Subsampling Conflicts to Construct Better Fuzzy Rules

Michael R. Berthold
Tripos, Inc.
601 Gateway Blvd., Suite 720
South San Francisco, CA 94080, USA
eMail: berthold@tripos.com

Abstract

Many fuzzy rule induction algorithms have been proposed during the past decade or so. Most of these algorithms tend to scale badly with large dimensions of the feature space because the underlying heuristics tend to constrain suboptimal features. Often noisy training instances also influence the size of the resulting rule set. In this paper an algorithm is discussed that extracts a set of so called mixed fuzzy rules. These rules can be extracted from feature spaces with diverse types of attributes and handle the corresponding different types of constraints in parallel. The underlying heuristic minimizes the loss of coverage for each rule when a conflict occurs. We present the original algorithm, which avoids conflicts for each pattern individually and demonstrate how a subsampling strategy improves the resulting rule set, both with respect to performance and interpretability of the resulting rules.

1. Introduction

Building models from data has started to raise increasing attention, especially in areas where a large amount of data is gathered automatically and manual analysis is not feasible anymore. Also applications where data is recorded online without a possibility for continuous analysis are demanding for automatic approaches. Examples include such diverse applications as the automatic monitoring of patients in medicine, optimization of industrial processes, and also the extraction of expert knowledge from observations of their behavior. Techniques from diverse disciplines have been developed or rediscovered recently, resulting in an increasing set of tools to automatically analyze data sets (an introduction to the most important of these techniques can be found in [3]). Most of these tools, however, require the user to have detailed knowledge about the tools' underlying algorithms, to fully make use of their potential. In order to offer the user the possibility to explore the data, unrestricted by a specific tool's limitations, it is necessary to provide easy to use, quick ways to give the user first insights. In addition, the extracted knowledge has to be presented to the user in an understandable manner, enabling interaction and refinement of the focus of analysis.

Learning rules from examples is an often used approach to achieve this goal. Most existing rule learning algorithms are however limited to a uniform type of features [7, 13, 16, 22, 1], in these cases numerical values. Other approaches can only handle a pre-defined partitioning of the numeric features [21], or generate a semi-global partitioning of the feature space, such as decision trees [15, 12]. Very often, the extracted rules also rely on constraints on all available features [18, 19, 11], an approach not feasible for large dimensions. This is similar to clustering techniques which rely on a distance function defined over all dimensions to extract a set of representative prototypes [8]. In order to be able to interpret the results, a rule based representation is usually preferable. More complicated structures offer greater flexibility but are often computationally very inefficient [10, 2].

The approach discussed in this paper can deal with various types of features in parallel and in addition constrains only those features that are needed for each rule individually. Therefore rules in different regions of the feature space can focus on different features, effectively letting each rule decide for itself which features to utilize. In addition, the presented algorithm combines specializing and generalizing rule induction. The resulting rules have an area of evidence as well as an area of support. This leads to a measure of confidence for the area covered by a rule, an important property for real world applications.

One disadvantage of this algorithm is its sequential nature; for each conflict all misclassifying rules are adjusted. This can – especially in high dimensions – lead to a suboptimal set of constraints. In order to avoid this problem we propose a subsampling strategy that assists in finding a better adjustment of the existing constraints to avoid a subset of conflicts. The resulting rules are more general and–as we demonstrate by using the Monk's data [20]–closer to the optimal representation.

0-7803-7078-3/01/$10.00 (C)2001 IEEE.

2. Mixed Fuzzy Rule Induction

2.1. Mixed Fuzzy Rules

Mixed fuzzy rules as used here are rules that handle different types of features. We restrict ourselves to the description of the algorithm with respect to continuous, granulated, and nominal features but other types of features can be handled similarly as well. Each mixed rule is defined through a fuzzy region in the feature space and a class label. (See [5] for a description of a related algorithm in the context of function approximation using fuzzy graphs.)

