Chapter 23

Rough Sets and Artificial Neural Networks

Marcin S. Szczuka

Institute of Mathematics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
e-mail: szczuka@mimuw.edu.pl

1 Introduction

This work is an attempt to summarize several approaches aimed at connecting Rough Set Theory with Artificial Neural Networks. Both methodologies have their place among intelligent classification and decision support methods. Artificial Neural Networks belong to most commonly used techniques in applications of Artificial Intelligence. During the last twenty years of its development numerous theoretical and applied works have been done in that field. Rough Set Theory which emerged about fifteen years ago is nowadays rapidly developing branch of AI and Soft Computing.

At the first glance the two methodologies we talk about have not too much in common. Basic rough sets deal with symbolic representation of data, they construct representation of knowledge in terms of attributes, semantic decision rules etc. On the contrary, neural networks in their basic form do not consider the detail meaning of knowledge gained in the process of model construction and learning. But, in spite of those differences it is interesting to try to incorporate both approaches into some combined system. The challenge is to get as much as possible from this association.

This work presents several approaches to the task of incorporating rough set and neural network methods into one system for decision (classification) support. Different results of attempts to preprocess data for a neural network with rough set methods, to construct a the network using knowledge from rough set calculations or to refine rough set results using network are described.

The work is organized as follows:

First section introduces the formalism necessary to describe basic notions of rough sets and neural networks.

Second section presents, using several examples of applications, the attempts to use rough set based methods as a data preprocessor. In those examples data are treated by rough set reduction and then a network is constructed over simplified dataset. Possible advantages and threads of such a way of creating decision support system are briefly discussed.

In the third section we present the concept of incorporating rough sets methods into construction of the neural net by using so called *rough neurons.*

Last section discusses usage of rough set methods and knowledge gained from them in the process of establishing the architecture and initial state of a neural network for a given problem. It touches numerous problems of dealing with continuously-valued features, continuous decision and others.

2 Basic notions

2.1 Rough set preliminaries

The basic notions of rough sets theory are: information system, decision table, reduct and others. We will introduce them now step by step. In order to represent the sets of data we use information systems.

An *information system* is defined by a pair $\mathbf{A} = (U, A)$, where U is a non-empty, finite set of *objects (*rows, records, samples, cases*)* called universe*,* $A = \{a_1, \ldots, a_{n_A}\}$ is a non-empty, finite set of *attributes,* i.e. $a_i : U \to V_{a_i}$ for $i \in \{1, ..., n_A\}$, where V_{a_i} is called *the domain of the attribute* a_i.

In case of real-valued attributes, where for each $i \leq n_A$ $a_i : U \to \Re$ is a real function on the universe U, its elements can be characterized as points:

$$P_u = (a_1(u), a_2(u), ..., a_{n_A}(u))$$

in n_A-dimensional affine space $\Re^{n_A}$.

To deal with tasks formulated as decision making or classification problems we will use the notion of a *decision table.* A decision table is generally an information system with distinguished decision. Formally, decision table is a pair $\mathbf{A} = (U, A \cup \{d\})$, $d \notin A$ where d is called *decision attribute* or *decision.* The elements of A are called *conditions.* We assume that the set V_d of values of the decision d is equal to $\{v_1, \ldots, v_{n_d}\}$ for some positive integer n_d called *the range of d.*The *decision classes* are defined by

$$C_i = \{x \in U : d(x) = v_i\}, \text{ for } i = 1, 2, ..., n_d.$$

They determine the partition $\{C_1, ..., C_{n_d}\}$ of the universe U.

For any information system we can define a relation between objects using their attribute values. For a given attribute a, the objects x, y are *a-indiscernible* if they have the *same value* on a, i.e. $a(x) = a(y)$. In these terms we call two objects indiscernible if one cannot distinguish between them using only the knowledge available in the decision table. This definition can be extended to any subset $B \subseteq A$ by

$$xIND(B)y \Leftrightarrow \forall_{a \in B}\, a(x) = a(y)$$

$IND(B)$ denotes the relation determined on the subset $B \subseteq A$.

Obviously, $IND(B)$ is an equivalence relation . Objects x, y satisfying the relation $IND(B)$ are *indiscernible* by attributes from B. We denote by

$$[x]_{IND(B)} = \{y : \langle x, y\rangle \in IND(B)\}$$

the equivalence class defined by the object $x \in U$.

The notions of an indiscernibility relation and an indiscernible object allow us to introduce key concepts in rough set theory: the *reduct* and the *core*.

A subset B of attribute set A is a *reduct* for A iff

$$IND(B) = IND(A) \text{ and } \forall_{b \in A} IND(B - \{b\}) \neq IND(A)$$

In other words, a reduct is a subset of attributes such that it is enough to consider only the features that belong to this subset and still have the same amount of information. Moreover the reduct have the property of minimality i.e. it cannot be reduced any more without loss in quality of information. There can be of course a lot of reducts for a given information system (decision table), in extreme cases as much as $\left(n \lfloor \frac{n}{2} \rfloor\right)$. Those reducts can intersect or be disjoint. By $RED(A)$ we denote the family of all reducts of a given information system. Reducts with the least possible number of attributes are called *minimal reducts* of A.

With an information system we may also connect the notion of the *core*:

$$CORE(A) = \bigcap RED(A)$$

The core corresponds to this part of information which cannot be removed from the system without loss in knowledge that can be derived from it. The core can be empty if there exist some disjoint reducts of A.

An information space of A is defined by $INF_A = \prod_{a \in A} V_a$. We define the information function $Inf_A : U \to INF_A$ by

$$\mathrm{Inf}_A (x) = (a_1 (u), \ldots, a_{n_A} (u)), \text{for any } u \in U.$$

Any object $u \in U$ is represented by its *information vector* $Inf(u)$.

Every information system $\mathbf{A} = (U, A)$ and a non-empty set $B \subseteq A$ define a *B-information function* by $Inf_B(u) = (a_i(u) : a_i \in B)$ for $u \in U$ and some linear order $A = \{a_1, ..., a_{n_A}\}$. The set $\{Inf_{Bd}(u) : u \in U\}$ is called the *B-information set* and it is denoted by V_B.

