Extracting Comprehensible Rules from Neural Networks via Genetic Algorithms.

Raul T. Santos
CEFET-PR/CPGEI,
Av. Sete de Setembro, 3165,
Curitiba-PR, Brazil,
raul@dainf.cefetpr.br

Júlio C. Nievola
PUC-PR/PPGIA,
Av. Imaculada Conceição, 1155,
Curitiba-PR, Brazil,
nievola@ppgia.pucpr.br

Alex A. Freitas
PUC-PR/PPGIA,
Av. Imaculada Conceição, 1155,
Curitiba-PR, Brazil,
alex@ppgia.pucpr.br
http://www.ppgia.pucpr.br/~alex

Abstract- A common problem in KDD (Knowledge Discovery in Databases) is the presence of noise in the data being mined. Neural networks are robust and have a good tolerance to noise, which makes them suitable for mining very noisy data. However, they have the well-known disadvantage of not discovering any high-level rule that can be used as a support for human decision making. In this work we present a method for extracting accurate, comprehensible rules from neural networks. The proposed method uses a genetic algorithm to find a good neural network topology. This topology is then passed to a rule extraction algorithm, and the quality of the extracted rules is then fed back to the genetic algorithm. The proposed system is evaluated on three public-domain data sets and the results show that the approach is valid.

1 Introduction

With the improvement of data storage technology, there has been a growing interest in extracting knowledge from data. Ideally, the discovered knowledge should be both accurate and comprehensible for the user [1]. One of the difficulties for the extraction of accurate knowledge is that the data being mined can be very noisy. In these cases neural networks are a viable solution, due to their relatively good tolerance to noisy and generalization ability.

On the other hand, it is well-known that neural networks usually represent their "knowledge" in the form of numeric weights and interconnections, which is not comprehensible for the user. This is a serious problem for the user, since (s)he cannot get any insight from the network's output. Hence, the user would have to blindly trust the answer given by the network, which is clearly undesirable in several application domains - e.g. in medical diagnosis of fatal diseases, where lives are at stake. In addition, if the user cannot understand nor validate the discovered knowledge, (s)he can decide to ignore it, which can also lead to most unfortunate decisions. A good example is this quote from [1]: "...the Three-Mile Island case, where all automatic devices correctly recommended a shutdown, but this recommendation was not acted upon by the human operators who did not believe the recommendation was well founded. A similar story applies to the Chernobyl disaster."

Hence, a knowledge discovery algorithm should be explicitly designed to discover comprehensible knowledge. Going further, some knowledge discovery systems are explicitly designed to discover comprehensible knowledge even at some expense of classification accuracy – see e.g. [2], [3], [4].

Therefore, there is a strong motivation to extract high-level, comprehensible rules from trained neural networks. The extracted rules usually are IF-THEN rules, where the IF part of the rule specifies a conjunction of conditions on predicting attribute values and the THEN part specifies a predicted value for the goal (or class) attribute. This work addresses the well-known classification task, widely investigated in the KDD literature.

A number of algorithms for extracting rules from neural networks have been proposed in the literature - see e.g. [5], [6], [7], [8], [9], [10], [11], [12], [13]. In general these algorithms assume that the trained neural network has undergone some postprocessing specifically designed for facilitating the task of rule extraction. For instance, typically the trained network has to be (sometimes drastically) pruned, to reduce the number of interconnections to be considered by the rule extraction algorithm. Only after a careful postprocessing is performed the network is given to the rule extraction algorithm. Then this algorithm is run *once* and its result is the rule set reported to the user.

Intuitively the above procedure can be improved by using the quality of the extracted rules as a feedback to an algorithm that searches for a good network topology. The basic idea is to use a powerful search method where a candidate solution is a neural network topology. This topology is then passed to a rule extraction algorithm. The quality of that topology is evaluated with respect to both the predictive accuracy and the comprehensibility of the extracted rules. The computed quality is then fed back to the search algorithm, which takes this quality into account to generate new candidate network topologies.

