Control and Cybernetics

vol. **29** (2000) No. 4

Neural methods of knowledge extraction

by

**Włodzisław Duch, Rafał Adamczak,
Krzysztof Grąbczewski and Norbert Jankowski**

Department of Computer Methods, Nicholas Copernicus University
ul. Grudziądzka 5, 87-100 Toruń, Poland
E-mail: duch,raad,kgrabcze,norbert@phys.uni.torun.pl

Abstract: Contrary to the common opinion, neural networks may be used for knowledge extraction. Recently, a new methodology of logical rule extraction, optimization and application of rule-based systems has been described. C-MLP2LN algorithm, based on constrained multilayer perceptron network, is described here in details and the dynamics of a transition from neural to logical system illustrated. The algorithm handles real-valued features, determining appropriate linguistic variables or membership functions as a part of the rule extraction process. Initial rules are optimized by exploring the accuracy/simplicity tradeoff at the rule extraction stage and the one between reliability of rules and rejection rate at the optimization stage. Gaussian uncertainties of measurements are assumed during application of crisp logical rules, leading to "soft trapezoidal" membership functions and allowing to optimize the linguistic variables using gradient procedures. Comments are made on application of neural networks to knowledge discovery in the benchmark and real life problems.

Keywords: data mining, decision support, logical rules, fuzzy rules, optimization, medical diagnosis.

1. Introduction

In many applications the rule-based classifiers may be created automatically by extracting the rules from data using machine learning (Mitchel, 1997), fuzzy logic (Kosko, 1992) or neural network methods (Duch et al., 2000). Classical crisp logic rules are obtained from fuzzy rules if all membership functions are rectangular (i.e. their values are 0 or 1). Rectangles allow to define logical linguistic variables for each feature by intervals or sets of nominal values and thus allow to express logical rules in simple sentences like "IF the odor is fishy THEN the mushroom is poisonous". If rectangular functions are softened or changed

Fuzzy logic classifiers are frequently based on a few triangular membership functions for each input feature, a further simplification comparing to trapezoidal functions.

If the number of rules is relatively small and their accuracy is sufficiently high, then rule-based classifiers are the optimal choice. Crisp logical rules are desirable since they are most comprehensible, but they have several drawbacks. First, when using crisp rules only one class is identified as the correct one, thus providing a black-and-white picture where some gradation could be appropriate. Second, reliable crisp rules may reject some cases as unclassified. Third, using the number of errors given by the crisp rule classifier for the cost function makes optimization difficult, since only non-gradient optimization methods may be used. All these problems are overcome if continuous membership functions are used, leading to the fuzzy rather than crisp rules. Fuzzy rules have two disadvantages: they are not so comprehensible as the crisp rules, and they usually involve more parameters determining positions and shapes of the membership functions.

Systems based on fuzzy logic frequently use a fixed set of membership functions with predetermined shapes. Although it helps to avoid overparameterization it creates some problems. Defining linguistic variables in such context-independent way amounts in effect to a regular partitioning of the whole input space into convex regions. This approach suffers from the curse of dimensionality, since with k linguistic variables in d dimensions the number of possible input combinations is k^d. Fuzzy rules simply pick up those areas in the input space that contain vectors from a single class only, but without the possibility of adapting membership functions to individual clusters in a single rule they do not allow for optimal description of these clusters. Much better results may be obtained with context-dependent linguistic variables (Duch et al., 1999), different in each rule.

Machine learning methods are frequently tested in artificial, noiseless domains (see the three Monk problems, Thrun et al., 1991), while their utility for real problems with large amount of data, overlapping classes and the need for simplified, although less accurate, data description is not apparent. Neural networks are universal classifiers used in such problems, but they have an opinion of being opaque black boxes. Several neural methods have been compared experimentally on the mushroom and the three Monk problems benchmark datasets (Andrews et al., 1995), and recently a comparison with some machine learning methods has been given (Duch et al., 2000). There is no reason why a simple classification model based on logical rules should always work, but in some cases it does and is certainly worth using. In many applications simple crisp logical rules proved to be more accurate and were able to generalize better than many machine and neural learning algorithms (Duch et al., 1998, 1999). One should always try to use the simplest description of the data possible, but not simpler. In a few applications fuzzy rules proved to be more accurate (Duch et al.,

is too large, other, more sophisticated classification models are needed – thus, a hybrid, neuro-logical algorithm is described in this paper.

Although interpretation of crisp rules seems to be straightforward, in fact it may be quite misleading. A small change in the value of a single feature may lead to a complete change of the predicted class. Thus, interpretation of crisp rules is not stable against small perturbations of input values. Fuzzy rules are better in this respect since estimation of probabilities of different classes change smoothly. There is a tradeoff between fuzziness and the degree of precision. If the membership functions are too broad, all classes have similar probability. In the opposite case perturbation of the input vector may significantly change classification probabilities, even if the size of the perturbation is within the range of accuracy of the measured input values. Interpretation without exploration of alternative diagnoses may in such cases be rather dangerous. Rough rules suffer from the same interpretative problems even to a greater degree, because rough classifiers (Pal and Skowron, 1999) produce a large number of unstable rules (Breiman, 1998, on the importance of stability).

Although the biggest advantage of rule-based classifiers is their comprehensibility, interpretation of rules in practice is not so simple. On the other hand, neural networks may easily be converted into systems that are equivalent to crisp or fuzzy rule-based classifiers and thus may have transparent interpretation. In this paper only one classical neural model, the constructive multilayer perceptron (C-MLP), constrained to work as a logical-like network (hence the name of the method, C-MLP2LN, Duch et al., 1998), is described. However, it should be clear that using neural network models based on localized separable transfer functions (Duch and Jankowski, 1999), such as the triangular functions, or soft trapezoidal functions, allows for a smooth transition from crisp to fuzzy rules and enables natural interpretation of rules. Such neurofuzzy systems (see Duch and Diercksen, 1995; Duch et al., 1997) may also be used for quite complex data analysis (Duch et al., 1999).

In the next section a short overview of recent work on extraction of knowledge from data is presented. The third section describes the latest developments of the C-MLP2LN model and illustrates the transition process from complex data description to simple decision borders realized by sets of crisp logic rules. The fourth section deals with optimization and application of sets of rules and the fifth section illustrates the method on a couple of problems. The paper ends with a short discussion.