We may define indiscernibility relation in other terms using information functions as:

$$IND(B) = \{(u, u') \in U \times U : Inf_B(u) = Inf_B(u')\}$$

The equivalence relation $IND(B)$ is also a useful tool to approximate subsets of the universe U. For any $X \subseteq U$ one can define the lower approximation and the upper approximation of X by:

$$\underline{X} = \{x \in U : [x]_{IND(B)} \subseteq X\} \text{ lower approximation}$$
$$\overline{X} = \{x \in U : [x]_{IND(B)} \bigcap X \neq \emptyset\} \text{ upper approximation .}$$

The pair $(\underline{X}, \overline{X})$ is referred to as the rough set of X.

The *boundary region* of $X \subseteq U$ is defined by $Bd(X) = \overline{X} - \underline{X}$.

For many of applications presented in this work the key feature of decision table is its *consistency*. We will say that decision table is *consistent* if there are no two objects in the table which have the same values of conditional attributes and different decision value. In case the decision table is consistent we have clear decomposition of attribute-value space into decision classes. In the opposite case we have to introduce some fault tolerance measurement to deal with cases that cause inconsistency.

A *decision rule* is a formula of the form:

$$((a_{i_1} = v_1) \wedge (a_{i_2} = v_2) \wedge ... \wedge (a_{i_k} = v_k)) \Rightarrow d = v_d$$

where $a_{i_1}, ..., a_{i_k} \in A$, $v_j \in V_{i_j}$ for $1 \leq j \leq k$, $v_d \in V_d$, $i_1, ..., i_k \in \{1, ..., n_A\}$ and $i_j \neq i_i$ for $i \neq j$. This kind of formula tells us that if the values of conditional attributes are as specified in the left part of the rule then the decision is as given inthe right part (i.e. v_d).

2.2 Neural networks

Artificial neural networks are described in detail in many publications. Here we will not provide detailed definition for all network paradigms to be used. We only briefly outline the main facts about networks that will be further discussed. In most of the applications presented in this work the classical multilayer feed-forward network as described in [2] or [5] is utilized. The most commonly used learning algorithm is backpropagation. By a *sigmoidal excitation function* for a neuron we will understand a mapping of the form:

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

where x represents weighted sum of inputs for a given neuron and β is the coefficient called *gain*, which determines the slope of our function.

3 Pre-processing with rough sets

One method of combining those two approaches is to use rough sets as the preprocessing tool for neural networks. This idea was investigated by several researchers and turned out to be effective for some applications. Herein some of them are presented together with the results of experiments that had been performed on different datasets. More detailed description can be found in papers referred to below.

First of all, we need to understand the basic idea at the basis of our proposed approach. We know that rough set methods allow us to reduce the size of dataset by removing some of the attributes while preserving information included in basic system. We may consider the possibility of reducing the dataset and then performing construction and learning of the neural net.

Obviously, one has to consider several measurements that determine usefulness of methods proposed. Here the ones that matter are:

- Change in the quality of classification before and after using rough set methods for data preprocessing.
- Change in network size measured in number of processing units (neurons) and weights.
- Change in learning effectiveness measured in time (number of steps) necessary to accomplish learning with desired effect.
- Flexibility of trained network (ability to recognize unseen objects).

The main thing that has to be done is to find the connection between the way we make reductions with rough sets and the characteristic of network constructed after these reductions. We should possess some guidelines to avoid considering all possible combinations. The attempts to that task were made and we will look into some of them.

The straightforward approach to this task is based on the following procedure:

1. Take the learning part of the decision table and calculate the set of possibly shortest (preferably minimal) reducts and the core.
2. Reduce the table using some reduct or sum of several calculated reducts i.e. remove from the table attributes not belonging to the reducts chosen.
3. Construct the neural network over the reduced set of data.
4. Perform network learning.
5. Do steps 3-4 until there is no chance for significantly better results of classification with this network.
6. Do steps 2-5 until satisfiable quality is obtained.

This procedure although very simple turned out to be effective for some datasets.

Below the results of experiments with data are presented. We will briefly describe the nature of datasets used for those experiments, outlining also differences between the technique described above and approaches used in particular cases.

Before we will discuss experimental results some fact have to be realized. First of all we have to understand the constraints of the process of reduction. The generic problem of finding a minimal reduct for a given decision table belong to the NP-hard class. It is, in general, equivalent to the problem of minimal prime implicant construction for a given boolean function (see [18]). Therefore, it is necessary to use different approximating techniques in order to obtain the set of minimal or sub-minimal reducts for a table. Among them are heuristic methods as well as genetic algorithms.

In this work we deal with connections between neural networks and rough sets, so it is proper to mention here that an attempt to use neural network for finding minimal reducts has been made. In [17] the method for constructing a Boltzmann machine which searches for a minimal reduct is described. It uses the technique of Simulated Annealing as the engine of optimization process.

Another constraint is the size of attribute value sets. If an attribute describes some continuous measurement then it can posses very large sert of possible values. For the rough set approach this situation is unwanted because in such cases there are many objects that differ on this attribute. Sometimes such a difference is negligible from the point of view of considered real world process, but still, due to equivalence condition in indiscernibility relation rough sets methods treat them as separate cases. Such a property of rough set methods leads to modified approaches that claim to resolve the mentioned problem. One of possible solutions is to use some preprocessing techniques to reduce the size of attribute value sets. Another way of solving such a problem is to introduce somehow weakened indiscernibility relation. We will discuss some of these methods further.

Now let us look at some applications of rough set reduction.

Example 1 Lymphography. This set of data is described in detail in [11]. It contains 148 objects. Each object is described by 18 attributes. Objects belong to 4 decision classes and the distribution among classes is 2,4,61,81. The table is consistent. This example is taken from [3].

Example 2 Picture. This dataset was created from microscopic histological pictures. From every picture 155 binary features were extracted. Data table consists of 216 such objects that are divided into 7 classes corresponding to different types of brain cancer. Table is consistent. Detailed description of this example can be found in [4].

Example 3 Election. This example comes from [3]. The table contains 444 objects. There are 30 conditional attributes. Objects belong to 2 decision classes and the distribution among classes is 211 to 233. The table is consistent.

Example 4 Digits. This data table was created using some feature extraction techniques from the set of 32×32 pixel images. Those images are part of NIST database of handwritten digits. After extraction each of 1000 objects has 80 binary descriptors. There are 10 decision classes representing digits from 0 to 9. Distribution between classes is almost uniform. The table is consistent. This case came from [20] and [21].

Example 5 Onko. This data table describes effects of some oncological examinations. It contains 267 cases representing 12 decision classes. Each case is described by 28 conditional attributes. The table is consistent. This example is described in [3].