This work follows this approach, and in particular it proposes the use of a genetic algorithm to search for network topologies that lead to high-quality (both accurate and comprehensible) extracted rules. Genetic algorithms are a robust search method [14] and are suitable for difficult problems - e.g. problems with very large search spaces and strong nonlinearity, such as the problem of finding good network topologies [15].

Although there has been many projects on using genetic algorithms for evolving neural network topologies [15], [16], [17], these projects do not address the issue of extracting comprehensible rules from the evolved network topologies. To the best of our knowledge, this work is the first to combine the idea of using a genetic algorithm to evolve a network topology with the idea of extracting rules from a neural network.

In the following sections, we present initially a discussion of related work. Next we present an overview of Data Mining, Neural Networks and Genetic Algorithms, and then the proposed method for rule extraction from neural networks via genetic algorithms. After that we show some computational results.

2 Related Work

Several projects have used genetic algorithms to optimize the topology of a neural network. For instance, [16] proposed a high-level individual encoding that ensures that only correct network topologies are generated. The high-level information used in his encoding includes e.g., for each layer, a radius parameter (specifying the connection radius for each unit of the layer) and information to distinguish between receptive connections (which connect the layer with one of the preceding layers) and projective connections (which connect the layer with the next layer).

[15] proposed a genetic algorithm that optimizes both the topology and the interconnection weights of the neural network. Their individual encoding represents a weighted graph, and different individuals of the population can represent networks with different topologies (with different number of nodes and/or interconnections). The crossover operator selects several weights in each parent and then produces two children by swapping these weights between the two parents. There are several kinds of mutation and other genetic operators, such as deleting some node(s) and its(their) corresponding interconnection(s), inserting some node(s) and its(their) corresponding interconnection(s), and randomly selecting a node or interconnection and its weight.

The use of evolutionary algorithms to optimize neural network topologies is not restricted to the classification task. For instance, [17] also used a genetic algorithm to optimize the topology of a neural network, but in their work the network corresponds to a domain knowledge theory (a set of previously-known rules), rather than being a network trained from the data. The aim of this work is theory revision, rather than classification as in our work.

Most of the above work has focused on using evolutionary algorithms with the explicit goal of finding a good topology of the neural network, where the goodness of a topology is related to the predictive accuracy of the network trained with that topology. In contrast, in our work the goodness of the neural network topology found by the evolutionary algorithm is related to the predictive accuracy and the comprehensibility of the rule set extracted from the neural network trained with that topology.

We now turn to projects whose goals is to extract rules from neural networks. It should be noted that most of these projects do not use an evolutionary algorithm to define the neural network topology.

There are two basic ways of representing IF-THEN rules: by using fuzzy rules and by using boolean rules. Neuro-fuzzy systems follow the former approach in three steps: (a) by using adequate mechanisms, knowledge expressed in the form of fuzzy rules is introduced into the ANN; (b) the ANN is trained with the goal of finding an adequate set of membership functions; (c) knowledge in the form of modified membership functions is extracted from the ANN. A representative project following this approach is Hayashi's FNES (Fuzzy Neural Expert System) [18], [19], which has the ability of explaining how a given solution was obtained. Other examples of this approach are the projects by Matsuoka et al. [20], Berenji [21] and Horikawa et al. [22].

The systems based on boolean rules can be grouped into three kinds of techniques: decompositional, pedagogical and ecletic techniques [8]. This latter is a kind of hybrid technique containing elements of the two former ones.

The decompositional technique focus on the extraction of rules at the level of individual units, in the intermediate or output layers. The output of each unit of the trained ANN must be mapped into boolean values, i.e. each unit represents a boolean rule, which leads to a transparent vision of the ANN. One of the first projects following this approach was proposed by Fu [23], who developed the KT algorithm. Towell & Shavlik [24] developed the SUBSET algorithm, based on the decompositional technique, which was very computationally expensive and produced rules with a large number of conditions in their antecedents. In order to improve their method, they developed a new algorithm called MofN [9]. Two other systems using the decompositional technique are RULENET [10], which has the limitation of being oriented for a specific application domain, and RULEX [11], which works with a CEBP (Constrained Error Back Propagation) network.

