Using Neural Network Rule Extraction and Decision Tables for Credit-Risk Evaluation

Bart Baesens • Rudy Setiono • Christophe Mues • Jan Vanthienen
Department of Applied Economic Sciences, K. U. Leuven, Naamsestraat 69, B-3000 Leuven, Belgium
Department of Information Systems, National University of Singapore, Kent Ridge, Singapore 119260, Republic of Singapore
Department of Applied Economic Sciences, K. U. Leuven, Naamsestraat 69, B-3000 Leuven, Belgium
Department of Applied Economic Sciences, K. U. Leuven, Naamsestraat 69, B-3000 Leuven, Belgium
bart.baesens@econ.kuleuven.ac.be • rudys@comp.nus.edu.sg
christophe.mues@econ.kuleuven.ac.be • jan.vanthienen@econ.kuleuven.ac.be

---

Credit-risk evaluation is a very challenging and important management science problem in the domain of financial analysis. Many classification methods have been suggested in the literature to tackle this problem. Neural networks, especially, have received a lot of attention because of their universal approximation property. However, a major drawback associated with the use of neural networks for decision making is their lack of explanation capability. While they can achieve a high predictive accuracy rate, the reasoning behind how they reach their decisions is not readily available. In this paper, we present the results from analysing three real-life credit-risk data sets using neural network rule extraction techniques. Clarifying the neural network decisions by explanatory rules that capture the learned knowledge embedded in the networks can help the credit-risk manager in explaining why a particular applicant is classified as either bad or good. Furthermore, we also discuss how these rules can be visualized as a decision table in a compact and intuitive graphical format that facilitates easy consultation. It is concluded that neural network rule extraction and decision tables are powerful management tools that allow us to build advanced and user-friendly decision-support systems for credit-risk evaluation.
(*Credit-Risk Evaluation; Neural Networks; Decision Tables; Classification*)

---

1. Introduction

One of the key decisions financial institutions have to make is to decide whether or not to grant a loan to a customer. This decision basically boils down to a binary classification problem which aims at distinguishing good payers from bad payers. Until recently, this distinction was made using a judgmental approach by merely inspecting the application form details of the applicant. The credit expert then decided upon the creditworthiness of the applicant, using all possible relevant information concerning his sociodemographic status, economic conditions, and intentions. The advent of data storage technology has facilitated financial institutions' ability to store all information regarding the characteristics and repayment behaviour of credit applicants electronically. This has motivated the need to automate the credit-granting decision by using statistical or machine-learning algorithms.

Numerous methods have been proposed in the literature to develop credit-risk evaluation models. These models include traditional statistical methods (e.g., logistic regression, Steenackers and Goovaerts 1989), nonparametric statistical models (e.g., k-nearest neighbour, Henley and Hand 1997, and classification trees, Davis et al. 1992) and neural network models (Desai et al. 1996). Most of these studies focus primarily on developing classification models with

Management Science © 2003 INFORMS
Vol. 49, No. 3, March 2003 pp. 312–329
0025-1909/03/4903/0312$5.00
1526-5501 electronic ISSN

high predictive accuracy without paying any attention to explaining how the classifications are being made. Clearly, this plays a pivotal role in credit-risk evaluation, as the evaluator may be required to give a justification for why a certain credit application is approved or rejected. Capon (1982, p. 90) was one of the first authors to argue that credit-risk evaluation systems should focus more on providing explanations for why customers default instead of merely trying to develop scorecards which accurately distinguish good customers from bad customers:

> What is needed, clearly, is a redirection of credit scoring research efforts toward development of explanatory models of credit performance and the isolation of variables bearing an explanatory relationship to credit performance.

Furthermore, there is often also a legal obligation to justify why credit has been denied. The Equal Credit Opportunities Act (1976) and Regulation B in the United States prohibit the use of characteristics such as gender, marital status, race, whether an applicant receives welfare payment, colour, religion, national origin, and age, in making the credit decision (Crook 1999). Hence, the issue of making credit-risk evaluation systems intelligent and explanatory is becoming more and more a key success factor for their successful deployment and implementation.

In this paper, we report on the use of neural network rule extraction techniques to build intelligent and explanatory credit-risk evaluation systems. While neural networks have been used before for this purpose (e.g., West 2000), there is still no consensus on their superiority with respect to more traditional statistical algorithms such as logistic regression. Although their universal approximation property seems attractive at first sight, their intrinsically black-box nature has prevented them from being successfully applied in a management science setting. This refers to the fact that they do not allow formalization of the relationship between the outputs and the inputs in a user-friendly, comprehensible way. Neural networks are therefore commonly described as opaque structures because they generate complex mathematical models which relate the outputs to the inputs using a set of weights, biases, and nonlinear activation functions which are hard for humans to interpret.

Recent developments in algorithms that extract rules from trained neural networks enable us to generate classification rules that explain the decision process of the networks. The purpose of our research is to investigate whether these neural network rule extraction techniques can generate meaningful and accurate rule sets for the credit-risk evaluation problem. We conduct experiments on three real-life credit-risk evaluation data sets. In this context, three popular neural network rule extraction techniques, Neurorule (Setiono and Liu 1996), Trepan (Craven and Shavlik 1996), and Nefclass (Nauck 2000), are evaluated and contrasted. The performance of these methods is compared with the C4.5(rule) decision-tree (rules) induction algorithm as well as the widely used logistic regression classifier. In a subsequent step of the decision-support system development process, the extracted rules are represented as a decision table (DT) (Vanthienen and Wets 1994, Wets et al. 1997). This is motivated by the fact that research in knowledge representation suggests that graphical representation formalisms (such as DTs) can be more readily interpreted and consulted by humans than symbolic rules (Santos-Gomez and Darnel 1992). Representing the knowledge learned by the neural networks as a decision table allows the visualization of the rules in a format that is easily comprehensible and verifiable by the credit-risk manager. Hence, in this paper, we also investigate the usefulness of DTs as an intuitive graphical visualization aid for credit-risk evaluation purposes.

This paper is organized as follows. In §2, we briefly explain the basic concepts of neural networks and discuss the Neurorule, Trepan, and Nefclass algorithms. Section 3 presents the empirical setup and the rule extraction results. The subsequent use of decision tables is discussed in §4. Conclusions are drawn in §5.

2. Neural Networks and Rule Extraction

2.1. Neural Networks

Neural networks are mathematical representations inspired by the functioning of the human brain. Many types of neural networks have been suggested in the literature for both supervized and unsupervized learning (Bishop 1995). Because our focus is on classification, we will discuss the Multilayer Perceptron (MLP) neural network in more detail.