2. Neural methods of knowledge extraction

A good strategy in data mining is to extract the simplest crisp logical rules first. If the number of logical rules required for high accuracy of classification is large, then more sophisticated methods, such as fuzzy rules, capable of providing complex decision borders, should be used. Are neural methods competitive to

are two issues here: understanding what neural networks really do, and using neural networks to extract logical rules describing the data. There is a strong competition from decision trees (Quinlan, 1993, Michie et al., 1994), which are fast, accurate and can easily be converted to sets of logical rules, from inductive methods of machine learning (Mitchell, 1997), and from systems based on fuzzy (Kosko, 1992) and rough sets theories (Pal and Skowron, 1999).

Despite this competition, neural networks seem to have important advantages, especially for real-life problems with continuously-valued inputs. Good linguistic variables may be determined simultaneously with logical rules, selection and aggregation of features into smaller number of more useful features may be incorporated in the neural model, adaptation mechanisms for continuously changing data (on-line learning) are built in, wide-margin classification provided by neural networks leads to more robust logical rules. An overview of neural methods used for extraction of logical rules has recently been published (Duch et al., 2000), therefore only a summary of our recent work on this subject is given here.

Knowledge that is understandable to humans may come in different forms. The simplest form of knowledge is contained in the standard IF ... THEN prepositional rules used in many expert systems. Non-standard rules, such as the M-of-N rules (M out of N antecedents should be true) are quite natural for the most common MLP neural networks, where the basic operation performed by the neurons is to compare weighted combination of input values with the threshold θ. The output function

$$o(I_i(\mathbf{X})) = \sigma(I_i) = \sigma\Big(\sum_j W_{ij} X_j - \theta_i\Big), \tag{1}$$

has usually the sigmoidal shape (for example it may be a logistic function $\sigma(I) = 1/(1+e^{-\beta I})$, where β is a constant determining the slope), and becomes at the limit of infinite slope a step function. On the other hand, the Radial Basis Function (RBF) networks (Bishop, 1995) frequently use Gaussian functions as transfer functions. Triangular functions and symmetric trapezoidal functions are also radial and may be used in RBF networks. In general, the separable output functions

$$o(\mathbf{X}) = \prod_i \mu_i(X_i), \tag{2}$$

computing products of one-dimensional function have a straightforward interpretation as the membership functions of linguistic variables (Duch and Diercksen, 1995). In the MLP network "natural" membership functions are obtained as a difference of two sigmoidal functions, $\mu_i(X_i) = \sigma(X_i) - \sigma(X_i - \theta_i)$ or the product of sigmoidal functions $\sigma(X_i)(1 - \sigma(X_i))$ in all dimensions. It is not difficult to prove that after normalization the two forms are identical:

$$\sigma(X+b)(1-\sigma(X-b)) \qquad \sigma(X+b) - \sigma(X-b) \tag{3}$$

These membership functions are easily realized using a pair of constrained MLP neurons (Fig. 1), where the weights are either zero or ± 1 and the thresholds define the linguistic variables. In the limit of high gain (large β in logistic functions) they are converted into crisp linguistic variables: s_k is true if the input value $X_i \in [X_{i,k}, X'_{i,k}]$, i.e. linguistic variables for a given feature X_i are parameterized by interval values $s_k(X_{i,k}, X'_{i,k})$.

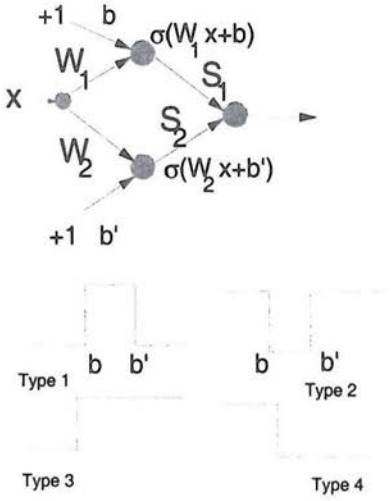

Figure 1. Two sigmoidal neurons are used to construct a linguistic unit converting continuous inputs to linguistic variables. Four basic types of "window" functions are obtained, depending on the **W** and **S** weight values.

Since crisp logical rules are the simplest and most comprehensible, they should be tried first. They provide hyperrectangular decision borders in the feature subspaces corresponding to variables appearing in rule conditions. This approximation may not be sufficient if complex decision borders are required, but it may work quite well if the problem has an inherent logical structure.

The classifier based on logical rules provides an approximation to the posterior probability $p(C_i|X;M)$, where the classification model M is composed of the set of rules. Crisp rules give $p(C_i|X;M) = 0,1$ but if clusters belonging to different classes overlap this is obviously wrong. Fuzzy rules, for example in the form

$$p(C_k|X;M) = \frac{\mu^{(k)}(X)}{\sum_i \mu^{(i)}(X)}, \tag{4}$$

where $\mu^{(k)}(X)$ is the value of the membership function defined for the cluster k,

dependent or **cluster-dependent membership functions** are rarely used in classification systems based on fuzzy logic, although they are quite natural in the neurofuzzy systems (Duch and Diercksen, 1995, Duch et al., 1997). Neurofuzzy systems adapt the number as well as the shapes of the membership functions to the data. Although various fuzzy, rough and neurofuzzy systems differ in their ability to discover and use logical rules for data description, their ultimate capability depends on the decision borders they may provide for classification. For example, if a simple rule $X_1+X_2 > 1$ classifies data correctly a large number of fuzzy or crisp rules may be created to obtain a poor description of the data, while systems that use rotated decision borders handle it perfectly with a single rule.

Extraction of linguistic variables and sets of logical rules proceeds in the following manner (Duch et al., 2000):

- Select linguistic variables. In case of continuous features X_i the linguistic variable s_k is true if the input value $X_i \in [X_{i,k}, X'_{i,k}]$, i.e. they are parametrized by interval values $s_k(X_{i,k}, X'_{i,k})$.
- Extract rules from the data using neural, machine learning or statistical techniques.
- Optimize linguistic variables (intervals they depend upon) using the rules and exploring the accuracy/rejection rate tradeoff.
- Repeat previous steps until a stable set of rules is found.
- Introduce and optimize input uncertainties.

The last step will be explained in Section 4. We have described several methods of initial rule extraction, based on decision trees (using the SSV separability criterion, Grąbczewski and Duch, 1999), Feature Space Mapping Network, search-based discretized networks and standard MLP networks trained with the backpropagation procedure (Duch et al., 2000). Since the constructive MLP network gave very simple and accurate sets of rules in a number of applications we have developed it further and present the algorithm in details below.