Example 6 Volleyball. The important features were extracted from video-recorded games of USA Men's Olympic Volleyball Team in 1993. The 144 cases with 13 conditional attributes and binary decision were examined. The table and results are taken from [22].

Example 7 Buses. This dataset collected at the Mechanical Department, Technical University of Poznań consists of 76 objects. There are 8 attributes and 2 decision classes with 30 and 46 objects respectively. Example came from [3].

Example 8 Production. This data table consist of some characteristics of automated production process. The goal is to foresee the level of possible breakdown in production circuit. Every sample is described by 28 attributes. There are 5 possible decisions. The table used for calculation was consistent but it may change with possible new cases as the modeled situation is time-changing. This example was published in [23] and [24].

Example 9 Breast Cancer. This dataset originally collected at the University Medical Centre, Institute of Oncology, Ljubljana was taken from the well known Machine Learning Database at the University of California, Irvine [10]. 285 cases are described by 8 conditional attributes. There are two possible decisions. This dataset contains some inconsistency. The example came from [3].

The table below summarizes some results of experiments over presented decision tables.

Name	*Input before*	*Input after*	*Net size before*	*Net size after*	*Result after*
Lymphography	35	18	40	24	-2.02%
Picture	155	17	192	53	-1.96%
Election	32	5	26	17	+3.38%
Digits	80	58	189	134	+1.4%
Onko	159	62	72	55	+6.37%
Volleyball	13	6	20	13	-0.6%
Buses	8	5	15	10	-5.27%
Production	28	6	35	13	better
Breast Cancer	14	13	24	24	+0.7%

The results shown in the table require some comments. First of all we have to explain the meaning of particular columns:

- *Input before* corresponds to the number of inputs to the network before rough set reduction.
- *Input after* corresponds to the number of inputs to the network after attribute reduction.
- *Net size before* corresponds to the size of the network before reduction measured in number of neurons. Sometimes also the number of weights is given in braces.
- *Net size after* corresponds to the size of the network after reduction measured in number of neurons. Sometimes also the number of weights is given in braces.
- *Results after.* This column summarizes change in quality of network answers after reduction. As the examples come from different sources, it is in fact impossible to find the common format for representing results.The most common measurement is the change in percentage of misclassified objects. This will be explained separately.

We have to make one more important remark before we step to result explanation. Cautious reader will notice for sure that in some of the examples the number of network inputs (*Input before*) does not match the number of attributes in decision table as described above. This situation occurs for following datasets: *Lymphography, Election, Breast Cancer, Onko*. It is the result of applying encoding procedure to some of the attributes.

Authors in [3] distinguish among others those attributes which have symbolic unordered values. It means that it is impossible to arrange values of particular attribute along some axis of significance. This kind of attributes is very inconvenient for neural network. To resolve possible problems the *one-from-n* encoding is applied. This encoding creates a binary vector whose elements correspond to network inputs. When an attribute takes a particular value, the corresponding vector element is equal to 1, while others to 0. Usage of such encoding allowed to perform extended reduction. If some attributes are encoded using the presented method, then we can take the decision table extended in such way and reduce it (calculate reducts and core). In this case, in fact, we make not only attribute reduction but also attribute domain reduction.

Now we can discuss the results of experiments.

In case of *Lymphography, Election, Breast Cancer, Onko* and *Buses* several experiments using reduction of attributes as well as reduction of attribute domains were performed. In the table the change in the missclassification rate over whole dataset is presented. In order to give better understanding we provide the table below with more detailed description of those cases.

	Election	*Breast Cancer*	*Buses*	*Lymphography*	*Onko*
Change in neurons(%)	-34.6	-12.5	-33.3	-42.8	-23.6
Change in weights(%)	-83.2	-20	-59.6	-82.4	-77.4
Error before(%)	7.21	28.77	6.58	19.59	3.74
Change in error(%)	+3.38	+0.70	-5.26	-2.02	+6.37

Change in neurons in table below is calculated using $(\frac{n_{old}}{n_{new}} \cdot 100) - 100$ where n_{old}, n_{new} represent number of neurons in network before and after reduction respectively. In the same manner **change in weights** is calculated using $(\frac{w_{old}}{w_{new}} \cdot 100) - 100$ where w_{old}, w_{new} represent number of adjustable weights in network before and after reduction respectively.

Error before represents percentage of missclassified samples for network before reduction.

Change in error correspond to the difference in percentage of misclassified cases after and before reduction. We can easily see that in those cases presented in our table significant reduction of network size was achieved. Although the classification error was sometimes bigger for reduced network, the network itself was far more manageable from computational point of view.

In case of *Picture* decision table authors constructed non-reduced neural network and performed some tests using the cross-validation technique. The average quality of reclassification over 10 repeats was 86.15%. After reduction the size of network decreased rapidly. The reduced set of attributes was almost 10 times smaller than original. It allowed to perform more attempts to construct a network. The best one had two hidden layers and achieved 88.07 % accuracy of reclassification on whole set of examples.

The *Digits* example differs slightly from others in method of reduction. For construction of reduced data set several reducts were used. The shortest reduct had 24 attributes. But experiments showed that it would be extremally difficult to construct the network in case only one reduct is used. Therefore reduction was done using the union of several reducts. After several experiments reduction from 80 to 58 attributes turned to be optimal. The loss of quality measured over several cross-validation tests dropped by 1.4% on whole data (1.9% on testing set) and was equal to 93.6% (85.6% respectively). But size of the network was significantly smaller. Moreover the number of learning steps necessary to achieve good classification dropped from 8000 to 4000. Stronger reduction gave better improvement in computational aspects but loss of classification quality was significant.

In the *Volleyball* example the reduction of attributes was compared with other prediction techniques, in particular, with basic rough set approach i.e. calculation of reducts and decision rules. The combination of rough set reduction and neural network produced best classifier among considered. The error rate dropped from 13.73% to 12.67% on the training set. On the testing set, the quality was worse that for non-reduced network by less than 1%. So, overall performance was better but the reduced network lost some ability to recognize unseen objects.

The *Production* data is an example of making time prediction using neural network. In this particular case the goal was not the quality of classification for separate objects, but the degree of similarity between actual curve representing dynamics of production (and possible breaks in circuit) and the one approximated using neural predictor. According to the detailed explanation presented in [23], [24] the neural network based on reduced set of attributes behaved better as it showed better stability. As the unimportant features have been removed, the reduced network reacted only slightly to noisy information.