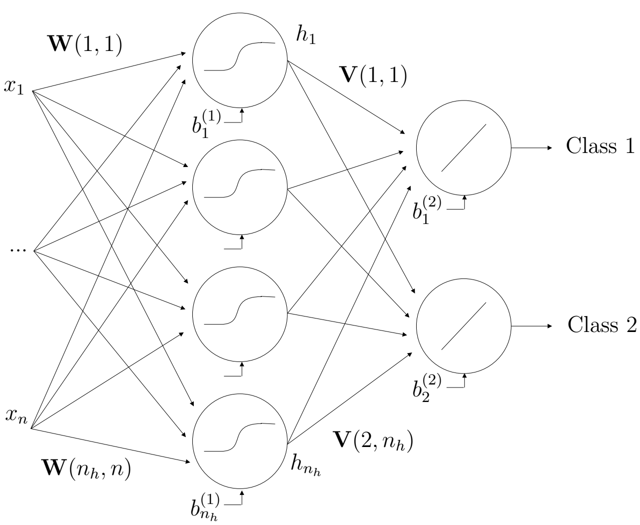
An MLP is typically composed of an input layer, one or more hidden layers, and an output layer, each consisting of several neurons. Each neuron processes its inputs and generates one output value which is transmitted to the neurons in the subsequent layer. One of the key characteristics of MLPs is that all neurons and layers are arranged in a feedforward manner and no feedback connections are allowed. Figure 1 provides an example of an MLP with one hidden layer and two output neurons for a binary classification problem. The output of hidden neuron i is computed by processing the weighted inputs and its bias term $b_i^{(1)}$ as follows:

$$h_i = f^{(1)}\left(b_i^{(1)} + \sum_{j=1}^{n} \mathbf{W}(i,j)x_j\right). \tag{1}$$

$\mathbf{W}$ is a weight matrix, whereby $\mathbf{W}(i,j)$ denotes the weight connecting input j to hidden unit i. In an analogous way, the output of the output neurons is computed:

$$z_i = f^{(2)}\left(b_i^{(2)} + \sum_{j=1}^{n_h} \mathbf{V}(i,j)h_j\right), \tag{2}$$

Figure 1 **Architecture of a Multilayer Perceptron**



with n_h the number of hidden neurons and $\mathbf{V}$ a weight matrix, whereby $\mathbf{V}(i,j)$ denotes the weight connecting hidden unit j to output unit i. The bias inputs play a role analogous to that of the intercept term in a classical linear regression model. The class is then assigned according to the output neuron with the highest activation value (*winner take all learning*). The transfer functions $f^{(1)}$ and $f^{(2)}$ allow the network to model nonlinear relationships in the data. Examples of transfer functions that are commonly used are the sigmoid

$$f(x) = \frac{1}{1+\exp(-x)},$$

the hyperbolic tangent

$$f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)},$$

and the linear transfer function $f(x) = x$.

The weights $\mathbf{W}$ and $\mathbf{V}$ are the crucial parameters of a neural network and need to be estimated during a training process which is usually based on gradient-descent learning to minimize some kind of error function over a set of training observations (Bishop 1995). Note that multiple hidden layers might be used, but theoretical works have shown that one hidden layer is sufficient to approximate any continuous function to any desired degree of accuracy (*universal approximation property*) (Bishop 1995).

As universal approximators, neural networks can achieve significantly better predictive accuracy compared to models that are linear in the input variables. However, their complex mathematical internal workings prevent them from being used as effective management tools in real-life situations (e.g., credit-risk evaluation) where besides having accurate models, explanation of the predictions being made is essential. In the literature, the problem of explaining the neural network predictions has been tackled by techniques that extract symbolic rules from the trained networks. These neural network rule extraction techniques attempt to open up the neural network black box and generate symbolic rules with (approximately) the same predictive power as the neural network itself. An advantage of using neural network rule extraction methods is that the neural network considers the contribution of the inputs towards classification as a group, while decision-tree algorithms like C4.5 measure the individual contribution of the inputs one at a time as the tree is grown.

Andrews et al. (1995) propose a classification scheme for neural network rule extraction techniques based on various criteria. In this paper, we will focus mainly on two dimensions when discussing the algorithms: the translucency of the rule extraction algorithm and the expressive power of the extracted rules.

The translucency criterion considers the technique's perception of the neural network. A decompositional approach starts extracting rules at the level of the individual hidden and output units by analysing the activation values, weights, and biases. Decompositional approaches then typically treat the hidden units as threshold units. On the other hand, a pedagogical algorithm considers the trained neural network as a "black box." Instead of looking at the internal structure of the network, these algorithms directly extract rules which relate the inputs and outputs of the network. These techniques typically use the trained network to classify examples and to generate additional "artificial" examples which are then used by a symbolic learning algorithm to infer the rules.

The expressive power of the extracted rules depends on the language used to express the rules. Propositional if-then rules are implications of the form **If** $X = a$ **and** $Y = b$, **then** class $= 1$. An example of a fuzzy classification rule is: **If** X is low **and** Y is medium, **then** class $= 1$, whereby low and medium are fuzzy sets with corresponding membership functions. M-of-N rules are usually expressed as follows: **If** {at least/exactly/at most} M of the N conditions $(C_1, C_2, \ldots, C_N)$ are satisfied, **then** class $= 1$.

In the following subsections we will discuss the Neurorule, Trepan, and Nefclass extraction algorithms. The motivation for choosing these algorithms is that they have different characteristics with respect to the classification scheme suggested by Andrews et al. (1995) and that they thus tackle the extraction problem in a totally different way. To our knowledge, the performance of these algorithms has never been compared for rule or tree extraction using real-life data.

2.2. Neurorule

Neurorule is a decompositional algorithm that extracts propositional rules from trained three-layered feedforward neural networks (Setiono and Liu 1996). It consists of the following steps:

Step 1. Train a neural network to meet the prespecified accuracy requirement.

Step 2. Remove the redundant connections in the network by pruning while maintaining its accuracy.

Step 3. Discretize the hidden unit activation values of the pruned network by clustering.

Step 4. Extract rules that describe the network outputs in terms of the discretized hidden unit activation values.

Step 5. Generate rules that describe the discretized hidden unit activation values in terms of the network inputs.

Step 6. Merge the two sets of rules generated in Steps 4 and 5 to obtain a set of rules that relates the inputs and outputs of the network.

Neurorule assumes the data are discretized and represented as binary inputs using the thermometer encoding for ordinal variables and dummy encoding for nominal variables (Setiono and Liu 1996). Table 1 illustrates the thermometer encoding for the ordinal income variable.

The continuous income attribute is first discretized to the values 1, 2, 3, and 4. This can be done by either a discretization algorithm (e.g., the algorithm of Fayyad and Irani 1993) or according to the recommendation from the domain expert. The four values are then represented by three thermometer inputs I_1, I_2, and I_3. If I_3 is 1, this corresponds to categorical income input ≥ 2, or original income input $>1{,}000$ Euro. This encoding scheme facilitates the generation and interpretation of the propositional if-then rules.

Table 1 The Thermometer Encoding Procedure for Ordinal Variables

Original input	Categorical input	Thermometer inputs I_1	I_2	I_3
Income $\leq$ 1,000 euro	1	0	0	0
Income $>$ 1,000 euro and $\leq$ 2,000 euro	2	0	0	1
Income $>$ 2,000 euro and $\leq$ 3,000 euro	3	0	1	1
Income $>$ 3,000 euro	4	1	1	1

Table 2 **The Dummy Encoding Procedure for Nominal Variables**

	I_1	I_2
Purpose = car	0	0
Purpose = real estate	0	1
Purpose = other	1	0

Neurorule assumes the nominal variables are represented by dummies. For example, when a nominal variable has three values, it is encoded with two dummy variables according to the setup shown in Table 2.

Neurorule typically starts from a one-hidden-layer neural network with hyperbolic tangent hidden neurons and linear output neurons. For a classification problem with C classes, C output neurons are used and the class is assigned to the output neuron with the highest activation value (*winner-take-all learning*). The network is then trained to minimize a regularized cross-entropy error function using the BFGS method, which is a modified quasi-Newton algorithm (Setiono 1995).

Determining the optimal number of hidden neurons is not a trivial task. Neurorule starts from an oversized network and then gradually removes the irrelevant connections. When all connections to a hidden neuron have been pruned, this neuron can be removed from the network. The selection of network connections for pruning is achieved by inspecting the magnitude of their weights (Setiono 1997). A connection with sufficiently small weight can be pruned from the network without affecting the network's classification accuracy.