3. C-MLP2LN model

MLP2LN is a smooth transformation between the MLP network and a network performing logical operations (Logical Network, LN) (Duch et al., 1998). This transformation should simplify the network as much as possible to facilitate logical rule extraction. Skeletonization of a large MLP network is the method of choice if our goal is to find logical rules for an already trained network. Otherwise, the constructive approach, starting from a single neuron and expanding the logical network during training (called further C-MLP2LN method) is faster and more accurate. Smooth transition from an MLP to a logical-type of network performing similar functions is achieved during network training by:
a) simplifying the network structure by decreasing the weights during the train-

b) gradually increasing the slope β of sigmoidal functions $\sigma(\beta x)$ to obtain crisp decision regions;
c) enforcing the integer weight values 0 and ± 1, interpreted as 0 = irrelevant input, $+1$ = positive and -1 = negative evidence.

To achieve these objectives two additional terms are added to the standard mean square error function $E_0(W)$ to form the total cost function $E(\mathbf{W})$:

$$\begin{aligned} E(\mathbf{W}) &= E_0(\mathbf{W}) + R_1(\mathbf{W}) + R_2(\mathbf{W}) \\ &= \frac{1}{2}\sum_p \sum_k (\mathbf{Y}_k^{(p)} - \mathbf{F}_k(\mathbf{X}^{(p)}; \mathbf{W}))^2 \\ &+ \frac{\lambda_1}{2}\sum_{i,j} W_{ij}^2 + \frac{\lambda_2}{2}\sum_{i,j} W_{ij}^2 (W_{ij}-1)^2 (W_{ij}+1)^2 \end{aligned} \tag{5}$$

The first part is the standard mean square error measure of matching the network output vectors $\mathbf{F}(\mathbf{X}^{(p)}; \mathbf{W})$ with the desired output vectors $\mathbf{Y}^{(p)}$ for all training data samples $\mathbf{X}^{(p)}$. The first regularization term $R_1(\mathbf{W})$, scaled by λ_1, is frequently used in the weight pruning or in the Bayesian regularization method (Bishop, 1995) to improve generalization of the MLP networks. The second regularization term $R_2(\mathbf{W})$, scaled by λ_2, is a sum over all weights and has a minimum (zero) for weights approaching zero or ± 1.

A naive interpretation why regularization works (for a more sophisticated view see Bishop, 1995, and references there) is based on observation that small weights and thresholds mean that only the linear part of the sigmoid around $\sigma(0)$ is used. Therefore, the decision borders are rather smooth. On the other hand, for logical rules decision borders should be sharp and the network should be as simple (skeletal) as possible. Therefore the regularization term that we have used so far may not be the most appropriate. Another regularization term:

$$R_1(W) = \frac{\lambda_1}{2}\sum_{ij} \frac{W_{ij}^2}{1+W_{ij}^2}, \tag{6}$$

does not grow to infinity for large weights and thus allows those weights that should not vanish at the end of the training to stay sufficiently large. It induces an extra weight change that is easy to implement in the backpropagation training procedure:

$$W_{ij} \leftarrow \left(1 - \frac{\lambda_1 \eta}{(1+W_{ij}^2)^2}\right) W_{ij} \tag{7}$$

where η is the learning constant.

The first regularization term is used at the beginning of the training to force as many weights as possible – without a sharp increase of the mean square error term $E_0(\mathbf{W})$ – to become sufficiently small to be removed. This term is switched

stage of the training. This allows the network to increase the remaining weights. Large weights, together with increasing slopes of sigmoids, lead to sharp decision borders of rectangular shape. Although non-zero weights have values restricted to ± 1, increasing the slopes β is equivalent to using the network with one, large non-zero weight value $W = \pm\beta$ with sigmoidal functions of a unit slope.

An obvious generalization is to use several different maximal W values in the final network, for example by adding, after skeletonization of the network, the following penalty term:

$$\sum_{i,j}(\sigma(W_{ij}+1) - \sigma(W_{ij}-1)). \tag{8}$$

This term will not restrict the weights to ± 1 but will allow them to grow beyond these values. If the network is used to extract the logical rules at the end of the training the slopes should be infinitely steep, corresponding to infinite non-zero weights (in practice $W = \pm 10000$ is used), so that nothing is gained. However, if the final goal is a hybrid, network-rule based neuro-logical system that provides logical description of data whenever possible, and more complex decision borders wherever necessary, this may be an attractive solution.

The architecture of the network is presented in Fig. 2. Logical (binary) inputs may be directly connected to the rule nodes (R-nodes), while all continuous inputs go through L-units creating linguistic variables. In some applications

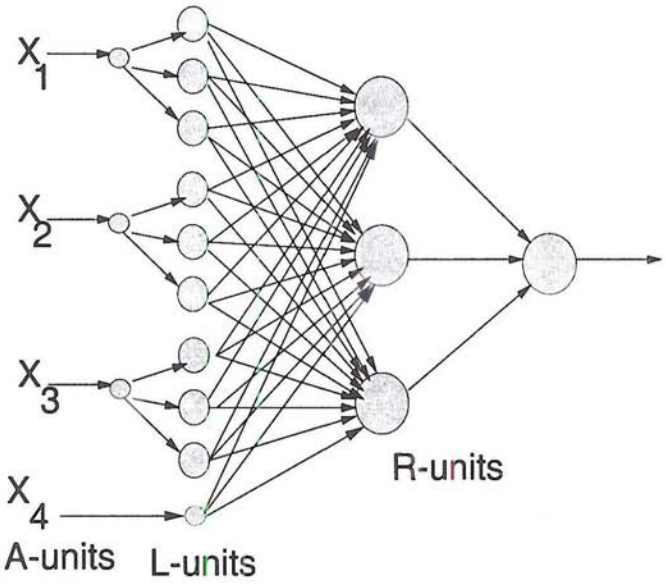

Figure 2. MLP network with linguistic and rule units. An additional aggregation layer provides the X_i inputs to the L-units; X_1–X_3 are real-valued, X_4 is a logical

with a large number of features **an aggregation** of features belonging to the same type is possible, leading to a smaller number of linguistic variables that carry more information. Groups of features that are of the same type are combined together by an additional layer of neurons between the input and the L-units. These aggregation units (A-units) should incorporate domain knowledge about the type of input features and usually may be linear. In some cases one may use information theory or separability criteria (Wettschereck, 1997) to set up the weights for these units, decreasing the number of adaptive parameters during the network training.

