Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER International Journal of Approximate Reasoning 32 (2003) 67–84

INTERNATIONAL JOURNAL OF APPROXIMATE REASONING

www.elsevier.com/locate/ijar

Mixed fuzzy rule formation

Michael R. Berthold *

Data Analysis Research Lab, Tripos Inc., 601 Gateway Blvd., Suite 720, South San Francisco, CA 94080, USA

Received 1 January 2002; accepted 1 April 2002

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 and in addition have trouble dealing with different feature types or noisy data. In this paper, an algorithm is proposed 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 extracted rules depend on individual subsets of only few attributes, which is especially useful in high dimensional feature spaces. The algorithm along with results on several classification benchmarks is presented and how this method can be extended to handle outliers or noisy training instances is sketched briefly as well.

Keywords: Fuzzy rules; Rule formation; Rule induction; Mixed rules; Explorative data analysis; Data mining; Outliers; Model hierarchy

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 are recorded online without a possibility for continuous analysis are demanding for automatic approaches. Examples include such diverse applications as the automatic

* Tel.: +1-6505538004; fax: +1-6505538005.
E-mail address: berthold@tripos.com (M.R. Berthold).

0888-613X/02/$ - see front matter
PII: S0888-613X(02)00077-4

monitoring of patients in medicine (which requires an understanding of the underlying behavior), 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 [1]). 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. However, most existing rule learning algorithms are limited to a uniform type of features [2–6], in these cases numerical values. Other approaches can only handle a pre-defined partitioning of the numeric features [7], or generate a semi-global partitioning of the feature space, such as decision trees [8–11]. Very often, the extracted rules also rely on constraints on all available features [12–15], 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 [16]. Approaches to extract fuzzy rules from clusters have also been proposed [17] but they have similar problems, that is, the resulting rules are constrained on all available features and a distance metric defined over all dimensions is required, which again makes it hard to apply this type of techniques to feature spaces with diverse types of attributes. However, in order to be able to interpret the results, such rule based representations are usually preferable. More complicated structures offer greater flexibility but are often computationally very inefficient [18,19].

The approach presented in this paper can deal with various types of features in parallel (in [20] the term *mixed rules* was introduced for rules of this type) 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. In effect, the algorithm traverses the version space (see [21,22] for a detailed introduction) from the top (in that it specializes its rule set) and through a smaller part of the version space also from the bottom (in that it generalizes within each rule as well). Therefore the resulting rules have an area of evidence as well as an area of support. Both constraints together lead to a measure of confidence for the area covered by a rule, an important property for real world applications.

An additional problem that severely affects the performance of many rule induction algorithms are outliers or distorted attributes. They heavily interfere

with the goal to extract meaningful representations. Most methods to deal with outliers try to completely ignore them, which can be potentially harmful since the very outlier that was ignored might have described a rare but still extremely interesting phenomena.

To address this problem we also describe an extension to the proposed algorithm that aims to build a compact and interpretable model while still maintaining all the information in the data. This is achieved through a two stage process. A first phase builds an outlier-model for data points of low relevance, followed by a second stage which uses this model as filter and generates a simpler model, describing only examples with higher relevance, thus representing a more general concept. The outlier-model on the other hand may point out potential areas of interest to the user. Experiments indicate that the two models in fact have lower complexity and sometimes even offer superior performance.

The remainder of this paper is organized as follows: in Section 2 we introduce the concept of mixed rules and describe the basic algorithm, followed by results on some well-known benchmark data sets. We continue by describing some aspects of the algorithm such as subsampling conflicts (Section 3) and detection of potential outliers (Section 4). After a brief conclusion (Section 5) Section 6 describes some potential extensions of this work.

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 [23] 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 ($1 \leqslant i \leqslant n$) can be one of the following:

- *continuous*, that is $D_i \subset \mathbb{R}$,
- *granulated*, that is $D_i = \{\mu_j \,|\, 1 \leqslant j \leqslant m_i\}$, or
- *nominal*, that is $D_i = \{val_j \,|\, 1 \leqslant j \leqslant m_i\}$,

where $\mu_j : \mathbb{R} \to [0,1]$ 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 10 psi, μ_{high} – pressure larger than 10 psi), 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 $x \ll 10$, $\mu_{\text{low}}(x) = 0$ for $x \gg 10$, 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* **R** operates on a feature space D and is defined through a fuzzy set which assigns a degree of fulfillment. In order to compute this fuzzy set efficiently, two vectors of constraints are used. Vector $\vec{c}^{\text{supp}} = (c_1^{\text{supp}}, \ldots, c_n^{\text{supp}})$ describes the most general constraint (the *support* region), whereas $\vec{c}^{\text{core}} = (c_1^{\text{core}}, \ldots, c_n^{\text{core}})$ 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 "temperature" 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:

  $$\mathbf{R}(\vec{x}) = \bigwedge_{i=1}^{n} (x_i \in c_i^{\text{supp}}).$$

  The disadvantage is a heavy portion of overlap between support regions of different 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:

$$\mathbf{R}(\vec{x}) = \bigwedge_{i=1}^{n} (x_i \in c_i^{\text{core}}).$$

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.

Hence 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*. Compute a degree of match for each rule and a corresponding input pattern $\vec{x}$. One possibility to combine the one-dimensional membership values is using as T-norm the minimum-operator:

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

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 fall 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 ontology 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 from data

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 (i.e., 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.

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

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 $(c_1^{\text{supp}}, \ldots, c_n^{\text{supp}})$. 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,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 membership of 0. We can distinguish two cases:

[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 [23] it is shown how overlapping classes can be used in the context of function approximation, however.

- $\vec{x}$ 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 its core region are considered. From those features, the one is 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 red $\in c_3^{\text{core}}$): $c_1'^{\text{supp}} = c_1^{\text{supp}} \setminus [0, 10] = (10, 50)$, or $c_2'^{\text{supp}} = c_2^{\text{supp}} \setminus \{\mu_{\text{high}}\} = \{\mu_{\text{low}}\}$. 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: $c_1'^{\text{supp}} = c_1^{\text{supp}} \setminus [0, 25] = (25, 50)$, or $c_1''^{\text{supp}} = c_1^{\text{supp}} \setminus [25, 50) = [0, 25)$. 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 two nominal values are still contained in the core region: $c_3'^{\text{supp}} = c_3^{\text{supp}} \setminus \{\text{red}\} = \{\text{blue}\}$. 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 **R** is specified by the volumes of the core and support regions:

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

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

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

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

with

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

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 is done by resetting the core-region of each rule to its anchor (similar to the original commit-step), but maintaining it's support region and by resetting its weight to 0. This is necessary to ensure that modified rules only model patterns in their core and weight that they cover with their modified support region. This also solves potential problems with cores that are bigger than their corresponding support. After the final epoch this effect is not possible.

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 [24]. Table 1 shows the properties of these data sets as well as the results of the proposed algorithm (column MRL = mixed rule learner) in comparison to other, well-known classification techniques (results from [24,25]). In addition to k nearest neighbor, a multi-layer perceptron, and the decision tree algorithm c4.5 [9], we have used a constructive training algorithm for probabilistic neural networks [25] (column DDA–PNN) to enable comparison with another local, constructive algorithm. As usual, the new method does not outperform existing algorithms on every

Table 1
The used data sets along with error rates (in percent) for the proposed algorithm and other well-known classification methods (results from [24,25])

Data set	No. of features	No. of classes	Size of data set		MRL	DDA (PNN)	Other results		
			Training	Test			kNN	C4.5	MLP
Diabetes	8	2	768	12-fold	27.7	24.1	32.4	27.0	24.8
Aust. Cred.	14	2	690	10-fold	19.1	16.1	18.1	15.5	15.4
Vehicle	18	4	846	9-fold	33.9	29.9	27.5	26.6	20.7
Segment	11	7	2310	10-fold	4.0	3.9	7.7	4.0	5.4
Shuttle	9	7	43,500	14,500	0.03	0.12	0.44	0.10	0.43
SatImage	36	6	4435	2000	14.3	8.9	9.4	15.0	13.9
DNA	180	3	2000	1186	28.6	16.4	14.6	7.6	8.8
Letter	16	26	15,000	5000	11.5	6.4	6.8	13.2	32.7

data set. Depending on the nature of the problem, the mixed rule induction method performs better, comparable, and sometimes also worse than existing methods.