Once a trained and pruned network has been obtained, the activation values of all hidden neurons are clustered to simplify the rule extraction process. In the case of hyperbolic tangent hidden neurons, the activation values lie in the interval $[-1, 1]$. A simple greedy clustering algorithm then starts by sorting all these hidden activation values in increasing order (Setiono et al. 1998). Adjacent values are then merged into a unique discretized value as long as the class labels of the corresponding observations do not conflict. The merging process hereby first considers the pair of hidden activation values with the shortest distance in between. Another discretization algorithm that can be used is the Chi2 algorithm, which is an improved and automated version of the ChiMerge algorithm and makes use of the χ^2 test statistic to merge the hidden activation values (Liu and Setiono 1995).

In Step 4 of Neurorule, a new data set is composed consisting of the discretized hidden unit activation values and the class labels of the corresponding observations. Duplicate observations are removed and rules are inferred relating the class labels to the clustered hidden unit activation values. This can be done using an automated rule-induction algorithm such as X2R (Liu and Tan 1995) or manually when the pruned network has only a few unique discretized hidden unit activation values. Note that Steps 3 and 4 can be done simultaneously by C4.5(rules) because C4.5(rules) can work with both discretized and continuous data (Quinlan 1993).

In the last two steps of Neurorule, the rules of Step 4 are translated in terms of the original inputs. First, the rules are generated describing the discretized hidden unit activation values in terms of the original inputs. To this end, one might again use an automated rule-induction algorithm (e.g., X2R, C4.5). This rule set is then merged with that of Step 4 by replacing the conditions of the latter with those of the former.

2.3. Trepan

Trepan was introduced in Craven and Shavlik (1996). It is a pedagogical algorithm which extracts decision trees from trained neural networks with arbitrary architecture by using a symbolic learning algorithm (see §2.1). Like in most decision-tree algorithms (Quinlan 1993), Trepan grows a tree by recursive partitioning. At each step, a queue of leaves is further expanded into subtrees until a stopping criterion is met. A crucial difference with decision-tree-induction algorithms is that the latter have only a limited set of training observations available. Hence, these algorithms typically suffer from having fewer and fewer training observations available for deciding upon the splits or leaf node class labels at lower levels of the tree. On the other hand, the primary goal of neural network rule extraction is to mimic the behaviour of the trained neural network. Hence, instead of using the original training observations, Trepan first relabels them according to the classifications made by the network. The relabelled training data set is then used to initiate the tree-growing process. Furthermore, Trepan can also enrich the training data with additional training instances which are then also labelled (classified) by the neural network itself. The network is thus used as an oracle to answer class membership queries about artificially generated data points. This way, it can be assured that each node split or leaf node class decision is based upon at least $S_{\min}$ data points where $S_{\min}$ is a user-defined parameter. In other words, if a node has only m training data points available and $m < S_{\min}$, then $S_{\min} - m$ data points are additionally generated and labelled by the network.

Generating these additional data points is by no means a trivial task. First of all, care should be taken that the generated data instances satisfy all constraints (conditions) that lie from the root of the tree to the node under consideration. Given these constraints, one approach might be to sample the data instances uniformly. However, a better alternative would be to take into account the distribution of the data. This is the approach followed by Trepan. More specifically, at each node of the tree, Trepan estimates the marginal distribution of each input. For a discrete valued input, Trepan simply uses the empirical frequencies of the various values, whereas for a continuous input a kernel density estimation method is used.

Trepan allows splits with *at least M-of-N* type of tests. Note that the test *at least 2 of* $\{C_1, C_2, C_3\}$ *is logically equivalent to* (C_1 *and* C_2) *or* (C_1 *and* C_3) *or* (C_2 *and* C_3). These M-of-N splits are constructed by using a heuristic search procedure. First, the best binary split is selected according to the information-gain criterion (Quinlan 1993). The best binary test then serves as a seed for the M-of-N search process, which uses the following operators:

- M-of-$N+1$: Add a new condition to the set; e.g., 2 of $\{C1, C2\}$ becomes 2 of $\{C1, C2, C3\}$.
- $M+1$-of-$N+1$: Add a new condition to the set and augment the threshold; e.g., 2 of $\{C1, C2, C3\}$ becomes 3 of $\{C1, C2, C3, C4\}$.

The heuristic search procedure uses a beam-search method with a beam width of two, meaning that at each point the best two splits are retained for further examination. Again, the information-gain criterion is used to evaluate the splits. Finally, once an M-of-N test has been constructed, Trepan tries to simplify it and investigates if conditions can be dropped and/or M can be reduced without significantly degrading the information gain.

Trepan uses one local and one global criterion to decide when to stop growing the tree. For the local stopping criterion, Trepan constructs a confidence interval around p_c, which is the proportion of instances belonging to the most common class at the node under consideration. The node becomes a leaf when $\text{prob}(p_c < 1 - \epsilon) < \alpha$, whereby α is the significance level and ϵ specifies how tight the confidence interval around p_c must be. Both values are set to 0.01 by default. The global criterion specifies a maximum on the number of internal nodes of the tree and can be specified in advance by the user. Trees with a small number of internal nodes are more comprehensible than large trees.

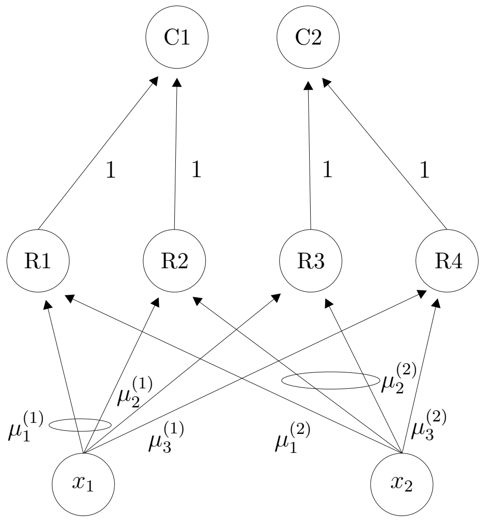
2.4. Nefclass

The category of neural network fuzzy-rule extraction techniques is often referred to as neurofuzzy systems. Basically, these systems encompass methods that use learning algorithms from neural networks to tune the parameters of a fuzzy system. In this section, we will further elaborate on Nefclass, which is a well-known neurofuzzy system (Nauck 2000).

Nefclass has the architecture of a three-layer fuzzy perceptron, whereby the first layer consists of input neurons, the second layer of hidden neurons, and the third layer of output neurons. The difference with a classical multilayer perceptron (cf. §2.1) is that the weights now represent fuzzy sets and that the activation functions are now fuzzy set operators. The hidden-layer neurons represent the fuzzy rules and the output-layer neurons the different classes of the classification problem with 1 output neuron per class. Figure 2 depicts an example of a Nefclass network. The fuzzy rule corresponding to rule unit R1 is expressed as follows:

$$\textbf{If } x_1 \text{ is } \mu_1^{(1)} \text{ and } x_2 \text{ is } \mu_1^{(2)} \textbf{ then } \text{Class} = C1, \qquad (3)$$

Figure 2 Example of Nefclass Network



whereby $\mu_1^{(1)}$ and $\mu_1^{(2)}$ represent the fuzzy sets defined for x_1 and x_2. Nefclass enforces all connections representing the same linguistic label (e.g., x_1 is small) to have the same fuzzy set associated with them. For example, in Figure 2, the fuzzy set $\mu_1^{(1)}$ is shared by the rule units R_1 and R_2, and thus has the same definition in both fuzzy rules.

Nefclass allows the user to model a priori domain knowledge before starting to learn the various parameters, or the classifier can also be created from scratch. In both cases, the user must start by specifying the fuzzy sets and membership function types for all inputs, which can be trapezoidal, triangular, Gaussian, or List.