Initial knowledge about the problem may be inserted directly into the network structure, defining initial network parameters and structure that is modified during on-line training in view of the incoming data. Since the final network structure becomes quite simple, insertion of partially correct rules to be refined by the learning process is quite straightforward.

The training proceeds separately for each output class. Although the method works with general multilayer backpropagation networks we recommend the C-MLP2LN constructive procedure that frequently leads to a satisfactory solution in a much faster way. This is due to the fact that no experimentation is needed to determine the network architecture and that a single neuron is trained at a time instead of all neurons simultaneously. While the actual differences in timing strongly depend on the problem, the constructive MLP2LN method has frequently been more than two orders of magnitude faster than the standard network.

As with all neural procedures, for some data the network training may slow down and require some experimentation so the procedure is not completely automatic. Typical parameter values that work in most cases are given in the description of the training procedure here.

1) Set up the structure of the aggregation layer and create L-units for continuous inputs, usually 1–3 units per input (too small number of the linguistic variables will lead to low accuracy of rules).
2) Create one hidden neuron (R-unit neuron) per class.
3) Train the neuron on data for the first class using backpropagation procedure with regularization. Start with small $\lambda_1 = 10^{-5}$ and $\lambda_2 = 0$ and the unit slope $\sigma(\beta x)$, $\beta = 1$.
4) If convergence is too slow add another R-unit neuron and train two neurons simultaneously; in rare cases training even more neurons may significantly speed up the training.
   (a) Train as long as the error decreases; then increase $\lambda_1 \leftarrow 10\lambda_1$ and the slope of sigmoidal functions $\beta \leftarrow \beta + 1$ and train further; repeat this step until a sharp increase of the error is noticed when λ_1 is increased.
   (b) Decrease λ_1 slightly until the error is reduced to the previous value

(c) Take $\lambda_2 = \lambda_1$ and put $\lambda_1 = 0$; train slowly increasing the slopes and λ_2 until the remaining weights reach 0 ± 0.05 or $\pm 1 \pm 0.05$.

(d) Set very large slopes $\beta \approx 1000$ and integer weights $0, \pm 1$.

5) Analyze the weights and the threshold(s) obtained by checking the combinations of linguistic features that activate the first neuron(s). This analysis allows to write the first group of logical rules that cover the most common input-output relations.
6) Freeze the weights of existing neurons during further training. This is equivalent to training only new neurons (usually one per class at a time).
7) Add the next neuron and train it on the remaining data in the same way as the first one. Connect it to the output neuron for the class it belongs to (if more than one R-neuron for this class has been created).
8) Repeat this procedure until all data are correctly classified, or the number of rules obtained grows sharply, signifying overfitting (for example one or more rules per one new vector classified correctly are obtained).
9) Repeat the whole procedure for data belonging to other classes.

The network expands after a neuron is added and then shrinks after connections with small weights are removed. A set of rules $\mathcal{R}_1 \vee \mathcal{R}_2 \vee \ldots \vee \mathcal{R}_n$ is found for each class separately. The output neuron for a given class is connected to the hidden neurons created for that class – in simple cases only one neuron may be sufficient to learn all instances, becoming an output neuron rather than a hidden neuron (Fig. 3). Output neurons performing summation of the incoming signals are linear and have either positive weight $+1$ (adding more rules) or negative weight -1. The last case corresponds to those rules that cancel some of the errors created by the previously found rules that were too general. They may be regarded as exceptions to the rules.

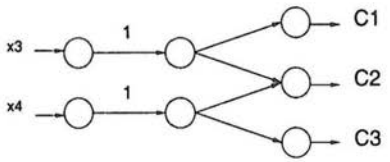

Figure 3. Structure of the simplest network solving the Iris problem.

Since each time only one neuron per class is trained the C-MLP2LN training is fast. Both standard MLP architecture with linguistic inputs or the L-R network may be used with the C-MLP2LN training algorithm. The first neuron for a given class learns the most general pattern, covering the largest number of instances. Therefore rules obtained by this algorithm are ordered, starting with

cases. An optimal balance between the number of rules and the generalization error is usually obtained when only the rules that cover larger number of cases are retained. The final solution may be presented as a set of rules, or as a network of nodes performing logical functions, with hidden neurons realizing the rules, and the hidden-output neuron weights set to ± 1. However, some rules obtained from analysis of the network may involve spurious conditions and therefore the optimization and simplification steps are necessary.

The λ_1 and λ_2 parameters determine the simplicity/accuracy tradeoff of the generated network and extracted rules. If a very simple network giving only a rough description of the data (and thus simple logical rules) is desired, λ_1 should be as large as possible. Although one may estimate the relative size of the regularization term versus the mean square error (MSE) a few experiments are sufficient to find the largest value for which the MSE is still acceptable and does not decrease quickly when λ_1 is decreased. Smaller values of λ_1 should be used to obtain more accurate networks (larger sets of rules). The final value of λ_2 near the end of the training is always set to larger values than the maximum value of λ_1 at the beginning of the training.

The dynamics of the learning process is illustrated using the well-known example of the Iris data. For each of the 3 different classes of the Iris flowers 50 samples are given, described by 4 numbers, the length and the width of flower's sepals and petals. The final structure of the simplest network that solves the problem is shown in Fig. 3. Only two of the four inputs have non-zero weights and only the second class needs the full L-unit, the weights in other L-units became sufficiently small to delete corresponding connections. There is no additional output layer since a single neuron classifies all data from the first class correctly (this class represents Iris-setosa variety) and the two other neurons make only 3 errors on the remaining two classes (Iris virginica and versicolor).