Summarizing the outcome of presented examples we may say that rough set reduction of network input data turned to be effective especially if we could not overcome the computational problems related to the size of the neural network. It is understandable that for some application we have to accept the trade-off between quality and computational ability.

We mentioned earlier that for some data, especially in case of continuously-valued attributes, rough set methods face some problems. As we will return to this topic further here we would like to provide simple example in order to illustrate the hard part.

Let us consider the well-known simple dataset *Iris* of iris classification. It was first published in 50's by R.A.Fisher and now is available in [10]. The decision table consists of 150 objects, each described by four conditional attributes and one decision attribute. The conditional attributes have numerical values, the decision takes one of three possible states. The simple calculation shows that any three-element subset of the set of attributes is a reduct. Such a situation is easy to predict as the conditions have many different values (large attribute domains). Several computational experiments have been made in attempt to use rough set reduction in this case. But results of experiments clearly showed that removal of any of the attributes causes rapid decrease of network classification quality. In non-reduced case simple network of 14 neurons with two hidden layers gave almost 100% accuracy for learning, testing sets being halves of the whole table. After reduction the best network gave only 75% accuracy on the whole table (as few as 55% on the testing set) even for a network significantly larger than the original, non-reduced one. The only possible conclusion is that rough set methods should be applied to the cases when attributes have large domains together with other methods that allow to preserve the important part of information which is redundant in straightforward rough set approach.

4 Rough set neurons and networks

We have already seen the definition of upper and lower approximations for a given set X of objects in the information system (decision table). To get some intuition about those approximations, it is convenient to think about the upper approximation as the set of object that are *possibly (plausibly)* similar to those in X according to the knowledge taken from decision table. The lower approximation of X is a set of objects that are *with certainty* similar only to the elements of X. The elements that belong to the boundary region are treated as those which may or may not belong to our set.

Driven by the idea of decomposing the set of all objects into three parts: the lower approximation, the boundary region and the outside area with respect to given X, Lingras ([8],[9]) introduced the idea of a rough neuron. Following his definitions we will now show the idea of rough neural network and present some application.

A rough neural network consists of both conventional and *rough* neurons. They form the classical multilayer structure with connections coming from layer to layer.

A *rough neuron* r may be understand as a pair of usual neurons. One of those two is referred as the *upper bound* denoted as $\overline{r}$, the other is called *lower bound* and denoted $\underline{r}$. The upper and lower bounds for a given rough neuron r "overlap" i.e. those two neurons exchange information.

The connections between classical and rough neurons are made as in usual case. While connecting rough neuron with classical one, we connect $\underline{r}$ and $\overline{r}$ separately. The situation starts to be more interesting when we want to connect two rough neurons. As each of rough neurons r, s is in fact a pair $\underline{r}, \overline{r}$ and $\overline{s}, \underline{s}$ respectively, we will distinguish three kinds of possible connections between them. The *full connectionism* occurs iff each of the components of r is connected both to $\overline{s}$ and $\underline{s}$. So in the situation of full connectionism we have together four connections between r and s.Two more possible ways of connecting such neurons are called *excitatory* and *inhibitory*. If the rough neuron r *excites* the activity of rough neuron s (i.e. increase in the output of r will result in the increase of the output of s) then we connect only $\overline{s}$ with $\overline{r}$ and $\underline{s}$ with $\underline{r}$. In the opposite situation, if r *inhibits* the activity of s (i.e. increase in the output of r corresponds to the decrease in the output of s) we connect only $\underline{s}$ with $\overline{r}$ and $\overline{s}$ with $\underline{r}$.

The classical neurons in our rough set network behave as usual. For calculation of their output we use sigmoidal function taken over a weighted sum of incoming signals. In case of the rough neuron we calculate outputs of upper and lower bound neurons using:

$$output_{\overline{r}} = \max\left(f\left(input_{\overline{r}}\right), f\left(input_{\underline{r}}\right)\right)$$

$$output_{\underline{r}} = \min\left(f\left(input_{\overline{r}}\right), f\left(input_{\underline{r}}\right)\right)$$

where f stands for a sigmoidal function and $input_{\overline{r}}, input_{\underline{r}}$ denote collected weighted input i.e.

$$input_i = \sum_{j:\ j \text{ connected with } i} w_{ij} \cdot output_j$$

for a neuron i.

If two rough neurons are partially connected then the excitatory or inhibitory nature of such connection is determined dynamically by polling the connection weights. At the beginning, we can make some assumptions about initial character (excitatory or inhibitory) of the connections. If we have assumed that the partial connection from rough neuron r to another rough neuron s is excitatory and $w_{\overline{r}\overline{s}} < 0$ and $w_{\underline{r}\underline{s}} < 0$, then the connection from rough neuron r to s is changed from excitatory to inhibitory by assigning $w_{\underline{r}\overline{s}} = w_{\underline{r}\underline{s}}$ and $w_{\overline{r}\underline{s}} = w_{\overline{r}\overline{s}}$. The links $(\overline{s}, \overline{r})$ and $(\underline{s}, \underline{r})$ are disabled while links $(\underline{s}, \overline{r})$ and $(\overline{s}, \underline{r})$ are enabled. On the other hand if neuron r is assumed to have an inhibitory partial connection to s and $w_{\underline{r}\overline{s}} > 0$ and $w_{\overline{r}\underline{s}} > 0$ then the connection between rough neurons r and s is changed from inhibitory to excitatory by assigning $w_{\underline{r}\underline{s}} = w_{\underline{r}\overline{s}}$ and $w_{\overline{r}\overline{s}} = w_{\overline{r}\underline{s}}$. The links $(\underline{s}, \overline{r})$ and $(\overline{s}, \underline{r})$ are disabled while links $(\overline{s}, \overline{r})$ and $(\underline{s}, \underline{r})$ are enabled.

The learning process for the network introduced above is based on the classical backpropagation paradigm. We tend to decrease the rate of error over the part of available examples that form the training set. As we perform the supervised learning and the desired values of network outputs over training samples are known, calculation of error is not a problem. In most cases this error is just a difference between expected and received network output. As all the neurons, both classical and rough in our network, use sigmoidal excitation function, the backpropagation step in learning process is also relatively easy to perform. Weights in the network are adjusted according to the simple backpropagation scheme (no momentum, no cumulative effects) using the equation:

$$w_{ji}^{new} = w_{ji}^{old} + \alpha \cdot err_i \cdot f'(input_i)$$

where f' is the derivative of sigmoidal function, α is the learning coefficient and err_i is an error for i-th neuron. Due to the properties of sigmoidal function, calculation of $f'(x) = f(x) \cdot (1 - f(x))$ is also easy. We perform learning by checking if trained network gives the required result for cases from testing set that were not presented to the network during training.