Nefclass starts by determining the appropriate number of rule units in the hidden layer. Suppose we have a data set D of N data points $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$,with input data $\mathbf{x}_i \in \mathbb{R}^n$ and target vectors $\mathbf{y}_i \in \{0, 1\}^m$ for an m-class classification problem. For each input x_i, q_i fuzzy sets $\mu_1^{(i)}, \ldots, \mu_{q_i}^{(i)}$ are defined. The rule-learning algorithm then proceeds as follows.

Step 1. Select the next pattern $(\mathbf{x}_j, \mathbf{y}_j)$ from D.

Step 2. For each input unit x_i, $i = 1, \ldots, n$, find the membership function $\mu_{l_i}^{(i)}$ such that

$$\mu_{l_i}^{(i)}(x_i) = \max_{l \in \{1, \ldots, q_i\}} \{\mu_l^{(i)}(x_i)\}. \tag{4}$$

Step 3. If there is no rule node R with

$$W(x_1, R) = \mu_{l_1}^{(1)}, \ldots, W(x_n, R) = \mu_{l_n}^{(n)}, \tag{5}$$

with $W(x_i, R)$ the fuzzy weight between input x_i and rule node R, then create such a node and connect it to output class node p if $\mathbf{y}_j(p) = 1$.

Step 4. Go to Step 1 until all patterns in D have been processed.

Obviously, the above procedure will result in a large number of hidden neurons and fuzzy rules. This can be remedied by specifying a maximum number of hidden neurons and keeping only the first k rule units created (*simple rule learning*). Alternatively, one could also keep the best k rules (*best rule learning*) or the best $\lfloor k/m \rfloor$ rules for each class (*best per class rule learning*).

Once the number of hidden units has been determined, the fuzzy sets between the input and hidden layer are tuned to improve the classification accuracy of the network. Hereto, Nefclass employs a fuzzy variant of the well-known backpropagation algorithm to tune the characteristic parameters of the membership functions (Nauck 2000). Nefclass also offers the possibility of pruning the rule base by removing rules and variables based on a simple greedy algorithm. The goal of this pruning is to improve the comprehensibility of the created classifier (see Nauck 2000 for more details).

3. Neural Network Rule Extraction Experiments

3.1. Data Sets and Experimental Setup

The experiments will be conducted on three real-life credit-risk evaluation data sets: German credit, Bene 1, and Bene 2. The Bene 1 and Bene 2 data sets were obtained from two major Benelux financial institutions. The German credit data set is publicly available at the UCI repository.[1] The German credit and Bene 1 data set will be used in due course to illustrate the various results. Their inputs are given in the Appendix. Because all data sets are rather large, each data set is randomly split into two-thirds training set and one-third test set. All inputs are discretized using the discretization algorithm of Fayyad and Irani (1993) with the default options on the training set. This algorithm uses an information entropy

[1] ⟨http://www.ics.uci.edu/~mlearn/mlrepository.html⟩.

Table 3 Data Set Characteristics

	Inputs before discretization	Inputs after discretization	Data set size	Training set size	Test set size	Goods/bads
German credit	20	15	1,000	666	334	70/30
Bene 1	33	24	3,123	2,082	1,041	66.7/33.3
Bene 2	33	29	7,190	4,793	2,397	70/30

minimization heuristic to discretize the range of a continuous attribute into multiple intervals. Table 3 displays the characteristics of all data sets.

We will also include C4.5 and C4.5rules, as well as logistic regression, as a benchmark to compare the results of the rule extraction algorithms. All algorithms will be evaluated by their classification accuracy as measured by the percentage correctly classified (PCC) observations and by their complexity. Because our main purpose is to develop intelligent credit-risk evaluation systems that are both comprehensible and user friendly, it is obvious that simple, concise rule sets and trees are to be preferred. Hence, we will also take into account the complexity of the generated rules or trees as a performance measure. The complexity will be quantified by looking at the number of generated rules or the number of leaf nodes and total number of nodes for the C4.5 and Trepan trees. Note that the total number of nodes of a tree is the sum of the number of internal nodes and the number of leaf nodes.

Because the primary goal of neural network rule extraction is to mimic the decision process of the trained neural network, we will also measure how well the extracted rule set or tree models the behaviour of the network. For this purpose, we will also measure the fidelity of the extraction techniques, which is defined as the percentage of observations that the extraction algorithm classifies in the same way as the neural network.

For the Neurorule analyses, we use two output units with linear activation functions, and the class is assigned to the output neuron with the highest activation value (winner-takes-all). A hyperbolic tangent activation function is used in the hidden layer.

Following Craven and Shavlik (1996), we set the S_{min} parameter for the Trepan analyses to 1,000, meaning that at least 1,000 observations are considered before deciding upon each split or leaf node class label. The maximum tree size is set to 15, which is the size of a complete binary tree of depth four.

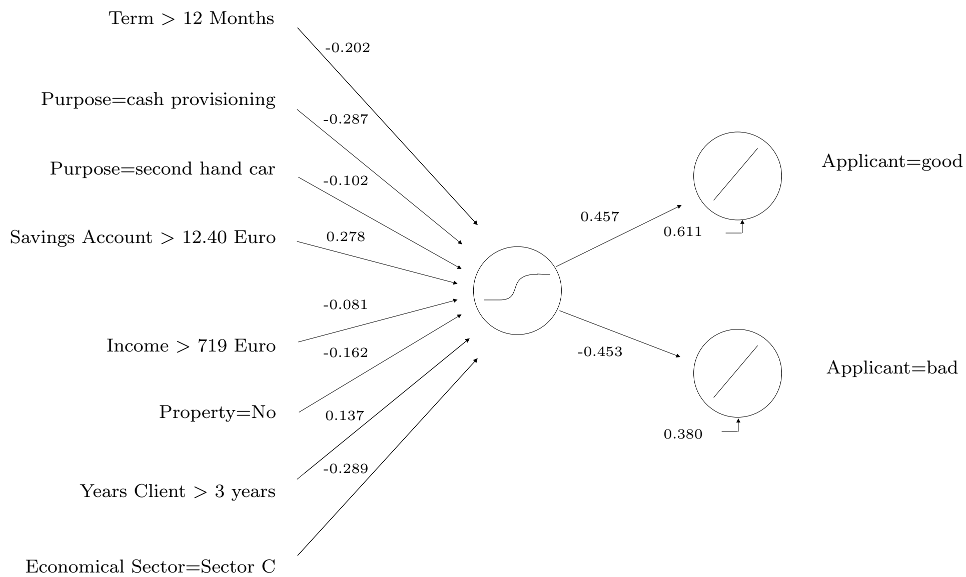
Because Trepan is a pedagogical tree-extraction algorithm, we can apply it to any trained neural network with arbitrary architecture. Hence, we will apply Trepan to the same networks that were trained and pruned by Neurorule. This will allow us to make a fair comparison between a pedagogical and a decompositional neural network rule extraction method.

For Nefclass, we will experiment with triangular, trapezoidal, and bell-shaped membership functions and use 2, 4, or 6 fuzzy sets per variable. We will also use both *best rule learning* and *best per class rule learning* with a maximum of 100 fuzzy rules.

3.2. Neural Network Rule Extraction Results

When representing all discretized inputs using the thermometer and dummy encoding, we ended up with 45 binary inputs for the German credit data set, 45 binary inputs for the Bene 1 data set, and 105 inputs for the Bene 2 data set. We then trained and pruned the neural networks for rule extraction using Neurorule and tree extraction using Trepan. Figure 3 depicts the neural network that was trained and pruned for the Bene 1 data set. Only 1 hidden unit was needed with a hyperbolic tangent transfer function. All inputs are binary; e.g., the first input is 1 if term $>$ 12 months, and 0 otherwise. Note that according to the pruning algorithm, no bias was needed to the hidden neuron for the Bene 1 data set. Of the 45 binary inputs, 37 were pruned, leaving only 8 binary inputs in the neural network. This corresponds to 7 of the original inputs depicted in Table 8 of the Appendix because the nominal purpose input has two corresponding binary inputs in the pruned network (purpose = cash provisioning and purpose = secondhand car).