The *feature space* D consists of n dimensions. Each dimension D_i ($1 \leq i \leq n$) can be one of the following:

- *continuous*, that is $D_i \subset \mathbb{R}$,
- *granulated*, that is $D_i = \{\mu_j \mid 1 \leq j \leq m_i\}$, or
- *nominal*, that is $D_i = \{\text{val}_j \mid 1 \leq j \leq m_i\}$,

where $\mu_j : \mathbb{R} \rightarrow [0,1]$ are the membership functions that specify the used granulation and val_j represent the nominal values.

example 2.1 *A three-dimensional feature space contains a numerical feature 'temperature' in the range* $[0,100]$, *a feature 'pressure' which is divided into two partitions* (μ_{low} *- pressure smaller than 10psi,* μ_{high} *- pressure larger than 10psi), and one feature 'color' which can have three values:* red, green, *and* blue. *This would result in:*

- *dimension* $n = 3$,
- $D_1 = [0,100]$
- $D_2 = \{\mu_{\text{low}}, \mu_{\text{high}}\}$, *where* $\mu_{\text{low}}(x) = 1$ *for* $x << 10$, $\mu_{\text{low}}(x) = 0$ *for* $x >> 10$, *and some transition from* 0 *to* 1 *around* $x = 10$ *(the precise shape of these membership functions is irrelevant for the examples),* μ_{high} *is exactly the opposite in this case, i. e.* $\mu_{\text{high}}(x) = 1 - \mu_{\text{low}}(x)$, *and*
- $D_3 = \{\text{red}, \text{green}, \text{blue}\}$

A *mixed rule* $\mathcal{R}$ operates on a feature space D and is defined through a fuzzy set which assigns a degree of fullfillment. In order to compute this fuzzy set efficiently, two vectors of constraints are used. Vector $\vec{c}^{\,\text{supp}} = (c_1^{\text{supp}}, \cdots, c_n^{\text{supp}})$ describes the most general constraint (the *support* region), whereas $\vec{c}^{\,\text{core}} = (c_1^{\text{core}}, \cdots, c_n^{\text{core}})$ indicates the most specific constraint (the *core* region) for this particular rule. Each one-dimensional constraint c_i defines a subset of the corresponding domain D_i it is responsible for. Constraints can be `true`, that is they do not constrain the corresponding domain at all.

example 2.2 *A rule could be valid for temperatures below* 50, *colors* red *and* blue, *and feature pressure has no influence:*

- $c_1^{\text{supp}} = [0,50) \subset D_1$,
- $c_2^{\text{supp}} = \texttt{true}$, *and*
- $c_3^{\text{supp}} = \{\text{red}, \text{blue}\} \subset D_3$

In addition, let us assume that the available data actually only contained examples for this rule of temperatures in $[20,45]$, *pressures below* 10*psi, and for color* red, *that is:*

- $c_1^{\text{core}} = [20,45] \subset c_1^{\text{supp}}$,
- $c_2^{\text{core}} = \{\mu_{\text{low}}\}$, *and*
- $c_3^{\text{core}} = \{\text{red}\} \subseteq c_3^{\text{supp}}$

Assuming that we already have an entire set of rules we can now classify new patterns. For this, the two different constraints can be used in several ways. Obviously only the specific or more general constraints can be used:

- *optimistic classification*: here the more general support-area of the rule is used:
  $\mathcal{R}(\vec{x}) = \wedge_{i=1}^{n} (x_i \in c_i^{\text{supp}})$
  The disadvantage is a heavy portion of overlap between support regions of rules. This leads to cases where no final classification is possible because rules of several different classes are activated.
- *pessimistic classification*: the smaller, more specific core region of the rule is used:
  $\mathcal{R}(\vec{x}) = \wedge_{i=1}^{n} (x_i \in c_i^{\text{core}})$
  The disadvantage here is that a large area of the feature space is not covered and - similar to the above case - no decision can be made.