In Fig. 4 contours of decision borders are shown at various training stages. 5 output values around 0.5 are shown. In the beginning of the training contours are broadly spaced and at the end they collapse to a single line. At the beginning of the training (first subfigure, after 20 learning epochs with $\lambda_1 = 10^{-5}$ and $\eta = 0.1$) the network has slopes $\beta = 1$ and the absolute value of the largest weight is around 4; sigmoidal functions are smooth and the position of the 0.5 contour is influenced (through the MSE minimization) by all vectors in the training set. The next two subfigures show contours after 100 and 400 epochs, with the same learning and regularization parameters. The largest weight grew to about 13. The next subfigure shows contours after another 200 epochs of training with $\lambda_1 = 10^{-3}$ and $\beta = 3$, and the fifth subfigure after another 200 epochs with $\lambda_1 = 0$ and $\lambda_2 = 10^{-2}$. Finally, the last figure shows the logical network with $\beta = 10000$ and ± 1 weights. Please note that the decision border between the first class (left corner, Iris-Setosa) and the other two classes is at the optimal position, $x_3 < 2.55$. Some machine learning algorithms

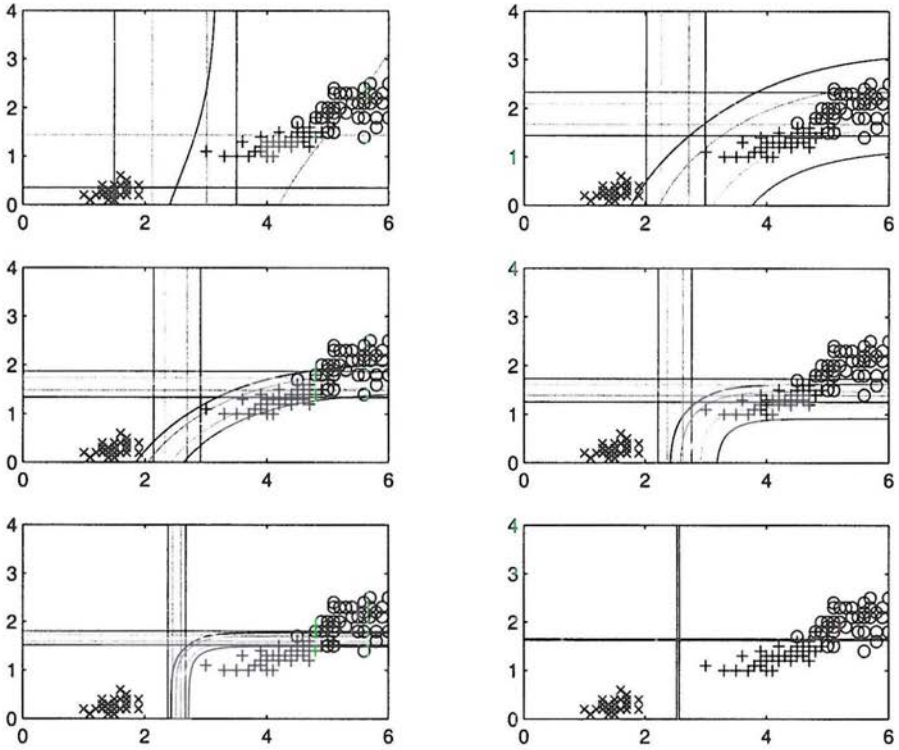

Figure 4. Contours of decision borders during training on the Iris data. The data is displayed in x_3 (petal length in centimeters) and x_4 (petal width) coordinates. Top left figure shows constant values of the network outputs at the beginning of the training when small slopes of the sigmoidal functions are used. During the training slopes gradually increase and contours collapse to a single line. The final subfigure at the bottom right corresponds to the infinite slopes of sigmoidal functions.

vectors in it. Such systems obviously produce rules giving poor generalization in crossvalidation tests.

The final rules obtained from this network are:

IF $(x_3 < 2.55)$ THEN Class1
IF $(x_4 > 1.66)$ THEN Class3
IF $(x_3 > 2.55 \wedge x_4 < 1.66)$ THEN Class2

MLP is changed into a logical network by increasing the slope β of sigmoidal functions to infinity, changing them into the step-functions. Such training process should be done carefully since very steep sigmoidal functions have non-zero

vectors contributing to the learning process goes to zero. Therefore, when convergence becomes slow for large slopes, it is necessary to stop network training, extract logical rules and optimize the intervals of the linguistic variables. This optimization step, described below, is performed at the level of the rule-based classifier, not the MLP network, and is independent of the methods used to generate the rules.

4. Optimization and application of logical rules

Optimization of linguistic variables, that the rules are based on, is done by minimization of the number of wrong predictions $\min_M[\sum_{i\neq j}\mathcal{P}(C_i,C_j)]$ (where $\mathcal{P}(C_i,C_j)$ is the confusion matrix for a rule-based classifier M), simultaneously with maximization of the predictive power of the classifier $\max_M[\mathrm{Tr}\,\mathcal{P}(C_i,C_j)]$ over all intervals X_k, X'_k contained in model M. This is equivalent to minimization without constraints of the following cost function $E(M)$:

$$E(M) = \gamma \sum_{i\neq j} \mathcal{P}(C_i,C_j) - \mathrm{Tr}\,\mathcal{P}(C_i,C_j) \geq -n, \tag{9}$$

where parameter γ decides whether high overall accuracy with low rejection rate or high reliability with larger rejection rate is desired. Minimization of this formula is difficult if $\mathcal{P}(C_i,C_j)$ depends in a discontinuous way on the parameters in M, requiring non-gradient minimization methods. This is unfortunately the case if a crisp logic rule-based classifier is used.

Real input values are obtained by measurements that are carried with finite precision, therefore it is natural to assume that instead of a crisp number x a Gaussian distribution $G_x = G(y;x,s_x)$ centered around x with dispersion s_x should be used. Performing a Monte Carlo sampling from Gaussian distributions for all input features and using the rule-based classifier M to assign a class to all vectors X' from the distribution $G_X = G(Y,X,S_X)$ allows to compute probabilities $p(C_i|X)$. Dispersions $S_X = (s_{x1}, s_{x2} \ldots s_{xN})$ define the volume of the input space around X that has an influence on computed probabilities. Assuming that uncertainties $s_i = s_{xi}$ are independent of feature values is a useful approximation justified if the data is properly standardized.

Since the erf function obtained from integration of Gaussian distributions is quite similar to the logistic function with a very good approximation a rule $R_{[a,b]}(x)$ which is true (R=1) if $x \in [a,b]$ and false otherwise (R=0) is fulfilled by a Gaussian number G_x with probability:

$$p(R_{[a,b]}(G_x) = T) \approx \sigma(\beta(x-a)) - \sigma(\beta(x-b)), \tag{10}$$

where $\beta = 2.4/\sqrt{2}s_x$ defines the slope of the logistic function $\sigma(x) = 1/(1+\exp(-\beta x))$. For large dispersion s_x this probability is significantly different from zero well outside the interval $[a,b]$. Thus, crisp logical rules for inputs with

membership functions defined by the difference of the two sigmoids, used with crisp input value. The slopes of these membership functions, determined by the parameter β, are inversely proportional to the uncertainty of the inputs. In our neural network approach to rule extraction such membership functions are computed by the network's "linguistic units".