Figure 3 Neural Network Trained and Pruned for Bene 1



For the Bene 2 data set, the pruning procedure removed 97 of the 105 binary inputs and the remaining 8 corresponded to 7 of the original inputs. The binarized German credit data set consists of 45 inputs of which 13 are retained, corresponding to 6 of the original inputs of Table 7 in the Appendix.

Table 4 presents the performance and complexity of C4.5, C4.5rules, the pruned neural network (NN), Neurorule, Trepan, and Nefclass on the discretized data sets. Table 5 presents the fidelity rates of Neurorule and Trepan on the training set (Fid_{train}) and the test set (Fid_{test}).

Table 4 Accuracy and Complexity of Decision Trees, Neural Networks, and Extraction Techniques

Data set	Method	PCC_{train}	PCC_{test}	Complexity
German credit	C4.5	80.63	71.56	38 leaves, 54 nodes
	C4.5rules	81.38	74.25	17 propositional rules
	Pruned NN	75.53	77.84	6 inputs
	Neurorule	75.83	77.25	4 propositional rules
	Trepan	75.37	73.95	11 leaves, 21 nodes
	Nefclass	73.57	73.65	14 fuzzy rules
Bene 1	C4.5	77.76	70.03	77 leaves, 114 nodes
	C4.5rules	76.70	70.12	17 propositional rules
	Pruned NN	73.05	71.85	7 inputs
	Neurorule	73.05	71.85	6 propositional rules
	Trepan	73.05	71.85	11 leaves, 21 nodes
	Nefclass	68.97	67.24	8 fuzzy rules
Bene 2	C4.5	82.80	73.09	438 leaves, 578 nodes
	C4.5rules	77.76	73.51	27 propositional rules
	Pruned NN	74.15	74.09	7 inputs
	Neurorule	74.27	74.13	7 propositional rules
	Trepan	74.15	74.01	9 leaves, 17 nodes
	Nefclass	70.06	69.80	4 fuzzy rules

Table 5 Fidelity Rates of Extraction Techniques

Data set	Method	Fid_{train}	Fid_{test}
German credit	Neurorule	99.70	98.80
	Trepan	94.07	93.11
Bene 1	Neurorule	100	100
	Trepan	100	100
Bene 2	Neurorule	99.71	99.79
	Trepan	99.91	99.83

For the German credit data set, Neurorule yielded a higher test set classification accuracy than C4.5rules and extracted only 4 propositional rules, which is very compact when compared to the 17 propositional rules inferred by C4.5rules. The Trepan tree obtained a better classification accuracy than C4.5 with fewer leaves and nodes. Also, Nefclass obtained a satisfactory classification accuracy, but it needed 14 fuzzy rules. The test set fidelity of Neurorule is 98.80%, whereas Trepan obtained 93.11% test set fidelity, which indicates that Neurorule mimics the decision process of the network better than Trepan.

For the Bene 1 data set, Neurorule performed significantly better than C4.5rules according to McNemar's test at the 5% significance level. Besides the gain in performance, Neurorule also uses only 6 propositional rules, whereas C4.5rules uses 17 propositional rules. The rule set inferred by Neurorule obtained 100% test set fidelity with respect to the pruned neural network from which it was derived. Trepan gave better performance than C4.5. Again, the tree was a lot more compact, consisting of only 11 leaves and 21 nodes. The Trepan tree also achieved 100% test set fidelity with respect to the pruned neural network. The high fidelity rates of Neurorule and Trepan indicate that both were able to accurately approximate the decision process of the trained and pruned neural network. Nefclass yielded a maximum test set accuracy of 67.24% with 8 fuzzy rules, which is rather poor compared to the other extraction algorithms.

For the Bene 2 data set, the performance difference between Neurorule and C4.5rules is not statistically significant at the 5% level using McNemar's test. However, the rule set extracted by Neurorule consists of only 7 propositional rules, which is a lot more compact than the 27 propositional rules induced by C4.5rules. Note that the rules inferred by Neurorule performed slightly better than the neural network itself, resulting in a test set fidelity of 99.79%. The tree inferred by Trepan has a very good performance and was again compact when compared to the C4.5 tree. Trepan achieved 99.83% test set fidelity. Again, Nefclass was not able to infer a compact and powerful fuzzy rule set.

We also contrasted the results of Table 4 with the performance of a logistic regression classifier which has been widely used in the credit industry. The logistic regression classifier achieved a test set classification accuracy of 70.66%, 70.51%, and 73.09% for the German credit, Bene 1, and Bene 2 data sets, respectively. For all these data sets, Neurorule obtained a significantly better performance than the logistic regression classifier at the 5% level. Although the absolute difference might seem small, it has to be noted that small absolute differences in classification performance, even a fraction of a percent, may, in a credit scoring context, translate into substantial future savings as the following quote of Henley and Hand (1997, p. 318) suggests: "Although the differences are small, they may be large enough to have commercial implications."

Figures 4 and 5 represent the rules extracted by Neurorule for the German credit and Bene 1 data sets, whereas Figures 6 and 7 represent the extracted Trepan trees for both data sets. Notice that both Trepan trees extensively use the M-of-N type of splits. Although these are powerful splits, their value in terms of comprehensibility is rather limited. It is very difficult to comprehend a Trepan tree and get a thorough insight into how the inputs affect the classification decision when there are many M-of-N splits present. On the other hand, when looking at the rules extracted by Neurorule, it becomes clear that these propositional rules are easy to interpret and understand.

While propositional rules are an intuitive and well-known formalism to represent knowledge, they are not necessarily the most suitable representation in terms of structure and efficiency of use in everyday business practice and decision making. Recent

Figure 4 Rules Extracted by Neurorule for German Credit

If (Checking account $\neq$ 4) **And** (Checking account $\neq$ 3) **And** (Term = 1)
And (Credit history $\neq$ 4) **And** (Credit history $\neq$ 3)
And (Credit history $\neq$ 2) **And** (Purpose $\neq$ 8)
Then Applicant = bad

If (Checking account $\neq$ 4) **And** (Checking account $\neq$ 3)
And (Credit history $\neq$ 4) **And** (Credit history $\neq$ 3)
And (Credit history $\neq$ 2) **And** (Term = 2)
Then Applicant = bad

If (Checking account $\neq$ 4) **And** (Checking account $\neq$ 3)
And (Credit history $\neq$ 4) **And** (Purpose $\neq$ 5) **And** (Purpose $\neq$ 1)
And (Savings account $\neq$ 5) **And** (Savings account $\neq$ 4)
And (Other parties $\neq$ 3) **And** (Term = 2)
Then Applicant = bad

Default class: Applicant = good

research in knowledge representation suggests that graphical representation formalisms can be more readily interpreted and consulted by humans than a set of symbolic propositional if-then rules (Santos-Gomez and Darnel 1992). In the following section, we discuss how the extracted sets of rules may be transformed into decision tables which facilitate the efficient classification of applicants by the credit-risk manager.