The pedagogical technique views the ANN as an opaque structure, used to generate examples for learning algorithms. In this kind of technique one searches for a transfer function, i.e. a relationship, expressed via IF-THEN rules, between the inputs and the outputs of the ANN. This technique does not use intermediate elements between the input and the output of the ANN. Based on this idea, Saito & Nakano [25] proposed a system that could generate rules. Unfortunately, however, the number of generated rules was too high, hindering their understanding. Trying to avoid this problem, Thrun developed the system VIA [26], which uses a procedure similar to classical sensibility analysis. Next Craven & Shavlik [27] developed an algorithm which is well-known by the complexity and the quality of the generated rules. Other examples of systems that use the pedagogical technique are RULENET [12], which extracts conjunctive rules, BRAINNE [28], which works with continuous data in the input layer (without the need for an initial discretization) and DEDEC [13], which extracts symbolic rules from a trained ANN.

3 Overview of Data Mining, Neural Networks and Genetic Algorithms

3.1. Data Mining with Neural networks

In essence the goal of data mining is to extract accurate, comprehensible knowledge from data. There are several kinds of data mining tasks, such as classification, clustering, discovery of association rules, etc. For an overview of the main data mining tasks, see e.g. [29]. This work focus on the well-known classification task [30], [31], where the goal is to predict the value of a goal (class) attribute given the value of predicting attributes.

In the context of classification, a popular neural network model is a fully-connected feedforward network. The network is organized in layers. The neurons in the input layer correspond to predicting attribute values of the data set being mined, whereas the output layer represents the predicted class for the record having the predicting attribute values given to the input layer.

A well-known limitation of neural networks in the context of data mining is that the "knowledge" of the system consists of numeric weights distributed across a large number of interconnections. Unfortunately, this representation does not provide a comprehensible explanation for the classification of a record. Since the comprehensibility of the discovered knowledge is important in data mining, as mentioned above, it is desirable to extract high-level, comprehensible rules from neural networks.

In this work the extracted rules are IF-THEN classification rules, where the IF part of the rule specifies a conjunction of conditions on predicting attribute values and the THEN part specifies a predicted value for the goal (or class) attribute.

The extraction of this kind of rule has the following advantages: (a) It allows human users to understand the output of the system; (b) It avoids the well-known knowledge engineering bottleneck of knowledge acquisition for symbolic systems; (c) It leads to an integration of the symbolic and connectionist approaches.

3.2. Evolving Neural Networks with Genetic Algorithms

The performance of a neural network is directly related to its architecture and parameters. Hence, the choice of an architecture for a neural network influences the learning time, the predictive accuracy, tolerance to noise and generalization ability of the network.

Although there are some mathematical methods to determine the architecture and parameters of a neural network, these methods usually are very computationally expensive, since an optimal definition of a neural network architecture is an NP-complete problem [16]. This motivates the use of a robust, global search method such as genetic algorithms to determine a good (though not necessarily optimal) network architecture.

Genetic algorithms are stochastic search algorithms based on abstractions of the processes of Neo-Darwinian evolution. The basic idea is that each "individual" of an evolving populations encodes a candidate solution to a given problem (in our case, a candidate neural network topology). Then these individuals evolve towards better and better individuals (i.e. better and better candidate solutions) via operators based on natural selection, i.e. survival and reproduction of the fittest, and genetics, e.g. recombination and mutation operators.

An important characteristic of genetic algorithms is that they perform a *global* search. Indeed, genetic algorithms work with a population of candidate solutions, rather than working with a single candidate solution at a time. This, together with the fact they that use stochastic operators to perform their search, reduce the probability that they will get stuck in local maxima, and increase the probability that they will find the global maximum [14].

A crucial issue in the design of a genetic algorithm is the choice of the fitness function. This is the function used to evaluate the quality of an individual, and it is the function to be optimized in the target problem. The higher the fitness of an individual, the higher the probability that it will be selected to participate in the formation of the next generation of individuals. In our case, the fitness function involves both the predictive accuracy and the comprehensibility of rules extracted from the individual's network topology, as will be seen later.