The probability that x belongs to a rule $R = r_1 \wedge \ldots \wedge r_N$ (each r_i is the rule condition, a subset or an interval) may be defined as a product of probabilities of $x \in r_i$ for $i = 1, \ldots, N$. Such definition assumes that all the attributes which occur in rule R are mutually independent, which is usually not the case. However, if the rule generator produces as simple rules as possible there should be no pairs of strongly dependent attributes in a single rule. Therefore, the product should be very close to real probability. Obviously the rule may not contain more than one premise per one attribute, but it is easy to convert the rules appropriately if they do not satisfy this condition.

Another problem occurs when probability of x belonging to a class described by more than one rule is estimated. Rules usually overlap because they use only a subset of all attributes and their conditions do not exclude each other. Summing and normalizing probabilities obtained for different classes may give results quite different from real Monte Carlo probabilities. To avoid this problem probabilities are calculated as:

$$P(x \in C) = \sum_{R \in 2^{\mathcal{R}_C}} (-1)^{|R|+1} P\Big(x \in \bigcap R\Big), \tag{11}$$

where $\mathcal{R}_C$ is a set of classification rules for class C, $2^{\mathcal{R}_C}$ is a set of all subsets of $\mathcal{R}_C$, $|R|$ is the number of elements in R and $\bigcap R$ is the subspace (for discrete x a set) created from conjunction of all rules R. If there are k rules for class C and they do not overlap this equation reduces to a sum $P(x \in R_1 \wedge R_2 \wedge \ldots \wedge R_k)$, otherwise regions where pairs are overlapping should be subtracted. Since this subtraction removes regions where 3 rules are overlapping twice they have to be added etc., hence the need for the $(-1)^{|R|+1}$ factor.

An assumption that the uncertainty of inputs s_i is identical in all points of the input space may not be justified. A more general approach to compute classification probabilities is based on a direct calculation of optimal soft-trapezoidal membership functions. Linguistic units of the LR-network provide such window-type membership functions, $L(x; a, b) = \sigma(\beta(x - a)) - \sigma(\beta(x - b))$. Relating the slope β to the input uncertainty allows to calculate probabilities that are the same as from the Monte Carlo sampling. A general rule node computes normalized product-type bicentral function:

$$R_j(\mathbf{X}; \mathbf{t}_j, \mathbf{b}_j, \mathbf{s}_j^L, \mathbf{s}_j^R) = \frac{\prod_{i \in \mathcal{I}(R_j)} \sigma((X_i - t_{ij} + b_{ij}) s_{ij}^L)(1 - \sigma((X_i - t_{ij} - b_{ij}) s_{ij}^R))}{} \tag{12}$$

where $\mathcal{I}(R_j)$ is a set of indices of features used in a given rule R_j. The output $O_j(\mathbf{X})$ of a neuron j that combines rules for separate clasess C_j is:

$$O_j(\mathbf{X}) = \sigma\Big(\sum_{i\in\mathcal{I}(C_j)} R_i(\mathbf{X};\mathbf{p}_i) - 0.5\Big), \tag{13}$$

where $\mathcal{I}(C_j)$ is a set of rules indices for a given class C_j and $\mathbf{p}$ is a set of all parameters in (12). Probability of the class C_j for given vector $\mathbf{X}$ is given by:

$$p(C_j|\mathbf{X};M) = \frac{O_j(\mathbf{X})}{\sum_i O_i(\mathbf{X})}, \tag{14}$$

and probability of class C_j for a given vector $\mathbf{X}$ and rule R_i is

$$p(C_j|\mathbf{X},R_i;M) = p(C_j|\mathbf{X})R_i(\mathbf{X};\mathbf{p}_i). \tag{15}$$

Optimization of centers $\mathbf{t}$, biases $\mathbf{b}$ and slopes $\mathbf{s}$ is done by the Kalman filter approach (Jankowski, 1999) or the batch version of gradient descent learning algorithm. Since probabilities $p(C_i|\mathbf{X};M)$ depend now in a continuous way on the linguistic variable parameters of the rule system M the error function comparing the true class $C(\mathbf{X})$ with the class C_i predicted with probability $p(C_i|\mathbf{X};M)$ is:

$$E(M,s_x) = \frac{1}{2}\sum_{\mathbf{X}}\sum_i (p(C_i|\mathbf{X};M) - \delta(C(\mathbf{X}),C_i))^2. \tag{16}$$

This function depends on the Gaussian uncertainties of inputs s_x or parameters of bicentral functions used to calculate probabilities. Confusion matrix computed using probabilities instead of the number of errors allows for optimization of (9) using gradient-based methods. This minimization may be performed directly or may be presented as a neural network problem with a special network architecture. Assuming that the uncertainty of s_x is a percentage of the range of X values optimization is reduced to a one dimensional minimization of the error function. Uncertainties s_x of the values of features may also be treated as additional adaptive parameters for optimization on the training data.

This approach leads to the following important improvements of any rule-based system:

- Crisp logical rules are used for maximum comprehensibility.
- Uncertainties of inputs are taken into account.
- Instead of 0/1 decisions the probabilities of classes $p(C_i|\mathbf{X};M)$ are obtained.
- Uncertainties of inputs s_x provide additional adaptive parameters.
- The neighborhood of $\mathbf{X}$ is explored and alternative classes discovered with increasing s_x.
- Inexpensive gradient methods are used allowing for optimization of very

- Rules with wider classification margins are obtained, overcoming the brittleness problem.

Wide classification margins are desirable to improve generalization of the classifier by optimizing the placement of the decision borders. If the vector $\mathbf{X}$ of unknown class is quite typical to one of the classes C_k increasing uncertainties of inputs s_x to a reasonable value (several times the real uncertainty, estimated for a given data) should not decrease the $p(C_k|\mathbf{X}; M)$ probability significantly. If this is not the case, $\mathbf{X}$ may be close to the class border and a detailed analysis of the influence of each feature on the classification probability should be performed.

An alternative way to go beyond the logical rules introduced in Jankowski (1999), Duch et al. (2000a) is based on *confidence intervals* and *probabilistic confidence intervals*. Confidence intervals are calculated individually for a given input vector while logical rules are extracted for the whole *training set*.