4. Visualizing the Extracted Rule Sets Using Decision Tables

Decision tables (DTs) provide an alternative way of representing data mining knowledge extracted by, e.g., neural network rule extraction in a user-friendly way (Wets et al. 1997). DTs are a tabular representation used to describe and analyse decision situations (e.g., credit-risk evaluation),where the state of

Figure 5 Rules Extracted by Neurorule for Bene 1

If Term > 12 months **And** Purpose = cash provisioning **And** Savings account $\leq$ 12.40 Euro **And** Years client $\leq$ 3 **Then** Applicant = bad

If Term > 12 months **And** Purpose = cash provisioning **And** Owns property = No **And** Savings account $\leq$ 12.40 Euro **Then** Applicant = bad

If Purpose = cash provisioning **And** Income > 719 Euro **And** Owns property = No **And** Savings account $\leq$ 12.40 Euro **And** Years client $\leq$ 3
Then Applicant = bad

If Purpose = second hand car **And** Income > 719 Euro **And** Owns property = No **And** Savings account $\leq$ 12.40 Euro **And** Years client $\leq$ 3
Then Applicant = bad

If Savings account $\leq$ 12.40 Euro **And** Economical sector = Sector C
Then Applicant = bad

Default class: Applicant = good

Figure 6 Tree Extracted by Trepan for German Credit

```
3 of {Credit history ≠ 4, Term = 2, Checking account ≠ 4}:
| 2 of {Credit history = 2, Savings account = 5, Purpose = 1}: Applicant = good
| Not 2 of {Credit history = 2, Savings account = 5, Purpose = 1}:
| | Checking account ≠ 3:
| | | Other parties ≠ 3:
| | | | 1 of {Credit history = 3, Savings account = 4}: Applicant = good
| | | | Not 1 of {Credit history = 3, Savings account = 4}:
| | | | | Purpose ≠ 5:
| | | | | | Credit history ≠ 2:
| | | | | | | Savings account ≠ 3:
| | | | | | | | Savings account ≠ 5:
| | | | | | | | | Purpose ≠ 1: Applicant = bad
| | | | | | | | | Purpose = 1: Applicant = good
| | | | | | | | Savings account = 5: Applicant = good
| | | | | | | Savings account = 3: Applicant = good
| | | | | | Credit history = 2: Applicant = bad
| | | | | Purpose = 5: Applicant = good
| | | Other parties = 3: Applicant = good
| | Checking account = 3: Applicant = good
Not 3 of {Credit history ≠ 4, Term = 2, Checking account ≠ 4}: Applicant = good
```

a number of conditions jointly determines the execution of a set of actions (Vanthienen and Wets 1994). In our neural network rule extraction context, the conditions correspond to the antecedents of the rules, whereas the actions correspond to the outcome classes (applicant = good or bad). A DT consists of four quadrants, separated by double lines, both horizontally and vertically (see Figure 8). The horizontal line

Figure 7 Tree Extracted by Trepan for Bene 1

```
2 of {purpose ≠ car, Savings account > 12.40 Euro, purpose ≠ cash}:
| Economical sector ≠ C: Applicant = good
| Economical sector = C:
| | Savings account ≤ 12.40 Euro: Applicant = bad
| | Savings account > 12.40 Euro: Applicant = good
Not 2 of {purpose ≠ car, Savings account > 12.40 Euro, purpose ≠ cash}:
| 3 of {Economical sector ≠ C, Term ≤ 12, Property = Yes, Years client > 3}:
| | Applicant = good
| Not 3 of {Economical sector ≠ C, Term ≤ 12, Property = Yes, Years client > 3}:
| | purpose ≠ cash:
| | | Income ≤ 719 Euro: Applicant = good
| | | Income > 719 Euro:
| | | | Property = Yes: Applicant = good
| | | | Property = No:
| | | | | Years client ≤ 3: Applicant = bad
| | | | | Years client > 3: Applicant = good
| | purpose = cash:
| | | Income ≤ 719 Euro:
| | | | Term ≤ 12 Months: Applicant = good
| | | | Term > 12 Months: Applicant = bad
| | | Income > 719 Euro: Applicant = bad
```

Figure 8 DT Quadrants

condition subjects	condition entries
action subjects	action entries

divides the table into a condition part (above) and an action part (below). The vertical line separates subjects (left) from entries (right).

The condition subjects are the criteria that are relevant to the decision-making process. They represent the attributes of the rule antecedents about which information is needed to classify a given applicant as good or bad. The action subjects describe the possible outcomes of the decision-making process (i.e., the classes of the classification problem). Each condition entry describes a relevant subset of values (called a state) for a given condition subject (attribute), or contains a dash symbol ("-") if its value is irrelevant within the context of that column. Subsequently, every action entry holds a value assigned to the corresponding action subject (class). True, false, and unknown action values are typically abbreviated by "×", "-", and ".," respectively. Every column in the entry part of the DT thus comprises a classification rule, indicating what action(s) apply to a certain combination of condition states. If each column only contains simple states (no contracted or irrelevant entries), the table is called an expanded DT, whereas otherwise the table is called a contracted DT. Table contraction can be achieved by combining columns that lead to the same action configuration. The number of columns in the contracted table can then be further minimized by changing the order of the conditions. It is obvious that a DT with a minimal number of columns is to be preferred because it provides a more parsimonious and comprehensible representation of the extracted knowledge than an expanded DT. This is illustrated in Figure 9.

Several kinds of DTs have been proposed. We will require that the condition entry part of a DT satisfies the following two criteria:

- Completeness: all possible combinations of condition values are included.
- Exclusivity: no combination is covered by more than one column.

As such, we deliberately restrict ourselves to single-hit tables, wherein columns have to be mutually exclusive, because of their advantages with respect to verification and validation (Vanthienen et al. 1998). It is this type of DT that can be easily checked for potential anomalies, such as inconsistencies (a particular case being assigned to more than one class) or incompleteness (no class assigned). The DT formalism thus allows for easy verification of the knowledge extracted by, e.g., a neural network rule extraction algorithm. Additionally, for ease of legibility, the columns are arranged in lexicographical order, in which entries at lower rows alternate first. As a result, a tree structure emerges in the condition entry part of the DT, which lends itself very well to a top-down evaluation procedure: Starting at the first row, and then working one's way down the table by choosing from the relevant condition states, one safely arrives at the prescribed action (class) for a given case. This condition-oriented inspection approach often proves more intuitive, faster, and less prone to human error than evaluating a set of rules one by one. Once the DT has been approved by the expert, it can, in a final stage, be incorporated into a deployable expert system (Vanthienen and Wets 1994).

Figure 9 Minimising the Number of Columns of a DT (Vanthienen and Wets 1994)

1. C1	Y				N			
2. C2	Y		N		Y		N	
3. C3	Y	N	Y	N	Y	N	Y	N
1. A1	-	×	×	×	-	×	-	×
2. A2	×	-	-	-	×	-	×	-

(a) Expanded DT

1. C1	Y			N	
2. C2	Y		N	-	
3. C3	Y	N	-	Y	N
1. A1	-	×	×	-	×
2. A2	×	-	-	×	-

(b) Contracted DT

1. C3	Y			N
2. C1	Y		N	-
3. C2	Y	N	-	-
1. A1	-	×	-	×
2. A2	×	-	×	-

(c) Minimised DT

Figure 10 Decision Table for the Rules Extracted by Neurorule on German Credit

1. Checking account	1 or 2										3 or 4
2. Credit History	0 or 1				2 or 3					4	-
3. Term	1			2	1	2				-	-
4. Purpose	1 or 5	8	other	-	-	1 or 5	8 or other			-	-
5. Savings account	-	-	-	-	-	-	1 or 2 or 3		4 or 5	-	-
6. Other parties	-	-	-	-	-	-	1 or 2	3	-	-	-
1. Applicant=good	-	×	-	-	×	×	-	×	×	×	×
2. Applicant=bad	×	-	×	×	-	-	×	-	-	-	-
	1	2	3	4	5	6	7	8	9	10	11

Figure 11 Decision Table for the Rules Extracted by Neurorule on Bene 1

1. Savings Account	≤12.40 Euro													> 12.40 Euro
2. Economical sector	Sector C	other												-
3. Purpose	-	cash provisioning							second-hand car				other	-
4. Term	-	≤ 12 months				> 12 months			-					
5. Years Client	-	≤ 3			>3	≤ 3	>3		≤ 3			> 3		
6. Property	-	Yes	No		-	-	Yes	No	Yes	No		-	-	-
7. Income	-	-	≤ 719 Euro	> 719 Euro	-	-	-	-	-	≤ 719 Euro	> 719 Euro	-	-	-
1. Applicant=good	-	×	×	-	×	-	×	-	×	×	-	×	×	×
2. Applicant=bad	×	-	-	×	-	×	-	×	-	-	×	-	-	-
	1	2	3	4	5	6	7	8	9	10	11	12	13	14

We will use the Prologa[2] software to construct the DTs for the rules extracted in §3.2. Prologa is an interactive design tool for computer-supported construction and manipulation of DTs (Vanthienen and Dries 1994).