It is interesting to see that for the Shuttle data set, the proposed methodology achieves results that are substantially 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 classes lie arbitrarily close to an axes parallel border. Such a scenario is modeled well by the underlying rules. But also for the other data sets the performance is comparable to standard algorithms. Only for the DNA data set does the proposed algorithm generate a rule set which performs substantially worse than all other methods. This is an effect due to the used heuristic for avoiding 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 1500 rules, a clear indication that no generalization took place. For such a scenario, the underlying heuristic would obviously need to be adjusted. A similar effect might cause the difference in performance for the vehicle data, where the algorithm discussed here is outperformed by a multi-layer perceptron but performs comparable to the other methods listed in Table 1. A more thorough analysis such as the one in [26] might help to investigate which characteristics of a data set are well suited to be modeled by the proposed technique.

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.

2.4. Using granulated features

Using the Iris data set [27], we will demonstrate how feature granulation can guide the rule extraction process. If all four features 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 training patterns: [4]

```
R1(25): if petal-length is low then class iris-setosa
R2(24): if petal-length is medium
```

[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. The other $3 * 25 = 75$ patterns were reserved for testing.

```
    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 two features `sepal-length` and `sepal-width` as well. From the UCI repository [28], 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.

3. Subsampling conflicts

As was visible in the previous section, some data sets result in either 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. Preliminary experiments showed promising results even for rather small thresholds (sampling five or 10 conflicts often seems enough to achieve much better generalization performance using smaller rule sets). For illustration, we discuss experiments on the Monks data [29]. 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: (attr_1 = attr_2) or (attr_5 = 1)
```

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

```
MONK-3: (attr_5 = 3 and attr_4 = 1) or
        (attr_5 != 4 and attr_2 != 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, seven rules are generated describing the underlying concept. The first two rules look as follows:

[5] The second monk's problem is not discussed here, since its 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 the following section, we will discuss how an approach to tolerate outliers can address noisy data.

```
R1: if attr_1 is (1 or 3) and attr_2 is 1
  and attr_4 is (1 or 3)
  and attr_5 is (1 or 3 or 4)
        then class 1
R2: if attr_5 is 1 then class 1
```

so, even though R2 nicely describes the second part of the condition (attr_5 = 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 attr_4 instead of attr_1 or attr_2. If one changes the conflict-avoidance heuristic to subsample 20 conflicts before a decision is being made, the following four rules are extracted:

```
R1: if attr_1 is 1 and attr_2 is 1 then class 1
R2: if attr_1 is 3 and attr_2 is 3 then class 1
R3: if attr_1 is 2 and attr_2 is 2 then class 1
R4: if attr_5 is 1 then class 1
```

which is indeed the optimal representation of the underlying concept.

The same applies to the third monk's problem. Without conflict subsampling seven 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 attr_4 is 1 and attr_5 is 3 then class 1
R2: if attr_2 is (1 or 2)
  and attr_5 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. Tolerating outliers

Most existing algorithms to construct rule-based models from data have tremendous problems with noisy data or data containing outliers. Usually an excessive number of rules is being introduced simply to model noise and/or outliers. This is due to the fact that these algorithms aim to generate conflict free rules, that is, examples encountered during training will result in a degree of membership > 0 only for those rules of the correct class. Unfortunately in case of outliers such an approach will, especially in high-dimensional feature spaces, result in an enormous amount of rules to avoid these conflicts.

Many algorithms approach this problem by trying to build a simpler model from the beginning by ignoring irrelevant patterns in the original data. Decision Trees, for example, do not split nodes anymore or prune splits on lower levels afterwards. One disadvantage of this way to handle irrelevant data is the

loss of information. It is usually not straight forward to extract knowledge about which areas of the feature space are modeled insufficiently or which example patterns were considered irrelevant. In the following a methodology is discussed which generates two models, one describing the overall behavior of the underlying system and a second model which describes patterns that were considered irrelevant or uninformative.

4.1. Extracting irrelevant rules

Using an already existing set of rules, we can in many cases easily determine parts that have low relevance, based on their weight or another parameter which denotes individual relevance. To measure a rule's relevance often the weight parameter w is used which represents the number of training patterns covered by rule $\mathbf{R}$. From this a measure for the importance or relevance of each rule can be derived, by simply using the percentage of patterns covered by this rule:

$$\Phi(\mathbf{R}) = \frac{w}{|\mathbf{T}|}.$$

Other measures which are also used determine the loss of information if rule $\mathbf{R}$ is omitted from the entire set of rules $\mathscr{R}$:

$$\Phi(\mathbf{R}) = I(\mathscr{R}) - I(\mathscr{R} \setminus \{\mathbf{R}\}),$$

where $I(\cdot)$ indicates a function measuring the information content of a rule set. For our experiments we used (an extensive overview can be found in [30] and also [31])

- the Gini-index:

$$I_{\text{Gini}}(\mathscr{R}) = 1 - \sum_{c=1}^{C} V_c(\mathscr{R})^2,$$

- and the fuzzy entropy function:

$$I_{\text{Entropy}}(\mathscr{R}) = -\sum_{c=1}^{C} (V_c(\mathscr{R}) \log_2 V_c(\mathscr{R})),$$

where $V_c(\mathscr{R})$ indicates the volume of all rules $\mathbf{R} \in \mathscr{R}$ which are assigned to class c. In [32,33,35] it is shown how this volume can be computed efficiently based on a system of fuzzy rules.

The choice of relevance-measure is made depending on the nature of the underlying rule generation algorithm, as well as the focus of analysis, i.e., the interpretation of important vs. unimportant or useless data points. Using such a measure of (notably subjective) relevance, we can now extract rules with low

relevance from this model, assuming that they describe points in the data which are outliers or sparse points:

$$\mathscr{R}_{\text{outlier}} = \mathscr{R} \setminus \{\mathbf{R} \in \mathscr{R} \,|\, \Phi(\mathbf{R}) < \theta_{\text{outlier}}\}.$$

Using this "outlier"-model as filter for a second training phase will then generate a new rule-based model which has less rules with higher significance. In fact, the original training data $\mathbf{T}$ is filtered and only data points which are not covered by the outlier model will be used to construct the new model:

$$\mathbf{T}_{\text{clean}} = \left\{ \left(\vec{x}, \vec{\mu}^{\text{target}} \right) \in \mathbf{T} \,|\, \forall \mathbf{R} \in \mathscr{R}_{\text{outlier}} : \mu_{\mathbf{R}}(\vec{x}) \leqslant \theta_{\text{filter}} \right\}$$

Fig. 1 shows the flow of this procedure.

Note, how the initial model is being used to extract the outlier-model. This model is then in turn used as a filter for the existing training data to generate the final model. In the following, we will show how this affects the size of the rule sets on two real-world datasets.

4.2. Experimental results