5. Summary of empirical results

Using the early version of theoretical ideas described above we have analyzed a large number of benchmark datasets (detailed comparison with other systems is given in Duch et al., 2000). These methods were also used in a real-life project, analyzing the psychometric data (Duch et al., 1999). Many results, including explicit logical rules, are collected in the Web page:
http://www.phys.uni.torun.pl/kmk/projects/rules.html

Rules are most useful when they are simple, comprehensible and accurate. Many sets of rules of various complexity have been generated using the C-MLP2LN approach. They may be used as a reference or benchmark for other rule extraction systems. Quite frequently only the reclassification accuracy (in-sample or overall accuracy) on the whole dataset for extracted rules is quoted. This may not be sufficient to estimate statistical accuracy of rules. When performing crossvalidation different rules are extracted for different partitions of the dataset and it becomes impossible to present a single set of rules or to compare rules obtained by different methods. The best comparison of accuracy is offered on large datasets with separate test parts, such as the hypothyroid or the NASA shuttle problem (both stored in the UCI repository, Murphy and Aha, 1994). The simplest rules are usually quite stable in crossvalidation tests and for such rules reclassification accuracy is close to statistical estimations.

C-MLP2LN was tried on the symbolic benchmark problems, the three Monk problems (Thrun et al., 1991) and the Mushroom problem (UCI repository). All the three Monk problems have been solved with 100% accuracy (Duch et al., 1997a). Four simple rules involving 6 features were found classifying all poisonous and edible mushrooms without errors. Since for this dataset there are 8124 vectors, with 22 symbolic features corresponding to 118 logical input variables, the task is nontrivial and shows the potential of the method in applications

Several small and noisy medical datasets were analyzed. Such datasets are difficult for many methods since they require good regularization or a very simple classifier to avoid overfitting of the data. Without regularization some methods may produce results that on the test set or in crossvalidation tests are below the base rate (frequency of the majority class). Although a good statistical approach to computational learning theory exists (Bishop, 1995) it is difficult in practice to find classifiers with complexity that would be optimized for a given dataset. When extracting logical rules with C-MLP2LN algorithm one immediately sees that the most general rules discovered at the beginning cover many cases while rules created with lower regularization parameters λ_1, λ_2 cover a few cases only and thus give too complex description of the dataset.

Consider the appendicitis dataset (Weiss and Kulikowski, 1991). It contains only 106 cases, with 8 attributes (results of medical tests), and 2 classes: 88 cases with acute appendicitis and 18 cases with other problems. Two simple rules:

$$\text{MNEA} > 6650 \vee \text{MBAP} > 12, \tag{17}$$

giving an overall accuracy of 91.5% result from a single neuron. Classification accuracy is improved by adding two more logical rules resulting from a second neuron created by the C-MLP2LN algorithm, but the first of these rules covers just two cases and the second just one case. Such rules are more likely due to the noise in the data then to a highly specific and rare cases of interest to an expert. What may be more interesting is to find rules of similar accuracy using other input features. Since initialization of the MLP network is random it has a chance to find several different solutions, for example

$$\text{WBC1} > 8400 \vee \text{MBAP} \geq 42, \tag{18}$$

has a slightly lower overall accuracy of 89.6%.

Another small dataset, the Ljubljana cancer data (from UCI repository, Murphy and Aha, 1994) contains 286 cases, 201 no-recurrence-events (70.3%) and 85 are recurrence-events. There are 9 input features, with 2 to 13 different values each. A single logical rule for the recurrence-events:

$$\text{involved nodes} > 2 \wedge \text{degree-malignant} > 2 \tag{19}$$

with ELSE condition for the second class, gives over 77% accuracy in crossvalidation tests. Although more accurate optimized rules have been found (Duch et al., 2000) crossvalidation tests showed no improvement. It is doubtful that there is more knowledge that may be extracted from this data than contained in the simple statement based on the rule given above: recurrence is expected if the number of involved nodes is bigger than 2 and the cells are highly malignant.

The quality of solutions that may be achieved using the C-MLP2LN algorithm is perhaps exemplified in the best way on a hypothyroid dataset. It contains 3772 cases for training, 3428 cases for testing, 22 attributes (15 binary,

and normal (no hypothyroid). The class distribution is very unbalanced: in the training set it is 93, 191, 3488 vectors and in the test set 73, 177, 3178. Our final optimized rules for the first two classes are (reliability of each rule is in parentheses):

$$\mathcal{R}_1(C_1)\text{: TSH} \geq 30.48 \wedge \text{FTI} < 64.27 \quad (97.06\%)$$

$$\mathcal{R}_2(C_1)\text{: TSH} \in [6.02, 29.53] \wedge \text{FTI} < 64.27 \wedge \text{T3} < 23.22 \quad (100\%)$$

$$\mathcal{R}_1(C_2)\text{: TSH} \geq 6.02 \wedge \text{FTI} \in [64.27, 186.71] \wedge \text{TT4} \in [50, 150.5) \wedge \text{on thyroxine} = \text{no} \wedge \text{surgery} = \text{no} \quad (98.96\%)$$

The ELSE condition has 100% reliability on the training set. These rules make only 4 errors on the training set (99.89%) and 22 errors on the test set (99.36%). They are more accurate than any other classification method that we have tried on this data, except for C4.5 decision tree (Quinlan, 1993) which gave slightly better test result.

The C-MLP2LN method may also fail in some cases, although it probably means that the data is not suitable for logical description. For example, we have analyzed the hepatobiliary disorders dataset (Hayashi et al., 1990), which contains medical records of 536 patients admitted to a university affiliated Tokyo-based hospital, with four types of hepatobiliary disorders: alcoholic liver damage, primary hepatoma, liver cirrhosis and cholelithiasis. The records included sex of the patient and the results of 9 biochemical tests. As in the original study 163 cases were used as the test data. A fuzzy neural network was trained until 100% correct answers were obtained on the training set. The accuracy on the test set varied from less than 60% to a peak of 75.5% but since there was no correlation between the results on the training and on the test set the method is unable to find the best solution. This data has also been analyzed by Mitra et al. (1997) using a knowledge-based fuzzy MLP system. Accuracy of results on the test set was between 33% and 66.3%, depending on the actual fuzzy model used. For this dataset 49 crisp logical rules were initially obtained by C-MLP2LN procedure, giving 83.5% accuracy on the training and 63.2% on the test set. Optimization did not improve these results significantly. Fuzzy rules derived using the FSM network, with Gaussian as well as with triangular functions, gave similar accuracy of 75.6–75.8%. The best results for this dataset, 83.4% on the training and 82.8% on the test set, were obtained with the weighted nearest neighbor ($k = 1$) method. Clearly, in this case the decision borders are too complex for logical rules.