But it is obviously much more desirable to combine the two constraints, resulting in a degree of membership for each rule. This solves the problem in areas of heavy overlap or no coverage at all.

- *fuzzy classification*: Here we compute a degree of match for each rule and a corresponding input pattern $\vec{x}$. The combination of one-dimensional membership values can for example be done using as T-norm the minimum-operator:

$$\mu(\mathcal{R}, \vec{x}) = \min_{i=1}^{n} \{\mu_i\{c_i^{\text{supp}}, c_i^{\text{core}}, x_i\}\}$$

  where the particular form of $\mu_i()$ depends on the type of domain D_i. For the choice of membership

functions various alternatives exist. For the nominal features one could simply assign the maximum degree of membership for patterns that fell inside the core region and the minimum degree of membership to the ones that only lie in the support region. One could also use an underlying onthology and actually compute a degree of match between the constraint and the input vector. For the granulated features pre-defined fuzzy membership functions can be used which assign degrees of membership to input patterns. And for the numerical domains most commonly a trapezoidal membership function is used which assigns values of 1 to patterns that fall inside the core region and linearly declines until it reaches 0 when they fall outside of the support region of the corresponding rule.

For the benchmark comparisons in the following sections, a winner-take-all scenario was used, that is, the class with maximum degree of membership was assigned as prediction to a new pattern.

2.2. Induction of Mixed Fuzzy Rules

The extraction of mixed rules as described above from example data is done by a sequential, constructive algorithm. Each pattern is analyzed subsequentially and rules are inserted or modified accordingly[1]. Several such epochs (that is, presentations of all patterns of the training set) are executed until the final rule set agrees with all patterns. In normal scenarios this stable status is reached after only few epochs, usually around five. An advantage over many other algorithms is the clear termination criterion as well as the possibility to prove formally that the algorithm does indeed terminate for a finite training set.

Let us now concentrate on the underlying behavior of the rule induction algorithm. For internal use each rule maintains two additional parameters:

- a weight w which simply counts how many patterns are explained by this particular rule, and
- a so-called anchor $\vec{\lambda}$ which remembers the original pattern that triggered creation of this rule.

For each pattern $(\vec{x}, k)$, where $\vec{x}$ is the input vector and k indicates the corresponding class[2], three cases are distinguished:

- **covered**: a rule of the correct class k exists which covers this pattern, that is, pattern $\vec{x}$ lies inside the support region specified by the vector of constraints $(c_1^{\text{supp}}, \cdots, c_n^{\text{supp}})$. That is, pattern $\vec{x}$ has a degree of membership greater then 0 for this rule. This fact will be acknowledged by increasing the core region of the covering rule, in case it does not already cover $\vec{x}$ which in effect increases the degree of membership to 1. In addition this rule's weight w is incremented.

  example 2.3 *If the rule from example 2.2 encounters another pattern* $\vec{x} = (15, 5, \text{blue})$ *(which is obviously covered by the support region of the rule), the core regions for* x_1 *and* x_3 *would need to be adjusted as follows:* $c_1^{\text{core}} = [15, 45]$ *and* $c_3^{\text{core}} = \{\text{red}, \text{blue}\}$.

- **commit**: If no rule of correct class k exists which covers pattern $\vec{x}$, a new rule needs to be inserted into the rule base. This rule's support region will initially cover the entire feature space, that is, $c_i^{\text{supp}} = \texttt{true}$ for all $i = 1, \ldots, n$. The core region will only cover $\vec{x}$ itself, that is, $c_i^{\text{core}} = [x_i, x_i]$ for numerical features, $c_i^{\text{core}} = \{x_i\}$ for nominal features, and in case of granulated features, the one partition which covers the component best will appear in the constraint. The new rule's weight w is set to 1 and the anchor is set to remember the original pattern $\vec{\lambda} = \vec{x}$.

  example 2.4 *The rule from the example above encounters another pattern* $(5, 5, \text{green})$*, which is obviously not covered by the existing rule. A new rule will therefore be created, having an unconstrained support region:* $c_1^{\text{supp}} = c_2^{\text{supp}} = c_3^{\text{supp}} = \texttt{true}$*, and a specific core region which covers only the new pattern:* $c_1^{\text{core}} = [5, 5]$, $c_2^{\text{core}} = \{\mu_{\text{low}}\}$, $c_3^{\text{core}} = \{\text{green}\}$.