Experiments on two datasets from the StatLog-archive [24] were performed to demonstrate the effect of the proposed methodology. The relevance function $\Phi(\mathbf{R}^{(j)}) = w^{(j)}$ with a threshold of $\theta_{\text{outlier}} = 5$ was used, that is rules which cover less than five patterns were considered irrelevant. The filtering threshold θ_{filter} was chosen to be 1, in effect removing patterns that lie within the cores of irrelevant rules only.

The first dataset contains images from Satellites (Satimage-dataset). Patterns with 36 attributes have to be separated into six different classes and 4435 training and 2000 test patterns were used. Table 2 (left) shows the results. Here "Stand" stands for the normal algorithm which generates rules in one run. H_1 indicates the general model generated through the algorithm explained above and H_2 denotes the outlier model. The number of rules for both models is shown in the last column. The number before the brackets indicates the size of

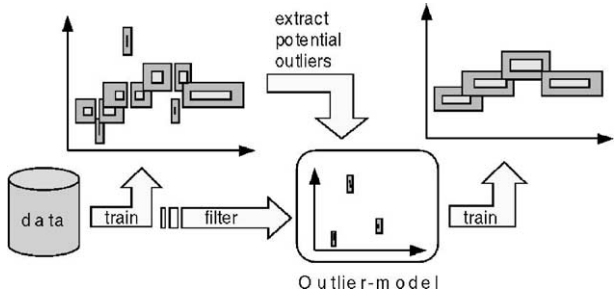

Fig. 1. The role of the two models during training.

Table 2
Results on the satimage and segment datasets

Level of distort. (%)	Used model	Satimage data		Segment data	
		Error (%)	No. of rules	Error (%)	No. of rules
0.0	Stand.	15.9	393	3.5	96
	$H_1(H_0)$	13.5	270 (60)	3.0	80 (12)
1.0	Stand.	17.1	394	5.2	108
	$H_1(H_0)$	13.5	313 (81)	4.3	86 (22)
2.0	Stand.	18.1	404	6.9	113
	$H_1(H_0)$	12.9	295 (109)	5.6	83 (30)
5.0	Stand.	18.1	479	6.1	144
	$H_1(H_0)$	12.4	334 (145)	3.8	107 (37)
10.0	Stand.	22.3	578	6.5	151
	$H_1(H_0)$	15.2	379 (199)	6.5	106 (45)

the rule-set of the general model H_1, whereas the number in brackets denotes the number of rules of the outlier model H_0. It is interesting to see how already without any additional distortion (0.0%) the two-stage model shows slightly better performance using a considerable smaller number of rules (270 vs. 393). Note also how the error rate on the unseen test data increases much slower with increases in distortion for the two-stage model. The gap between the sizes of the two models widens as well.

The second dataset is the Segment data from the same archive where 19 inputs and 7 classes are used with 2079 training and 220 test patterns. Table 2 (right) shows the results on this dataset. Here the effect in performance is not as obvious. Still noticeable, however, is the difference in model size. While the size of the separate outlier-model increases with increasing distortion, the size of the model representing the more general behavior grows much slower.

It is obvious that this methodology can be applied to other rule induction algorithms as well. As long as it is possible to evaluate and extract local parts of a model easily such a filtering procedure can be used. For Neural Networks and also Decision Trees such an approach is not as easily applicable, however. Pruning parts of a such structures can affect the decision function in potentially large areas of the feature space.

5. Conclusions

We have presented a new method for fuzzy rule formation. 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 this particular part of the feature space. The

classification performance of the new algorithm was demonstrated on benchmarks from the well-known StatLog project. We also demonstrated the interpretability of the extracted rules using the Iris and Monks data. Two extensions to the algorithm were outlined. First, 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. Second an approach to model (not only discard!) potential outliers was presented and evaluated on two benchmark data sets with various degrees of artificially created outliers.

We believe that rule induction algorithms as demonstrated here have tremendous potential in the areas of data mining and explorative data analysis. In addition to extraction of models that inhibit good generalization performance, rule models enable the user to actually understand the underlying behavior. This brings rule induction methods a large step towards explorative and real intelligent data analysis.

6. Future work

Extensions of this work focus mainly on two directions. Obviously building a two-stage hierarchy is only the beginning. In order to enable explorative data analysis an entire hierarchy of models at different levels of granulation will be beneficial. Rather than trying to find global and local trends in the same model such a hierarchy would enable the user to zoom in and out of the model, offering precisely the level of detail needed at the moment. In addition, work has been started in the areas of visualization of the extracted models. Rather than showing the user one long list of rules, a visual representation helps to intuitively understand the underlying relationships. In [34] a first step in this direction has been reported.

Acknowledgements

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 for his support and the opportunity for many stimulating discussions. Thanks also to the anonymous reviewers for their constructive feedback.

References