Having in mind the above construction of rough neurons let us look how this idea was utilized in some practical application. In [8] the dataset containing information about traffic parameters called DHV is described. The task is to predict the volume of traffic using data about this volume from the last week.

The classical neural network (Conventional) constructed for this task has seven input neurons corresponding to the values in particular days of previous week, four neurons in hidden layer and one output neuron. The neurons in this typical network are fully connected.

Two different networks to solve this problem were constructed using the idea of rough neuron. First of them (Rough 1) had rough neurons only in input layer. This network had seven input rough neurons, eight hidden conventional neurons and one output conventional neuron. In fact this particular network was only an extension of the normal model because it contained no connections between rough neurons. The second rough network model (Rough 2) had seven input neurons, four hidden rough neurons and one output classical neuron. The important difference in rough network approach is that they take as the inputs the upper and lower bounds for attributes. So in fact this network has twice the number of inputs as compared to the conventional one.

The table below presents results obtained for the traffic data using three described networks.

$Network\ model$	$Training\ set$ $Max.\ Error$	$Training\ set$ $Avg.\ Error$	$Testing\ set$ $Max.\ error$	$Testing\ set$ $Avg.\ error$
$Conventional$	46.2%	9.6%	28.1%	9.7%
$Rough1$	17.5%	5.5%	24.9%	8.1%
$Rough2$	13.7%	5.8%	23.0%	8.0%

5 Rough sets and discretization in network construction

So far, we have seen utilization of some simple rough set concepts in creation of neural networks. Now we would like to deal with a little bit more complicated task. Rough set methods give us the possibility to search for classifiers defined in terms of decision rules, reducts, discernibility etc. It is natural that equipped with such a knowledge we should be able to construct the neural network with better initial architecture than the one constructed without such guidelines. We are eager to reduce the exhausting stage of designing proper network architecture by applying some automated technique which utilizes knowledge about data that we already have. Secondly, the network itself does not provide us with clear interpretation of knowledge it contains ([19]). Fortunately, rough set methods ([15],[18]) can help to construct initial network in terms of such parameters like the numbers of scaling conditions, minimal decision rules and decision classes in discrete case.

As mentioned above the classical rough set approach faces difficulties when confronted with continuously valued attributes. Therefore we will present some discretization (quantization) techniques that allow to produce attributes with small, discrete sets of values preserving information included in original, real-valued decision table. The presented approach comes from [16].

5.1 Hyperplane discretization

The main problem of such discretization is how to approximate decision classes $\{C_1, ..., C_{n_d}\}$ by possibly small and regular family of subsets $\tau_k \subseteq \Re^{n_A}$, where any k points to some decision value $v_{l(k)}$ e.g. in terms of its high frequency of occurrence for objects in τ_k.

In [14] searching for such decision rules was performed by defining hyperplanes over $\Re^n$. Any hyperplane

$$H = \{(x_1, x_2, ..., x_n) \in \Re^{n_A} : \alpha_0 + \alpha_1 x_1 + \cdots + \alpha_{n_A} x_{n_A} = 0\}$$
$$\text{where } \alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n_A} \in \Re$$

splits C_l into two subclasses defined by:

$$C_l^{U,H} = \{u \in C_l : H(u) \geq 0\}$$

$$C_l^{L,H} = \{u \in C_l : H(u) < 0\}$$

where, for a given hyperplane, the function

$$H : U \to \Re$$

is defined by

$$H(u) = H(Inf_A(u))$$

Let us propose some measures estimating the quality of hyperplanes with respect to the decision classes $C_1, C_2, ..., C_{n_d}$. Consider the function $award(H) =$

$$\sum_{l1 \neq l2} card\left(C_{l1}^{U,H}\right) \cdot card\left(C_{l2}^{L,H}\right) \tag{1}$$

If $award(H) > award(H^{'})$ for some hyperplanes H, H', then the number of discernible pairs of objects from different decision classes by H is greater than the corresponding number defined by $H^{'}$. Thus, this is H which should be considered while building decision rules.

In view of large complexity of searching for fixed number of hyperplanes simultaneously, the following sequential algorithm was implemented.

1. Find optimal hyperplane H_1 with respect to *award*.

2. Find hyperplane H_2 by maximizing function $award(H/H_1) =$

$$\sum_{case=L,U} \sum_{l1 \neq l2} card\left(C_{l1}^{U,H} \cap C_{l1}^{case,H_1}\right) \cdot card\left(C_{l2}^{L,H} \cap C_{l1}^{case,H_1}\right)$$

3. Repeat the above step considering function $award(H/H_1, ..., H_j)$ constructed for hyperplanes found step by step, until obtaining satisfactory degree of decision classes' approximation for some number n_h of hyperplanes.

Remark. Function (1) can be combined with parameters like e.g.

$$penalty(H) = \sum_{l=1}^{n_d} card\left(C_l^{U,H}\right) \cdot card\left(C_l^{L,H}\right)$$

or replaced by others, with respect to requirements.

Remark. The number of decision rules, equal to 2^{n_h} due to all possible combinations of position of objects with respect to n_h hyperplanes, can be reduced to the number $n_r \leq 2^{n_h}$ of minimal decision of the form $\tau_k \Rightarrow d = v_{l(k)}$, where no component τ_{kj} corresponding to hyperplane H_j can be rejected without decrease in given degree of approximation.

A method for generating hyperplanes, its advantages and limitations is also described in detail in one of the chapters in this book, namely the one authored by Nguyen Hung Son.

5.2 Hyperplane-based network

Once the hyperplanes and decision rules are constructed for a given $\mathbf{A}$, we may put them into the neural network.

Proposition 1. *Given a decision table* $\mathbf{A} = (U, A \cup \{d\})$ *and the set of* n_h *hyperplanes inducing* n_r *decision rules, one can construct four-layer neural network with* $n_A + 1$ *inputs,* n_h *and* n_r *neurons in hidden layers respectively, and with* n_d *outputs, such that it recognizes objects in* U *just like in the case of corresponding hyperplane decision tree.*

Proof. The network has n_A inputs corresponding to conditional attributes. There is also one additional constant input called bias. Every input neuron sends its signal to all neurons in hidden layer. For each hyperplane H we construct one neuron in hidden layer. This neuron has weights equal to coefficients describing corresponding hyperplane.