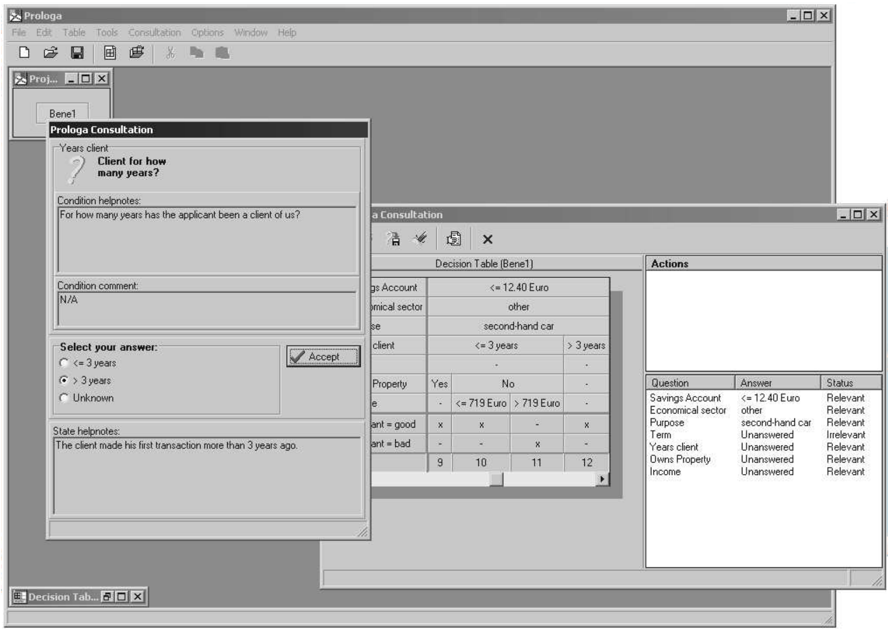
Figures 10 and 11 depict the contracted DTs generated from the rules extracted by Neurorule for the discretized German credit and Bene 1 data set. It is important to note that transforming a set of propositional rules into a DT does not cause any loss of predictive accuracy; i.e., the DTs depicted in Figures 10 and 11 achieve exactly the same classification accuracy as the rules of Figures 4 and 5. For the German credit data set, the fully expanded table contained 6,600 columns, which is the product of the number of distinct attribute values ($= 4 \times 5 \times 2 \times 11 \times 5 \times 3$). This could be reduced to 11 columns by using table contraction and table minimization. For the Bene 1 data set, the fully expanded table contained 192 columns and the contracted and minimized table 14 columns. Both contracted DTs provide a parsimonious representation of the extracted knowledge, consisting of only a small number of columns, which allows for easy consultation. Table 6 presents the properties of the DTs built for the rules extracted by Neurorule and the Trepan trees on all three discretized credit-risk data sets. Note that we converted the Trepan trees to an equivalent set of rules to build the DTs. Because Nefclass gave rather bad performance on all data sets, we did not include DTs for the extracted fuzzy rules.

[2] ⟨http://www.econ.kuleuven.ac.be/tew/academic/infosys/research/Prologa.htm⟩

Table 6 The Number of Columns in the Expanded and Reduced DTs for the Three Data Sets for the Rules and Trees Extracted by Neurorule and Trepan

Data set	Extraction method	Number of columns in expanded DT	Number of columns in reduced DT
German credit	Neurorule	6,600	11
	Trepan	6,600	9
Bene 1	Neurorule	192	14
	Trepan	192	30
Bene 2	Neurorule	192	26
	Trepan	192	49

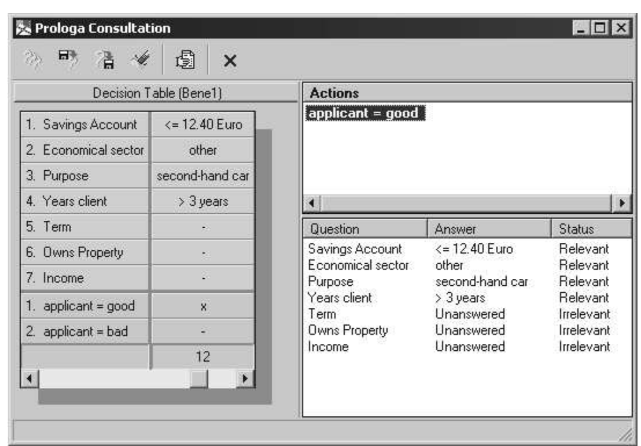
Figure 12 Example Consultation Session in PROLOGA



In all cases, the contracted tables were satisfactorily concise and did not contain any anomalies, thus demonstrating the completeness and the consistency of the extracted rules. For the Bene 1 and Bene 2 data sets, the DTs built for the rules extracted by Neurorule were more compact than those for the Trepan trees. We also constructed the DTs for the rules induced by C4.5rules and found that these tables were huge and impossible to handle because of the large number of generated rules and unpruned inputs.

DTs allow for an easy and user-friendly consultation in everyday business practice. Figure 12 presents an example of a consultation session in PROLOGA. Suppose we try to work ourselves towards column 12 of the DT for Bene 1 depicted in Figure 11. We start by providing the system with the following inputs: Savings account $\leq$12.40 Euro, economical sector = other, and purpose = secondhand car. At this point, the term input becomes irrelevant (indicated by "-"), and hence the system prompts for the next relevant input, which is the number of years the applicant has been a client of the bank. We then indicate that the applicant has been a client for more than 3 years. The other remaining inputs (owns property and income) then become irrelevant, which allows the system to draw a conclusion: applicant = good. This is illustrated in Figure 13. For this particular applicant, the system needed only 4 of the 7 inputs to make a classification decision. This example clearly illustrates how the use of DTs allows one to reach a decision promptly by neglecting the irrelevant inputs during the decision process. It is precizely this property that makes DTs interest-

Figure 13 Classifying an Applicant in PROLOGA



ing management tools for decision support in credit scoring.

5. Conclusion

Recently, neural networks have attracted a lot of interest in the context of developing credit-risk evaluation models because of their universal approximation property. However, most of this work focuses primarily on developing networks with high predictive accuracy without trying to explain how the classifications are being made. In application domains such as credit-risk evaluation, having a set of concise and comprehensible rules is essential for the credit-risk manager. In this paper, we have evaluated and contrasted three neural network rule extraction techniques—Neurorule, Trepan, and Nefclass, for credit-risk evaluation. The experiments were conducted on three real-life financial credit-risk evaluation data sets. It was shown that, in general, both Neurorule and Trepan yield a very good classification accuracy when compared to the popular C4.5 algorithm and the logistic regression classifier. Furthermore, it was concluded that Neurorule and Trepan were able to extract very compact rule sets and trees for all data sets. The propositional rules inferred by Neurorule were especially concise and very comprehensible. We also described how DTs could be used to represent the extracted rules. DTs represent the rules in an intuitive graphical format that can be easily verified by a human expert. Furthermore, they allow for easy and user-friendly consultation in everyday business practice. We demonstrated that the DTs for the rules and trees extracted by Neurorule and Trepan are compact and powerful. We conclude by saying that neural network rule extraction and DTs are effective and powerful management tools which allow us to build advanced and user-friendly decision-support systems for credit-risk evaluation. Furthermore, it would be interesting to apply the suggested approach to other interesting management science problems: e.g., churn prediction, customer retention, and bankruptcy prediction.