[1] Michael Berthold, David J. Hand (Eds.), Intelligent Data Analysis: An Introduction, Springer, Berlin, 1999.

[2] P. Clark, T. Niblett, The CN2 induction algorithm, in: Machine Learning, 3, 1989, pp. 261–283.
[3] R.S. Michalski, I. Mzetic, J. Hong, N. Lavrac, The multipurpose incremental learning system AQ15, in: Proceedings of National Conference on AI, AAAI, 5, 1986, pp. 1041–1045.
[4] S. Salzberg, A nearest hyperrectangle learning method, in: Machine Learning, 6, 1991, pp. 251–276.
[5] D. Wettschereck, A hybrid nearest-neighbour and nearest-hyperrectangle learning algorithm, in: Proceedings of the European Conference on Machine Learning, 1994, pp. 323–335.
[6] Shigeo Abe, Ming-Shong Lan, A method for fuzzy rules extraction directly from numerical data and its application to pattern classification, IEEE Transactions on Fuzzy Systems 3 (1) (1995) 18–28.
[7] Li-Xin Wang, Jerry M. Mendel, Generating fuzzy rules by learning from examples, IEEE Transactions on Systems, Man, Cybernetics 22 (6) (1992) 1313–1427.
[8] J. Ross Quinlan, Induction of decision trees, in: Machine Learning, 1, 1986, pp. 81–106.
[9] J. Ross Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, Los Altos, CA, 1993.
[10] Zheru Chi, Hong Yan, ID3-derived fuzzy rules and optimized defuzzification for handwritten numeral recognition, IEEE Transactions on Fuzzy Systems 4 (1) (1996) 24–31.
[11] Cezary Janikow, Fuzzy decision trees: Issues and methods, IEEE Transactions on Systems, Man, Cybernetics – Part B: Cybernetics 28 (1) (1998) 1–14.
[12] Patrick K. Simpson, Fuzzy min-max neural networks – part 1: classification, IEEE Transactions Neural Networks 3 (5) (1992) 776–786.
[13] Patrick K. Simpson, Fuzzy min–max neural networks – part 2: clustering, IEEE Transactions Fuzzy Syst. 1 (1) (1993) 32–45.
[14] Charles M. Higgins, Rodney M. Goodman, Learning fuzzy rule-based neural networks for control, in: Advances in Neural Information Processing Systems, CA, 5, Morgan Kaufmann, Los Altos, CA, 1993, pp. 350–357.
[15] Detlef Nauck, Fuzzy data analysis with NEFCLASS, in: Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference, Piscataway, July 2001, IEEE, New York, 2001, pp. 1413–1418.
[16] Rajesh Davé, Raghu Krishnapuram, Robust clustering methods: a unified view, IEEE Transactions on Fuzzy Systems 5 (2) (1997) 270–293.
[17] Frank Klawonn, Annette Keller, Fuzzy clustering and fuzzy rules, in: Proceedings of the 7th International Fuzzy Systems Association World Congress (IFSA'97), vol. 1, Academia, Prague, 1997, pp. 193–198.
[18] Amir B. Geva, Hierarchical unsupervised fuzzy clustering, IEEE Transactions on Fuzzy Systems 7 (6) (1999) 723–733.
[19] Shigeo Abe, Ruck Thawonmas, A fuzzy classifier with ellipsoidal regions, IEEE Transactions on Fuzzy Systems 5 (3) (1997) 358–368.
[20] Detlef Nauck, Using symbolic data in neuro-fuzzy classification, in: Proceedings of 18th NAFIPS International Conference, IEEE, New York, 1999, pp. 536–540.
[21] Tom Mitchell, Generalization as search, Artificial Intelligence 18 (2) (1982) 203–226.
[22] Tom Mitchell, Machine Learning, McGraw Hill, New York, 1997.
[23] Michael R. Berthold, Klaus-Peter Huber, Constructing fuzzy graphs from examples, Intelligent Data Analysis 3 (1) (1999) 37–54.
[24] D. Michie, D.J. Spiegelhalter, C.C. Taylor (Eds.), Machine Learning, Neural and Statistical Classification, Ellis Horwood Limited, Chichester, UK, 1994.
[25] Michael R. Berthold, Jay Diamond, Constructive training of probabilistic neural networks, Neurocomputing 19 (1998) 167–183.
[26] Dietmar Heinke, Fred H. Hamker, Comparing neural networks, IEEE Transactions on Neural Networks 9 (6) (1998) 1279–1291.

[27] R.A. Fisher, The use of multiple measurements in taxonomic problems, in: Annual Eugenics, II, 7, John Wiley, New York, 1950, pp. 179–188.
[28] C.L. Blake, C.J. Merz, UCI repository of machine learning databases, at ics.uci.edu in pub/machine-learning-databases, 1998.
[29] Sebastian B. Thrun, The MONK's problems – a performance comparison of different learning algorithms, Tech. Rep., Carnegie Mellon University, Pittsburgh, PA, December 1991.
[30] C. Apte, S.J. Hong, J.R.M. Hosking, J. Lepre, E.P.D. Pednault, B.K. Rosen, Decomposition of heterogeneous classification problems, Intelligent Data Analysis 2 (2) (1998).
[31] V. Cherkassky, F. Mulier, Learning from Data, John Wiley and Sons Inc., New York, 1998.
[32] Rosaria Silipo and Michael R. Berthold, Discriminative power of input features in [35] (1999).
[33] Rosaria Silipo, Michael R. Berthold, Input features impact on fuzzy decision processes, IEEE Transaction on Systems, Man, and Cybernetics, Part B: Cybernetics 30 (6) (2000) 821–834.
[34] Michael R. Berthold and Lawrence O. Hall, Visualizing fuzzy points in parallel coordinates, Tech. Rep. UCB/CSD-99-1082, University of California at Berkeley, Dec. 1999.
[35] David Hand, Joost Kok, Michael Berthold (Eds.), Advances in Intelligent Data Analysis, Lecture Notes in Computer Science, 1642, Springer, Berlin, 1999.