- **shrink**: For both of the above cases, a third step is used to ensure that no existing rule of conflicting class $l \neq k$ covers $\vec{x}$. This is done by reducing the support regions $\vec{c}^{\,\text{supp}}$ for each rule of class $l \neq k$ in such a way that $\vec{x}$ is not covered by the modified rule, i.e. results in a degree of membeship of 0. We can distinguish two cases:

  - $\vec{x}$ lies inside the support region, but outside of the core region: $\vec{x} \in \vec{c}^{\,\text{supp}}$ and $\vec{x} \notin \vec{c}^{\,\text{core}}$. In this case we can avoid the conflict without loosing coverage of previous patterns. We simply reduce the support area just enough so that $\vec{x}$ is not covered anymore. For this, all features for which the corresponding component of $\vec{x}$ does not lie in it's core region are considered. From those features, the one is

[1] Later in this paper we will discuss how a subsampling procedure can improve the performance of this pattern-by-pattern approach.

[2] The presented algorithm can also be used to handle different degrees of membership to several classes, for simplicity we concentrate on mutually exclusive classes. In [5] it is shown how overlapping classes can be used in the contect of function approximation, however.

chosen that results in a minimal loss of volume. This constraint is then modified accordingly.

example 2.5 *Let us consider the rule in example 2.3. If the next pattern* $\vec{x} = (10, 20, \text{red})$ *is of different class, this rule needs to be refined to avoid the resulting conflict. In this case it is sufficient to alter the support region. For this we have two choices, either* c_1^{supp} *or* c_2^{supp} *can be modified (*c_3^{supp} *is not an option since* $\text{red} \in c_3^{\text{core}}$*):* $c_1'^{\text{supp}} = c_1^{\text{supp}} \backslash [0, 10] = (10, 50)$*, or* $c_2'^{\text{supp}} = c_2^{\text{supp}} \backslash \{\mu_{\text{high}}\} = \{\mu_{\text{low}}\}$*. The choice between these two alternatives is made based on the respective loss in volume.*

- $\vec{x}$ lies inside the support region and inside of the core region: $\vec{x} \in \vec{c}^{\,\text{supp}}$ and $\vec{x} \in \vec{c}^{\,\text{core}}$. In this case it is not possible to avoid the conflict without loosing coverage of previous patterns[3]. Similar to the above solution, one feature is chosen that results in a minimal loss of volume and both, the support and the core region are modified accordingly.

  example 2.6 *Let us again consider the rule in example 2.3. If the next pattern* $\vec{x} = (25, 5, \text{red})$ *is of different class, this rule needs to be refined to avoid the resulting conflict. In this case it is not sufficient to alter the support region since* $\vec{x}$ *lies inside the core region as well. Now we have three choices. For feature* 1 *two choices exist, the support region can be constrained either on the left or right side:* $c_1'^{\text{supp}} = c_1^{\text{supp}} \backslash [0, 25] = (25, 50)$*, or* $c_1''^{\text{supp}} = c_1^{\text{supp}} \backslash [25, 50) = [0, 25)$*. Feature* 2 *does not allow us to avoid the conflict since we would create an empty constraint, thus rendering this rule useless. Feature* 3 *can be used since still two nominal values are contained in the core region:* $c_3'^{\text{supp}} = c_3^{\text{supp}} \backslash \{\text{red}\} = \{\text{blue}\}$*. The choice between these three alternatives is again made based on the respective loss in volume.*