After a set of individuals is selected for reproduction, other genetic operators - typically crossover and mutation - are applied to the selected individuals, to form a new, probably improved generation of individuals (recall that the new individuals inherit genetic material from some of the fittest individuals of the previous generation).

Crossover is a recombination operator that swaps genetic material between two individuals. In essence this operator works in two steps, as follows. First, an integer position k (called crossover point) is selected at random, so that k lies between two genes (string elements) of the pair of individuals to undergo crossover. Second, two new individuals are created by swapping all genes at the right of the crossover point k. A simple example of the application of the crossover operator is presented in Figure 1. As shown in Figure 1(a) the first individual has genes denoted

X1...X4, while the second individual has genes denoted Y1...Y4. The crossover point k was randomly chosen as 2. Hence, the two genes corresponding to positions 3 and 4 are swapped between the two original individuals, producing the two new individuals shown in Figure 1(b).

The mutation operator simply changes the value of a gene to a new random value. In the simplest case, if a gene can take on a binary value the mutation operator inverts the current value of the gene - i.e. a “0” mutates into a “1” and vice-versa. During a run of a GA, mutation is typically applied with a much lower frequency than crossover. A high frequency of mutation would be undesirable, because the GA would tend to behave like a random search; on the contrary, a low frequency of mutation is desirable, since it increases the genetic diversity of individuals in the current population.

X1X2	X3X4	X1X2	Y3Y4
Y1Y2	Y3Y4	Y1Y2	X3X4
(a) Before crossover		(b) After crossover	

Figure 1: Simple example of crossover in genetic algorithms.

4 The proposed method for rule extraction from neural networks via genetic algorithms

As mentioned above, this work proposes the use of a genetic algorithm to find a feedforward neural network topology which leads to high-quality (accurate and comprehensible) extracted rules. In our genetic algorithm each individual of the population represents a neural network topology trained by the RPROP algorithm [32]. This topology is then passed to a rule extraction algorithm (see below), whose output is a set of classification rules. The quality of these rules is then evaluated with respect to both predictive accuracy and comprehensibility. The result of this evaluation is used as the fitness value of the individual.

It should be noted that in our approach the fitness function evaluates the quality of a rule set, rather than the quality of a single rule. This is an important advantage of the proposed approach, since it copes naturally with the problem of rule interaction. (After all, a set of *good rules* is not necessarily a *good set* of rules.)

The genetic algorithm for evolving neural network topologies was implemented by using the ENZO tool [33], whose flow of execution is shown in Figure 2. ENZO evolves a fully-connected feedforward neural network with a single hidden layer. However, the ENZO tool does not offer any facility for extracting rules from the trained neural network. Therefore, we have extended the ENZO software with a new module in charge of extracting rules from the network topology of an individual. The flow of execution of the extended ENZO tool, including the rule extraction module, is shown in Figure 3. The new module is an adaptation of the rule extraction algorithm RX [6], which will be described in section 4.1. This algorithm was chosen due to its relative simplicity and good results reported in the literature [6], but we make no claim that it is the best algorithm for extracting rules from trained neural networks. In any case, note that the basic idea of our approach, namely combining genetic algorithms for evolving neural network topologies with a rule extraction algorithm, could be easily applied to several other rule extraction algorithms.

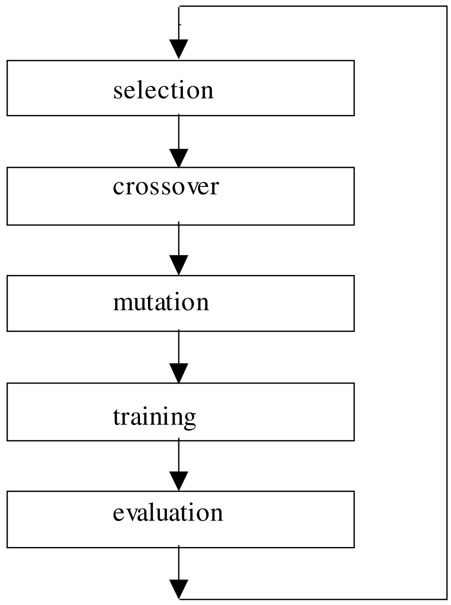

Figure 2: Evolution cycle of Enzo.