6. Discussion

Machine Learning community has focused on artificial problems where a few symbolic attributes are defined (for example, the three Monk problems). It is quite hard to find results of machine learning methods for the datasets stored

1994). In data mining problems many continuously-valued features may be present and large sets of rules may be needed. Rule-based classifiers are useful only if rules are reliable, accurate, stable and sufficiently simple to be understood. Most classifiers are unstable (Breiman, 1998) and lead to rules that are significantly different if the training set is slightly changed. Such rules contain little useful information and in fact may be rather misleading. Even if stable and robust rules are found the user should be warned about potential misclassifications, other classification options, and sensitivity of the classification probability to small variations of each feature. Neural methods are capable of providing simple and accurate sets of rules. They are wide-margin classifiers, placing their decision borders as far from the data as possible and thus providing good linguistic variables with optimal discretization of continuous features. They may also produce many sets of rules of various complexity (thanks to different regularization levels) as well as different but equivalent sets of rules (thanks to random initialization).

In this paper the C-MLP2LN constructive constrained multilayer perceptron has been described in detail. An example was given illustrating the dynamics of decision borders converging to a solution equivalent to logical rules. These initial rules are then optimized by exploring the reliability/rejection rate tradeoff. In the final step an assumption about the uncertainties in the inputs is made, allowing to use crisp logical rules to compute classification probabilities. Crisp rules are then equivalent to fuzzy rules with soft trapezoidal membership functions. In practical applications users are interested in relevant features and may rarely be satisfied with answers to questions "why" based on quotation of complex sets of logical rules. Similarity to prototypes, or case-based interpretation, is an alternative to rule-based systems. Therefore one should not exaggerate the importance of logical description as the only understandable alternative to other classification methods.

Neural methods are so far restricted to relatively simple form of prepositional rules based on linguistic variables. This is sufficient for classification problems, where each case is described in the same feature space. In some applications more complex descriptions are required, with stepwise concept building. Chemical problems may be a good example here. Unfortunately it is difficult to find benchmark data for such cases.

Acknowledgments: Support by the Polish Committee for Scientific Research is gratefully acknowledged.

References

ANDREWS, R., DIEDERICH, J. and TICKLE, A.B. (1995) A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks. *Knowledge-Based Systems*, 8, 373–389.

BISHOP, C. (1995) *Neural networks for pattern recognition.* Clarendon Press,

BREIMAN, L. (1998) Bias-Variance, regularization, instability and stabilization. In: *Neural Networks and Machine Learning*, Bishop, C., ed., Springer.

BUTCHER, J.N. and ROUSE, S.V. (1996) Personality: individual differences and clinical assessment. *Annual Review of Psychology*, **47**, 87.

DUCH, W. and DIERCKSEN, G.H.F. (1995) Feature Space Mapping as a universal adaptive system. *Computer Physics Communication*, **87**, 341–371.

DUCH, W., ADAMCZAK, R. and GRĄBCZEWSKI, K. (1997a) Extraction of crisp logical rules using constrained backpropagation networks. *Proc. of International Joint Conference on Neural Networks (IJCNN'97)*, Houston, Texas, pp. 2384–2389.

DUCH, W., ADAMCZAK, R. and GRĄBCZEWSKI, K. (1998) Extraction of logical rules from backpropagation networks. *Neural Processing Letters*, **7**, 1–9.

DUCH, W., ADAMCZAK, R. and GRĄBCZEWSKI, K. (2000) Methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks* (in print).

DUCH, W., ADAMCZAK, R. and JANKOWSKI, N. (1997) New developments in the Feature Space Mapping model. *3rd Conf. on Neural Networks, Kule, Poland, Oct. 1997*, pp. 65–70.

DUCH, W., GRĄBCZEWSKI, K., JANKOWSKI, N. and ADAMCZAK, R. (2000a) Optimization and interpretation of rule-based classifiers. *Intelligent Information Systems IX, Bystra, Poland, June 2000* (submitted).

DUCH, W. and JANKOWSKI, N. (1999) New neural transfer functions. *Neural Computing Surveys*, **2**, 639–658.

DUCH, W., KUCHARSKI, T., GOMUŁA, J. and ADAMCZAK, R. (1999) *Metody uczenia maszynowego w analizie danych psychometrycznych. Zastosowanie do wielowymiarowego kwestionariusza osobowości MMPI-WISKAD.* Toruń, 650 pp.

GRĄBCZEWSKI, K. and DUCH, W. (1999) A general purpose separability criterion for classification systems. *4th Conf. on Neural Networks and Their Applications, Zakopane*, pp. 203–208.

HAYASHI, Y., IMURA, A. and YOSHIDA, K. (1990) Fuzzy neural expert system and its application to medical diagnosis. In: *8th International Congress on Cybernetics and Systems, New York City*, pp. 54–61.

JANKOWSKI, N. (1999) *Ontogenic neural networks and their applications to classification of medical data.* PhD thesis, Department of Computer Methods, Nicholas Copernicus University, Toruń, Poland.

KOSKO, B. (1992) *Neural Networks and Fuzzy Systems.* Prentice Hall.

MICHIE, D., SPIEGELHALTER, D.J. and TAYLOR, C.C. (1994) *Machine learning, neural and statistical classification.* Elis Horwood, London.

MITCHELL, T.M. (1997) *Machine Learning.* McGraw-Hill.

MITRA, S., DE, R. and PAL, S. (1997) Knowledge based fuzzy MLP for classification and rule generation. *IEEE Transactions on Neural Networks*, **8**,

MURPHY, P.M. and AHA, D.W. (1994) *UCI repository of machine learning databases.* Univ. of California at Irvine, Dept. of Information and Computer Science. http://www.ics.uci.edu/~mlearn/MLRepository.html

PAL, S.K. and SKOWRON, A. (1999) *Rough Fuzzy Hybridization. A New Trend in Decision-Making.* Springer-Verlag.

QUINLAN J.R. (1993) *C4.5: Programs for machine learning.* Morgan Kaufman, San Mateo.

THRUN, S.B. et al. (1991) *The MONK's problems: a performance comparison of different learning algorithms.* Carnegie Mellon University, CMU-CS-91-197.

WEISS, S.M. and KULIKOWSKI, C.A., eds. (1991) *Computer systems that learn.* Morgan Kauffman, San Mateo, CA.

WETTSCHERECK, D., AHA, D.W. and MOHRI, T. (1997) A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. *Artificial Intelligence Review*, **11**, 273–314.