For all neurons in the first hidden layer the threshold functions have the same form

$$h_j(x) = \begin{cases} 1 \; for \; x \geq 0 \\ -1 \; for \; x < 0 \end{cases}$$

This is also the case for thresholds in the second hidden layer, which are given as

$$r_k(x) = \begin{cases} 1 \; for \; x \geq 1 \\ 0 \; for \; x < 1 \end{cases}$$

Neurons in this layer correspond to binary hyperplane decision rules. The weights connecting these two layers correspond to the way of occurrence of hyperplane attributes in rules. For instance, let the 5-th minimal decision rule τ_5 be of the form

$$(H_2(u) < 0) \;\&\; (H_4(u) \geq 0) \;\&\; (H_7(u) < 0) \Rightarrow d(u) = v_4 \tag{2}$$

Then the corresponding weights leading to the 5-th neuron in the second hidden layer take the following values:

$$w_{j5} = \begin{cases} \frac{1}{3} \; for \; j = 4 \\ -\frac{1}{3} \; for \; j = 2 \, or \, 7 \\ 0 \; otherwise \end{cases} \tag{3}$$

Thus, according to the above example, the 5-th neuron in the second hidden layer will be active (its threshold function will reach 1) for some $u \in U$ iff u satisfies conditions of the above decision rule.

For every decision value we construct one neuron in output layer, so together n_d outputs from the network. The l-th output is supposed to be active iff given object put into the network belongs to corresponding decision class C_l. To achieve such a behavior we link every decision rule neuron only with the output neuron corresponding to decision value indicated by decision rule. Thus, in case of our example, the weights between the 5-th neuron in the second hidden layer and the output layer are as follows:

$$w_{5l} = \begin{cases} 1 \; for \; l = 4 \\ 0 \; otherwise \end{cases}$$

All neurons in the output layer receive threshold functions

$$out_l(x) = \begin{cases} 1 \; for \; x \geq 1 \\ 0 \; for \; x < 1 \end{cases}$$

To give some intuition how this method of network construction works, let us take a brief look at the iris classification example presented above. As decision in this case has three possible values, our universe should be decomposed into three decision classes. For Iris data, decision classes are linearly separable except for two objects. But there exists a single hyperplane distinguishing one of decision classes from the others. The remaining two classes can be distinguished using simple hyperplane if we allow two mentioned objects to be missclassified or else we have to use more than one hyperplane. In case we want 100% accuracy the network constructed using technique from above will have 4 inputs, 4 neurons in first hidden layer (as 4 hyperplanes are necessary to completely decompose universe), 5 neurons in second hidden layer corresponding to decision rules and finally 3 output neurons corresponding to the decision.

5.3 Modifications of the weights

The above neural network, although clear and valid in its construction, does not express as much yet as it could. First of all, it does not deal with non-deterministic decision rules which are often the only way to derive any information from data. Let us go back to the example of decision rule (2) and assume that it was stated with some degree of approximation not less than 0.9, where the value

$$P\left(d = v_4 \,|\, H_2 < 0, H_4 \geq 0, H_7 < 0\right) = 0.9$$

corresponds to the frequency of occurrence of v_4 as a decision value for the subspace

$$C_4^{L,H_2} \cap C_4^{U,H_4} \cap C_4^{L,H_7}$$

corresponding to conditions of decision rule. In this case we propose to replace previous output functions by

$$out_l\left(x\right) = x$$

and link output neurons with weights w_{kl} corresponding to frequency of decision value v_l conditioned by decision rule τ_k. Then, answering with a decision value with the highest value of the output function, we obtain the same classification as in case of decision rules. Additional information about degrees of approximation for applied rules can be derived as well.

One should realize that in case of non-deterministic rules frequencies of decision values may be often similar under given conditions. In fact, to evaluate degrees of approximation for non-deterministic decision rules, we need a measure not corresponding to concrete decision values, like e.g.

$$Q\left(\tau_k\right) = \sum_{v_l \in V_d} \left(P\left(d = v_l | \tau_k\right)\right)^q \qquad (4)$$

where $q > 1$, τ_k decision rule. Now, one can express the meaning of particular hyperplanes with respect to the given decision rule by computing the change of Q caused by rejecting particular hyperplane conditions. Let us denote by ρ_{kj} decision rule τ_k without the j-th component τ_{kj}. Then, for any $j = 1, .., n_h$ and $k = 1, .., n_r$ we would like to put

$$w_{jk} = \pm \frac{1}{N_k} \cdot \left(Q\left(\tau_k\right) - Q\left(\rho_{kj}\right)\right)$$

Remark. If one regards function (4) as the degree of approximation of decision classes, then the factor $1/N_k$ is due to normalize weights coming into the neuron corresponding to the k-th decision rule. Due to remark 5.1, each decision rule is minimal in the sense that Q may only decrease after rejecting any hyperplane condition. Thus, the sign $\pm$ is adjusted just for denoting the position of points in τ_k with respect to the j-th hyperplane (compare with (3)).

5.4 Interpretation of neuron functions

To improve flexibility of learning, replacing original threshold functions with continuous ones should be performed. In fact, such a change enables to encode more information within our network model. Let us consider the class of (rescaled) bipolar sigmoidal functions of the form

$$h_j(x) = \frac{2}{1 + e^{-\alpha_j x}} - 1$$

for hyperplane layer. Parameters α_j express degrees of vagueness for particular hyperplanes. Parallel nature of computations along the neural network justifies searching for such parameters locally for each H_j with respect to other hyperplanes, by applying adequate statistical or entropy-based methods (compare with [6],[25]).

Degrees of vagueness, proportional to the risk of basing on corresponding hyperplane cuts, find very simple interpretation. Let us weaken decision rule thresholds by replacing the initial function r_k by

$$r_k(x) = \begin{cases} 1 \; for \; x \geq 1 - \varepsilon_k \\ 0 \; for \; x < 1 - \varepsilon_k \end{cases}$$

where parameter ε_k expresses the degree of belief in decision rule supported by τ_k or, more precisely, in the quality of hyperplanes which generate it. Then, for fixed ε_k, increasing α_j for some H_j occurring in τ_k implies that for objects which are "uncertain" with respect to the j-th cut function r_k equals to 0 and no classification is obtained.