In both cases the loss in volume needs to be computed. Since we are dealing with disjunctive constraints, the resulting computation is straight forward. The volume of a rule $\mathcal{R}$ is specified by the volumes of the core and support regions:

$$\text{vol}(\mathcal{R}) = (\text{vol}(\vec{c}^{\,\text{supp}}), \text{vol}(\vec{c}^{\,\text{core}}))$$

where the volume of a constraint can be computed as follows:

$$\text{vol}(\vec{c}) = \prod_{i=1}^{n} \text{vol}(c_i)$$

with

$$\text{vol}(c_i) = \begin{cases} 1 & : c_i = \texttt{true} \\ \frac{c_i.\max - c_i.\min}{D_i.\max - D_i.\min} & : D_i \text{ is numeric} \\ \frac{|c_i|}{|D_i|} & : D_i \text{ is granulated or nominal} \end{cases}$$

Obviously other choices are possible as well. Using a volume based heuristic ensures that the resulting rules cover as much as possible of the feature space. But one could, for example, also include a weighting scheme that prefers constraints on certain features or use a built-in preference for certain types of constraints. Note that in the case described above, the algorithm is based on a greedy strategy. What results in a minimal loss of volume for one conflicting pattern at a time might not be a good solution for the overall set of conflicts. Further below we will discuss how a subsampling of conflicts can address this issue.

After presentation of all patterns for one epoch, all rules need to be reset. This done by reseting the core-region of each rule to it's anchor (similar to the original commit-step), but maintaining it's support region and by reseting it's weight to 0. This is necessary to ensure that patterns that are not covered by a rule anymore (due to subsequent shrinks) only model patterns in their core and weight that they cover with their modified support region. This also solves problems with cores that are bigger than their corresponding support. After the final epoch this effect is impossible.

After presentation of all patterns for a (usually small) number of epochs, the rule set will stop to change and training can be terminated. It is actually possible to prove that the algorithm will terminate guaranteed, for a finite set of training examples. A worst-case analysis finds that the maximum number of epochs is equivalent to the number of training examples, but in practice less than 10 epochs are almost always sufficient to reach equilibrium of the rule set.

2.3. Experimental Results

The evaluation of the proposed methodology was conducted using eight data sets from the StatLog project [14] and the results are reported in [4]. As usual, the new method does not outperform existing algorithms on every data set. Depending on the nature of the problem, the mixed rule induction method performs better, comparable, and sometimes also worse than existing methods.

Two data sets are worth taking a closer look at, however. For the Shuttle data set (9 features, 7 classes, 43,500 training instances, 14,500 test cases) the proposed methodology achieves results that are substan-

[3] Those patterns will result in creation of a new rule during subsequent epochs.

tially better than any of the other algorithms, in fact, the new algorithm has a better generalization performance than all techniques evaluated in the StatLog project. This is due to the axes parallel nature of the generated rules. The Shuttle data set has one class boundary where patterns of two different class lie arbitrarily close to an axes parallel border. Such a scenario is modeled well by the underlying rules. However, for the DNA data set (180 features, 3 classes, 2,000 training instances, 1,186 test cases) the proposed algorithm generates a rule set which performs substantially worse than all other methods. This is an effect due to the used heuristic to avoid conflicts. In case of the DNA data set almost 60% of all features are useless, and, even worse, exhibit random noise. This leads the conflict avoidance heuristic to choose features to constrain almost randomly. The resulting rule set consists of almost 1.500 rules, a clear indication that no generalization took place. For such a scenario the underlying heuristic would obviously need to be adjusted.

In the context of rule extraction, pure numerical performance is, however, very often not the only concern. In the following we will demonstrate how the use of granulated features can result in rule sets that enable the user to understand the structure of the extracted model.