Appendix

Table 7 Attributes for the German Credit Data Set

Nr	Name	Type	Explanation
1	Checking account	nominal	1: <0 DM; 2: ≥ 0 and <200 DM; 3: ≥ 200 DM/salary assignments for at least one year; 4: no checking account
2	Term	continuous	
3	Credit history	nominal	0: no credits taken/all credits paid back duly; 1: all credits at this bank paid back duly; 2: existing credits paid back duly till now; 3: delay in paying off in the past; 4: critical account/other credits (not at this bank)
4	Purpose	nominal	0: car (new); 1: car (old); 2: furniture/equipment; 3: radio/television; 4: domestic appliances; 5: repairs; 6: education; 7 vacation; 8 retraining; 9: business; 10: other
5	Credit amount	continuous	
6	Savings account	nominal	1: <100 DM; 2: ≥ 100 DM and <500 DM; 3: ≥ 500 and <1000 DM; 4: ≥ 1000 DM; 5: unknown/no account
7	Present employment since	nominal	1: unemployed; 2: <1 year; 3: ≥ 1 year and <4 years; 4: ≥ 4 and <7 years; 5: ≥ 7 years
8	Installment rate	continuous	
9	Personal status and sex	nominal	1: male, divorced/separated; 2: female, divorced/separated/ married; 3: male, single; 4: male, married/widowed; 5: female, single
10	Other parties	nominal	1: none; 2: co-applicant; 3: guarantor
11	Present residence since	continuous	
12	Property	nominal	1: real estate; 2: if not 1: building society savings agreement/life insurance; 3: if not 1/2: car or other; 4: unknown/no property
13	Age	continuous	
14	Other installment plans	nominal	1: bank; 2: stores; 3: none
15	Housing	nominal	1: rent; 2: own; 3: for free
16	Number of existing credits at this bank	continuous	

Table 7 Continued

Nr	Name	Type	Explanation
17	Job	nominal	1: unemployed/unskilled-nonresident; 2: unskilled-resident; 3: skilled employee/official; 4: management/self-employed/ highly qualified employee/officer
18	Number of dependents	continuous	
19	Telephone	nominal	1: none; 2: yes, registered under the customer name
20	Foreign worker	nominal	1: yes; 2: no

Table 8 Attributes for the Bene 1 Data Set

Nr	Name	Type
1	Identification number	continuous
2	Amount of loan	continuous
3	Amount on purchase invoice	continuous
4	Percentage of financial burden	continuous
5	Term	continuous
6	Personal loan	nominal
7	Purpose	nominal
8	Private or professional loan	nominal
9	Monthly payment	continuous
10	Savings account	continuous
11	Other loan expenses	continuous
12	Income	continuous
13	Profession	nominal
14	Number of years employed	continuous
15	Number of years in Belgium	continuous
16	Age	continuous
17	Applicant type	nominal
18	Nationality	nominal
19	Marital status	nominal
20	Number of years since last house move	continuous
21	Code of regular saver	nominal
22	Property	nominal
23	Existing credit info	nominal
24	Number of years client	continuous
25	Number of years since last loan	continuous
26	Number of checking accounts	continuous
27	Number of term accounts	continuous
28	Number of mortgages	continuous
29	Number of dependents	continuous
30	Pawn	nominal
31	Economical sector	nominal
32	Employment status	nominal
33	Title/salutation	nominal

References

Andrews, R., J. Diederich, A. B. Tickle. 1995. A survey and critique of techniques for extracting rules from trained neural networks. *Knowledge Based Systems* **8**(6) 373–389.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K.

Capon, N. 1982. Credit scoring systems: A critical analysis. *J. Marketing* **46** 82–91.

Craven, M. W., J. W. Shavlik. 1996. Extracting tree-structured representations of trained networks. D. Touretzky, M. Mozer, M. Hasselmo, ed. *Advances in Neural Information Processing Systems (NIPS)*, Vol. 8. MIT Press, Cambridge, MA, 24–30.

Crook, J. N. 1999. Who is discouraged from applying for credit? *Econom. Lett.* **65** 165–172.

Davis, R. H., D. B. Edelman, A. J. Gammerman. 1992. Machine learning algorithms for credit-card applications. *IMA J. Math. Appl. Bus. Indust.* **4** 43–51.

Desai, V. S., J. N. Crook, G. A. Overstreet Jr. 1996. A comparison of neural networks and linear scoring models in the credit union environment. *Eur. J. Oper. Res.* **95**(1) 24–37.

Fayyad, U. M., K. B. Irani. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. *Proc. Thirteenth Internat. Joint Conf. Artificial Intelligence (IJCAI)*. Chambery, Morgan Kaufmann, France, 1022–1029.

Henley, W. E., D. J. Hand. 1997. Construction of a *k*-nearest neighbour credit-scoring system. *IMA J. Math. Appl. Bus. Indust.* **8** 305–321.

Liu, H., R. Setiono. 1995. Chi2: Feature selection and discretization of numeric attributes. *Proc. Seventh IEEE Internat. Conf. Tools Artificial Intelligence (ICTAI)*. IEEE Computer Society Press, Los Alamitos, CA, 388–391.

——, S. T. Tan. 1995. X2R: A fast rule generator. *Proc. IEEE Internat. Conf. Systems, Man Cybernetics*. IEEE Press, Piscataway, NJ, 631–635.

Nauck, D. 2000. Data analysis with neuro-fuzzy methods. Habilitation thesis, University of Magdeburg, Germany.

Quinlan, J. R. 1993. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, Chambery, France.

Santos-Gomez, L., M. J. Darnel. 1992. Empirical evaluation of decision tables for constructing and comprehending expert system rules. *Knowledge Acquisition* **4** 427–444.

Setiono, R. 1995. A neural network construction algorithm which maximizes the likelihood function. *Connection Sci.* **7**(2) 147–166.

——. 1997. A penalty function approach for pruning feedforward neural networks. *Neural Comput.* **9**(1) 185–204.

——, H. Liu. 1996. Symbolic representation of neural networks. *IEEE Comput.* **29**(3) 71–77.

——, J. Y. L. Thong, C. Yap. 1998. Symbolic rule extraction from neural networks: An application to identifying organizations adopting IT. *Inform. Management* **34**(2) 91–101.

Steenackers, A., M. J. Goovaerts. 1989. A credit scoring model for personal loans. *Insurance: Math. Econom.* **8** 31–34.

Vanthienen, J., E. Dries. 1994. Illustration of a decision table tool for specifying and implementing knowledge based systems. *Internat. J. Artificial Intelligence Tools* **3**(2) 267–288.

——, G. Wets. 1994. From decision tables to expert system shells. *Data Knowledge Engrg*. **13**(3) 265–282.

——, C. Mues, A. Aerts. 1998. An illustration of verification and validation in the modelling phase of KBS development. *Data Knowledge Engrg*. **27** 337–352.

West, D. 2000. Neural network credit scoring models. *Comput. Oper. Res.* **27** 1131–1152.

Wets, G., J. Vanthienen, S. Piramuthu. 1997. Extending a tabular knowledge based framework with feature selection. *Expert Systems Appl.* **13** 109–119.

Accepted by Christopher Tang; received February 6, 2002. This paper was with the authors 7 weeks for 1 revision.