If one wants to modify functions in the second hidden layer similarly as in the first, the idea of extracting initial weights from the degrees of precision for reasoning with given hyperplanes as conditions should be followed. We claim that formulas for the decision rule functions should be derived from the shapes of functions in the previous layer. Thus, for function

$$r_k(x) = \frac{1}{1 + e^{-\beta_k x}}$$

corresponding to the decision rule τ_k, the quantity of β_k is given by formula

$$\beta_k = \sum_{j=1}^{h} \alpha_j \cdot |w_{jk}|$$

5.5 Tuning of conditional hyperplanes

Modifications introduced for initial model of hyperplane-based neural network enable to include necessary information for improvement of decision classification. Obviously, described changes may cause that our network becomes inconsistent with decision rules for some part of training objects. It means that, e.g. for majority frequency rules, the output corresponding to a decision value pointed by some rule may not be the one with the highest value of the output function.

Such inconsistency, however, is justified by computing all weights and neuron functions from decision table itself. Moreover, we have still possibility of tuning the network by the wide range of learning techniques.

In classical backpropagation networks ([2],[5]) update of weights is based on gradient descent technique. The backpropagation method allows us to perform learning by minimizing any differentiable error function δ. The update for any weight w in the network is given by:

$$\Delta w = -\eta \frac{\partial \delta}{\partial w}$$

where η is a learning coefficient.

To be in agreement with the way of computing initial weights in the learning process, we consider error functions of the form

$$\delta (u) = \frac{1}{q} \sum_{1 \leq l \leq n_d} (out_l (u) - in_l (u))^q$$

where $out_l (u) =$

$$\sum_{1 \leq k \leq n_r} w_{kl} \left(1 + \exp \left(-\beta_k \sum_{1 \leq j \leq n_h} w_{jk} \left(\frac{2}{1 + \exp(-\alpha_j H_j (u))} - 1\right)\right)\right)^{-1}$$

and

$$in_l (u) = \begin{cases} 1 \; for \; d(u) = v_l \\ 0 \; otherwise \end{cases}$$

We can also use the cumulative error given by $\delta (U) =$

$$\frac{1}{q \cdot card (U)} \sum_{u \in U} \sum_{1 \leq l \leq n_d} (out_l (u) - in_l (u))^q$$

In this case we back-propagate the global error from the whole set of objects.

Once more we would like to stress that error functions given above correspond to the quality measure Q introduced before. Thus, if one would like to consider hyperplane decision rules minimal in sense of another criterion, the way of measuring classification error should be verified properly.

In classical neural network learning we may manipulate with some coefficients to control the learning process ([2],[5]). In presented approach we may use this ability in order to introduce some meaning for such operations. Change of weights in the first hidden layer corresponds to the change of elevation of hyperplanes. Hence, by setting constraints for value of learning coefficients we may induce the learning in case we e.g. do not want the hyperplanes to change too rapidly. The standard heuristics in the area of network learning, like momentum factor ([5]), can also be used, although they do not have explicit interpretation in terms of hyperplanes and decision trees.

During the learning process we should still remember about the interpretation of weights and functions. Starting from the initial structure obtained from data by the sequential algorithm for finding hyperplanes, we begin to modify weights due to given learning method. Then, however, for possibly improved classification we cannot determine how the decision rules behave over data actually. Another point is to keep decision rules minimal for foregoing hyperplane weights to make the whole process more clear. Thus, it turns out to be very important to preserve the balance between what is derived from the learning process and what is obtained from described construction.

5.6 Searching for optimal decision scaling

From the very beginning of this section we considered decision tables with real-valued conditions and discrete decision with n_d possible values. Such a case, occurring in many classification problems, becomes much more complicated when decision attribute d is real-valued as well. Obviously, one can assume some initial scaling over d and perform the decision process just like before. However, although sometimes such a scaling is given, in many applications we do not need to scale properly but also reason with real values after obtaining decision rules.

Objectives of proper decision scaling create wide range of often contradictive requirements. One of possible methods is to scale decision attribute to obtain a small number of hyperplane-based decision rules. In such a case, however, derived rules may be not precise or safe enough to apply in real-life situation. One solution is to scale decision attribute uniformly under some assumed degree of scaling precision expressed by n_d and to construct hyperplane-based neural network for n_d outputs. Then, we obtain some kind of parallel fuzzy inference model with continuous excitation functions corresponding to the states of binary fuzzy variables ([7],[19]).

Now, there are two methods of obtaining the proper decision system for a given data table. The first one is to synthesize a corresponding neural network by methods described previously, where the new error function is defined by

$$\delta(u) = \frac{1}{q}\left(\frac{\sum_{1\leq l\leq n_d} v_l \cdot (out_l(u) - in_l(u))}{\sum_{1\leq l\leq n_d} out_l(u)}\right)^q \tag{5}$$

The main disadvantage of such an approach is, however, the lack of information about the quality of initial decision scaling. Thus, although we can obtain quite effective model for reasoning, it is a black box because our knowledge about dependencies within data becomes unclear.

Another possibility is to use the network constructed and tuned for the scaled decision attribute to improve the scaling itself. In this case we tune the values v_l corresponding to particular outputs to minimize function (5). As such a optimization process is very complex, we propose to use some heuristics like e.g. genetic algorithm ([12]), where each chromosome in any evolution step corresponds to some scaling of decision attribute. The length of chromosomes is due to initially assumed exactness of scaling and the fitness of any individual is oppositely proportional to the quantity (5) computed from the network.

Remark. During the evolution process the weights of the neural network remain constant as expressing linguistic rules ([19]) corresponding to such a fuzzy-neural inference. However, parameters of excitation functions may be sometimes modified according to the changes of the scaled decision values for the points in $\Re^{n_A}$. Another solution is to optimize these parameters in parallel with decision scaling. It leads, however, to considering longer chromosomes in population steps.

The presented above methodology is still in the process of development and it does require thorough experimental verification. There are some other recent results of application of rough set methods in design of fuzzy MLP's (Multi Layer Perceptrons), but they do not touch the problem of real-valued attributes and decisions. For reference see [1].

Acknowledgement First of all I want to thank professor Andrzej Skowron for the invitation to write this text. I want to thank the authors of papers I cite in this work. They really did me a favor by providing the information about their previous and current research. Many thanks to my colleagues Nguyen Hung Son and Dominik Ślęzak who contributed a lot to my work.
This work was partially supported by the grant No 08T11C01011 from National Committee for Scientific Research and by the ESPRIT project 20288 CRIT-2.