Using the well known Iris data set [9] we can nicely demonstrate how feature granulation will in fact guide the rule extraction process. If all four feature are granulated into three equidistant linguistic values "low", "medium", and "high", the proposed algorithm finds seven rules. In the following we list the three rules with the highest weight, all together covering over 90% of all example patterns[4]:

```
R1(25): if petal-length is low
          then class iris-setosa
R2(24): if   petal-length is medium
          and petal-width is (low or medium)
          then class iris-virginica
R3(21): if   petal-length is (medium or high)
          and petal-width is high
          then class iris-versicolor
```

The other four rules describe the remaining five patterns by using the other two features `sepal-length` and `-width`. From the UCI repository [6] it is known that the features regarding the petal size carry most of the class-discriminative information, which is nicely complemented by the above result and can also be seen when analyzing the underlying model itself [17].

[4]The number in brackets following the rule symbol denotes the number of patterns covered by this rule. In case of the used Iris data set, each class consists of 25 patterns.

3. Subsampling Conflicts

As was shown in [4], some data sets result in very large rule sets or relatively low generalization performance. This is obviously due to the inductive bias of the proposed algorithm but also partly due to the used heuristic which avoids conflicts based purely on one single, conflicting example pattern. In subsequent experiments, subsampling of conflicts was explored. For this, each rule maintains a small list of individual conflicts and tries to solve as many of them as possible when a certain threshold is reached. Our experiments showed good results even for rather small thresholds (sampling $5 - 10$ conflicts often seems enough to achieve considerably better performance for smaller rule sets). For illustration, we discuss experiments on the Monks data [20]. The task here is to extract rules from data which was generated according to predefined rules. The data sets are based on six nominal attributes with values $1, 2, 3, 4$ (not all attributes use all four nominal values). The first monk's problem is defined by the underlying concept:

```
MONK-1: (attr1 = attr2) or (attr5 = 1)
```

and the third[5] monk's problem is based on the concept[6]:

```
MONK-3: (attr5  = 3 and attr4  = 1) or
        (attr5 != 4 and attr2 != 3)
```

It is interesting to see what rule sets are generated by the initial algorithm which avoids individual conflicts. For the first monk's problem 7 rules are generated describing the underlying concept. The first two rules look as follows:

```
R1: if     attr1 is (1 or 3) and attr2 is 1
       and attr4 is (1 or 3)
       and attr5 is (1 or 3 or 4)
           then class 1
R2: if attr5 is 1 then class 1
```

so, even though R2 nicely describes the second part of the condition (`attr5=1`), R1 only describes a special case of the first part. This is due to the sequential nature of the algorithm, which in this particular case chose to avoid a conflict by restricting `attr4` instead of `attr1` or `attr2`. If one changes the conflict-avoidance heuristic to subsample twenty conflicts before a decision is being made, the following four rules are extracted:

```
R1: if attr1 is 1 and attr2 is 1 then class 1
R2: if attr1 is 3 and attr2 is 3 then class 1
R3: if attr1 is 2 and attr2 is 2 then class 1
R4: if attr5 is 1 then class 1
```

which is indeed the optimal representation of the underlying concept.

[5]The second monk's problem is not discussed here, since it's underlying concept is harder to represent using only disjunctive rules. The results for that problem are similar, however.

[6]For illustrative purposes we ignore the 5% additional noise in the training set that are usually used for this problem. In [4] we discussed how an approach to tolerate outliers can address noisy data.

The same applies to the third monk's problem. Without conflict subsampling 7 rules are generated. When conflicts are avoided based on a subsampling of 20 conflicts, this reduces to the following two rules, which again are optimal:

```
R1: if attr4 is 1 and attr5 is 3 then class 1
R2: if     attr2 is (1 or 2)
       and attr5 is (1 or 2 or 3)
           then class 1
```

A subsampling of conflicts obviously leads to a reduction of the rule set. In the two cases shown above, the modified algorithm in fact retrieves the true underlying concepts.

4. Conclusions

We have extended a recently presented method for rule induction. The generated rules handle different types of attributes and through their individual assignment of constraints it is possible to extract these rules also from high-dimensional data sets – the resulting rule will only use a small individual subset of features which were considered important in the particular part of the feature space. A method to improve the underlying online heuristic was presented that operates by subsampling conflicts in order to make better decisions about local feature importance. We demonstrated how the interpretability of the extracted rules improves using the iris and monks data.

