but knowledge of the resulting system is essential [6].

Most methods for learning fuzzy rule bases (or fuzzy regression trees), however, 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). 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. Secondly, it is usually not possible to define the goal function freely. This becomes crucial, when global criteria—like interpretability measures [12]—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 [9, 14].

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

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 [14] or meta-optimization techniques like boosting [16].

Currently, only a few approaches like genetic programming [17] 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 requirements. Comparable approaches for classification problems using genetic algorithms have been presented in [11] and [19]. The main disadvantage of these approaches is their complexity, caused by the use of GAs and a binary 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 [5] and feature selection [4].

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

2 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.

2.1 Generating the initial rule set

In our approach we use a method which finds all rules fulfilling minimal requirements in terms of confidence and support called *FS-Miner* [8]. Although it might be 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, 10]). The underlying set of predicates was defined using *CompFS* [7]. For each attribute, a partition into five fuzzy sets was created automatically. Furthermore, ordering based predicates were defined, too [3].

2.2 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 $k \neq l$.

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 coding, 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 total 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. When using a binary 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.

2.3 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 difficult, 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 [19].

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:

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

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:

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

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.

2.4 The evolutionary process

The basic algorithm consists of three steps [5, 13]. 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. apply bacterial mutation
3. apply gene transfer
4. until stopping condition is not fulfilled, go to 2.
5. return best bacterium

2.5 Generating the initial population

First, an initial bacterium population of N_{ind} bacteria $\{\xi_i, i \in I\}$ is created randomly ($I = \{1,\ldots,N_{\text{ind}}\}$). Figure 1 shows a bacterium ξ_i with $n = 50$ and $m = 5$.

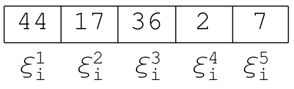

Figure 1: A single bacterium

2.6 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 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 $l \pm l^\star$, where $l^\star$ 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 $l = 1$.

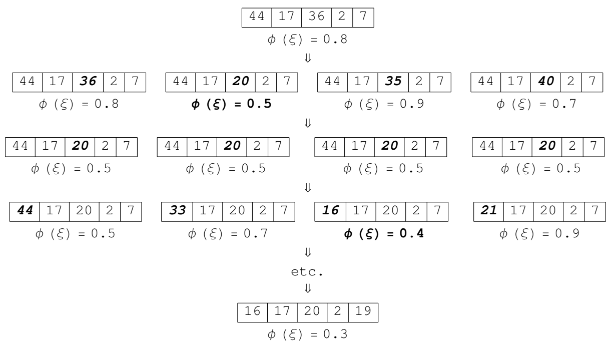

Figure 2: Bacterial mutation

2.7 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 ($N_{\text{ind}} = 4, N_{\text{inf}} = 3$).

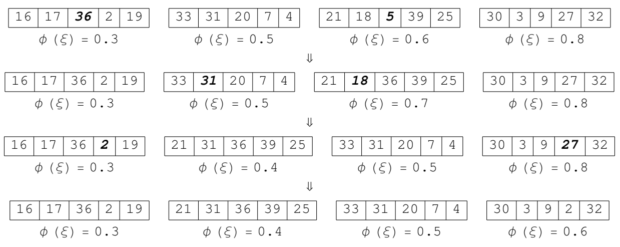

Figure 3: Gene transfer

2.8 Stopping condition

If all individuals in the population are equal or 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.

3 Simulation results

3.1 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. Furthermore, we used two methods from the WEKA 3-4 toolkit [18], namely *M5-Prime* [15], and *M5-Rules*. *M5-Prime* and *M5-Rules* generate decision trees and decision rules to solve the regression learning problem. The latter two methods do not use a predefined set of predicates/decisions but compute the decision boundaries problem specific. 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 Fig. 4 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.

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

The results obtained using the BEA are in five of seven cases better than those of the 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. 1, where the ratio of null predictions is a divisor of the normalized mean squared error. This means, that a decrease of $x\%$ in coverage is equivalent to an increase of $x\%$ of the MSE in terms of the evaluation measure. 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*	*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	0.016 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** 0.0131 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	0.008 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	**0.025** 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.	**0.011** 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	**0.006** 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	**0.051** 0.185 20.6

Figure 4: Comparison of results for 7 UCI data sets

3.2 Housing data

Let us take a closer look to 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 in Fig. 5.

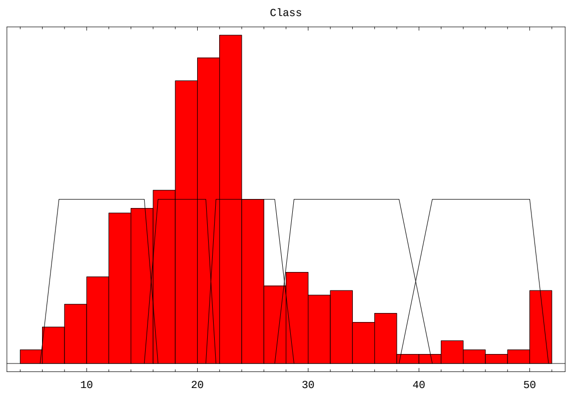

Figure 5: Partitioning of the class parameter in the housing data set

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. 6. 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), and the corresponding confidence and support.

4 Conclusions

In this paper we have shown how bacterial evolutionary algorithms can be applied to identify the optimal subset of rules for a given regression learning problem. Bacterial evolutionary algorithm seems to be more efficient than the standard genetic algorithms. The reason for that is the different nature of the operations in the BEA ([13]). Bacterial mutation is more effective than classical mutation in GA because of the cloning procedure. Every clone brings a new chance to find a better solution anywhere in the search space, thus wider space can be explored. In the gene transfer we do not lose good individuals, because the information flow is directed from the superior sub-population to the inferior one. The algorithm presented is capable of optimizing freely definable goal functions which enables the explicit formulation of interpretability quality measures. We have shown, that although the underlying language (i.e. the predicates used) is very simple, we are often capable of finding better solutions than by traditional top-down approaches.

Future work will 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 study different evaluation functions which allow the user to specify his needs more easily.

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. *Fuzzy Orderings — Theory and Application Perspectives*. Studies in Fuzziness and Soft Computing. Physica-Verlag, Heidelberg.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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, 2006.

[10] 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.

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 6: Result obtained for the housing data set

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

[12] D. Nauck. Measuring interpretability in rule-based classification systems. In *Proc. 3rd IEEE Int. Conf. on Fuzzy Systems*, pages 196–201, Tokyo, October 2003.

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

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

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

[16] G. Ridgeway, D. Madigan, and T. Richardson. Boosting methodology for regression problems. In *Proc. Artificial Intelligence and Statistics*, 1999.

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

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

[19] Y. Yi and E. Hüllermeier. Learning complexity-bounded rule-based classifiers by combining association analysis and genetic algorithms. In *Proc. Joint 4th Conf. of the European Society for Fuzzy Logic and Technology and 11 Recontres Francophones sur la Logique Floue et ses Applications*, Barcelona, September 2005.