References

1. Banerjee, M., Mitra, S., Pal, S.K. Rough fuzzy MLP: Knowledge encoding and classification. IEEE Transactions on Neural Networks(1997) (submitted)
2. Hecht-Nielsen, R.: Neurocomputing. Addison-Wesley, New York (1990)
3. Jelonek, J., Krawiec, K., Słowiński, R.: Rough set reduction of attributes and their domains for neural networks, Computational Intelligence **11/2** (1995) 339–347
4. Jelonek, J., Krawiec, K., Słowiński, R., Stefanowski, J., Szymaś, J.: Rough sets as an intelligent front-end for the neural network. In: Proceedings of First National Conference "Neural Networks and their Applications", April 12-15, Kule (1994) 268–273
5. Karayiannis, N.B., Venetsanopoulos, A.N.: Artificial neural networks: Learning algorithms, performance evaluation and applications. Kluwer, Dortrecht (1993)
6. Kohavi, R., Sahami, M.: Error–based and entropy–based discretization of continuous features. In: E. Simoudis, J. Han, and U.M. Fayyad (eds.): Proc. of the Second International Conference on Knowledge Discovery & Data Mining. Portland, Oregon (1996) 114–119
7. Kruse, R., Gebhardt, J., Klawonn F.: Foundations of fuzzy systems. Wiley, Chichester (1994)
8. Lingras, P.: Rough neural networks. In: Proceedings of the Sixth International Conference, Information Procesing and Management of Uncertainty in Knowledge-Based Systems (IPMU'96), July 1-5, Granada, Spain (1996) **3** 1445–1450
9. Lingras, P.: Comparison of neofuzzy and rough neural networks. In: P.P. Wang (ed.): Proceedings of the Fifth International Workshop on Rough Sets and Soft Computing (RSSC'97) at Third Annual Joint Conference on Information Sciences (JCIS'97), Duke University, Durham, NC, USA, Rough Set & Computer Science **3**, March 1–5 (1997) 259–262

10. Machine learning databases, University of California, Irvine. ftp://ics.uci.edu/machine-learning-databases
11. Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N.: The multi–purpose incremental learning system AQ15 and its testing applications to three medical domains. In: Proc. of 5 National Conference on Artificial Intelligence, Philadelphia, Morgan-Kaufman, (1986) 1041-1045
12. Michalewicz, Z.: Genetic algorithms + data structures = evolution programs. Springer–Verlag, Berlin (1992)
13. Nguyen, H.Son, Nguyen, S. Hoa: From optimal hyperplanes to optimal decision tree. In: S. Tsumoto, S. Kobayashi, T. Yokomori, H. Tanaka, and A. Nakamura (eds.): Proceedings of the Fourth International Workshop on Rough Sets, Fuzzy Sets, and Machine Discovery (RSFD'96), The University of Tokyo, November 6–8 (1996) 82–88
14. Nguyen, H. Son., Nguyen, S. Hoa, Skowron, A.: Searching for features defined by hyperplanes. In: In: Z.W. Ras, M. Michalewicz (eds.), Ninth International Symposium on Methodologies for Intelligent Systems. Zakopane, Poland, June 9–13, Lecture Notes in Artificial Intelligence (ISMIS'96) **1079**, Springer–Verlag, Berlin (1996) 366–375
15. Nguyen, H.Son, Skowron, A.: Quantization of real-valued attributes. Rough Set and Boolean Reasoning Approaches. In: P.P. Wang (ed.): Second Annual Joint Conference on Information Sciences (JCIS'95), Wrightsville Beach, North Carolina, 28 September - 1 October (1995) 34–37
16. Nguyen, H. Son, Szczuka, M., Ślęzak, D.: Neural networks design: Rough set approach to real-valued data. In: J. Komorowski, J. Zytkow, (eds.), The First European Symposium on Principle of Data Mining and Knowledge Discovery (PKDD'97), June 25–27, Trondheim, Norway, Lecture Notes in Artificial Intelligence **1263**, Springer-Verlag, Berlin (1997) 359–366
17. Sapiecha, P.: An approximation algorithm for certain class of NP-hard problems. In: ICS Research Report **21/92** Warsaw University of Technology (1992)
18. Skowron, A., Rauszer, C.: The discernibility matrices and functions in information systems. In: R. Słowiński (ed.): Intelligent Decision Support – Handbook of Applications and Advances of the Rough Sets Theory, Kluwer Academic Publishers, Dordrecht (1992) 331–362
19. Szczuka, M., Ślęzak, D.: Hyperplane-based neural networks for real-valued decision tables. In: P.P. Wang (ed.): Proceedings of the Fifth International Workshop on Rough Sets and Soft Computing (RSSC'97) at Third Annual Joint Conference on Information Sciences (JCIS'97), Duke University, Durham, NC, USA, Rough Set & Computer Science **3**, March 1–5 (1997) 265–268
20. Szczuka, M.: Aproksymacja funkcji za pomocą sieci neuronowych z wykorzystaniem metod zbiorów przybliżonych. Master Thesis, Faculty of Mathematics, Informatics and Mechanics, The University of Warsaw (1995)
21. Szczuka, M.: Rough set methods for constructing artificial neural networks. In: B.D. Czejdo, I.I. Est, B. Shirazi, B. Trousse (eds.), Proceedings of the Third Biennial European Joint Conference on Engineering Systems Design and Analysis **7**, July 1-4, Montpellier, France (1996) 9–14
22. Świniarski, R., Berzins, A.: Rouh sets expert system for on-line prediction of volleyball game progress. In: B.D. Czejdo, I.I. Est, B. Shirazi, B. Trousse (eds.), Proceedings of the Third Biennial European Joint Conference on Engineering Systems Design and Analysis **7**, July 1-4, Montpellier, France (1996) 3–8

23. Świniarski, R., Hunt, F., Chalvet, D., Pearson, D.: Prediction system based on neural networks and rough sets in a highly automated production process. In: Proceedings of the 12^{th} System Science Conference, Wrocław, Poland (1995)
24. Świniarski, R., Hunt, F., Chalvet, D., Pearson, D.: Intelligent data processing and dynamic process discovery using rough sets, statistical reasoning and neural networks in a highly automated production systems. In: Proceedings of the First European Conference on Application of Neural Networks in Industry, Helsinki, Finland (1995)
25. Vapnik, V.N.: The nature of statistical learning theory. Springer–Verlag, New York (1995)

This article was processed using the LaTeX macro package with LMAMULT style