Acknowledgments

This research was carried out while the author was with the Berkeley Initiative in Soft Computing (BISC) at UC Berkeley and was supported by stipend Be1740/7-1 of the "Deutsche Forschungsgemeinschaft" (DFG). The author thanks Prof. Lotfi A. Zadeh and his group for his support and the opportunity for many stimulating discussions.

References

[1] S. Abe and M.-S. Lan. A method for fuzzy rules extraction directly from numerical data and its application to pattern classifiction. *IEEE Transactions on Fuzzy Systems*, 3(1):18–28, 1995.

[2] S. Abe and R. Thawonmas. A fuzzy classifier with ellipsoidal regions. *IEEE Transactions on Fuzzy Systems*, 5(3):358–368, 1997.

[3] M. Berthold and D. J. Hand, editors. *Intelligent Data Analysis: An Introduction*. Springer Verlag, 1999.

[4] M. R. Berthold. Learning fuzzy models and potential outliers. In *Computational Intelligence in Data Mining*, pages 111–126. Springer-Verlag, 2000.

[5] M. R. Berthold and K.-P. Huber. Constructing fuzzy graphs from examples. *Intelligent Data Analysis*, 3(1):37–54, 1999. (http://www.elsevier.nl/locate/ida).

[6] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. at ics.uci.edu in pub/machine-learning-databases, 1998.

[7] P. Clark and T. Niblett. The CN2 induction algorithm. In *Machine Learning*, 3, pages 261–283, 1989.

[8] R. Davé and R. Krishnapuram. Robust clustering methods: A unified view. *IEEE Transactions on Fuzzy Systems*, 5(2):270–293, May 1997.

[9] R. A. Fisher. The use of multiple measurements in taxonomic problems. In *Annual Eugenics, II*, 7, pages 179–188. John Wiley, NY, 1950.

[10] A. B. Geva. Hierarchical unsupervised fuzzy clustering. *IEEE Transactions on Fuzzy Systems*, 7(6):723–733, Dec. 1999.

[11] C. M. Higgins and R. M. Goodman. Learning fuzzy rule-based neural networks for control. In *Advances in Neural Information Processing Systems*, 5, pages 350–357, California, 1993. Morgan Kaufmann.

[12] C. Janikow. Fuzzy decision trees: Issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 28(1):1–14, 1998.

[13] R. S. Michalski, I. Mzetic, J. Hong, and N. Lavrac. The multipurpose incremental learning system AQ15. In *Proceedings of the National Conference on AI, AAAI*, 5, pages 1041–1045, 1986.

[14] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Limited, 1994.

[15] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[16] S. Salzberg. A nearest hyperrectangle learning method. In *Machine Learning*, 6, pages 251–276, 1991.

[17] R. Silipo and M. R. Berthold. Input features impact on fuzzy decision processes. *IEEE Transcation on Systems, Man, and Cybernetics, Part B: Cybernetics*, 30(6):821–834, 2000.

[18] P. K. Simpson. Fuzzy min-max neural networks – part 1: Classification. *IEEE Transactions on Neural Networks*, 3(5):776–786, Sept. 1992.

[19] P. K. Simpson. Fuzzy min-max neural networks – part 2: Clustering. *IEEE Transactions on Fuzzy Systems*, 1(1):32–45, Jan. 1993.

[20] S. B. Thrun. The MONK's problems – a performance comparison of different learning algorithms. Technical report, Carnegie Mellon University, Pittsburgh, PA, December 1991.

[21] L.-X. Wang and J. M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1313–1427, 1992.

[22] D. Wettschereck. A hybrid nearest-neighbour and nearest-hyperrectangle learning algorithm. In *Proceedings of the European Conference on Machine Learning*, pages 323–335, 1994.