We emphasize that the rule extraction module is executed once for each individual of the population. This module not only extracts a set of rules from the network topology associated with the individual but also evaluates the quality of this rule set with respect to both predictive accuracy and comprehensibility, as mentioned above.

4.1. Rule Extraction

As mentioned above, our rule extraction algorithm is an adaptation of the the rule extraction algorithm RX [6], as described in Figure 4. It essentially works as follows.

First of all, the algorithm discretizes the activation values of the hidden layer neurons via clustering. Then it enumerates the discretized activation values of each hidden layer neuron and, for each combination of those values, it propagates them through the network in order to obtain the activation values in the output layer neurons. By observing the association between the propagated combination of values and the output activation values, the algorithm extracts rules where the antecedent (IF part) contains the discretized hidden layer values and the consequent (THEN part) contains the output layer values. A similar process is used to extract rules where the antecedent contains the input layer values and the consequent contains the hidden layer values. Finally, these two rule sets are combined, by generating a set of rules where the antecedent contains the input layer values and the consequent contains the output layer values.

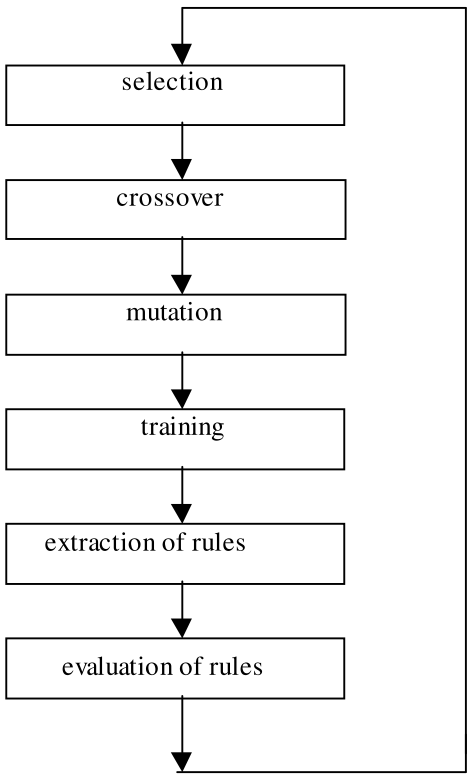

Figure 3: Evolution cycle of Enzo with new module.

The value of the parameter ε in the second line of the algorithm described in Figure 4 was empirically set to 0.8 in our experiments.

We have adapted the RX algorithm in two ways, as follows.

First, note that the original algorithm RX was designed to be run *once*, accepting as input a trained (and duly pruned) neural network. Hence, the output of the algorithm is the final set of rules to be reported to the user. Therefore, the algorithm extracts rules from a neural network trained on the entire training data set, and the extracted rules are evaluated on a separate test set. In our case the rule extraction algorithm is run many times (once for each population individual, as mentioned above), and the quality of the extracted rules must be fed back to the genetic algorithm, to guide the selection operator.

Hence, we cannot evaluate the quality of the extracted rules on the test set. Therefore, we divide the available data into three mutually exclusive and exhaustive data sets: a training set, a validation set and a test set. Only the training and validation sets are used during the evolution of the genetic algorithm. Once a neural network is trained on the training set, the rule extraction algorithm is executed by accessing only the training set. However, once the rules are extracted, the quality of those rules is evaluated on the validation set, separate from the training set. This avoids an *overfitting* of the rules to the training set. The result of the evaluation of the extracted rules on the validation set is then fed back to the genetic algorithm as the fitness of the corresponding individual (network topology). Finally, after the evolution of the genetic algorithm is completed, the rule set extracted from the best individual of the last generation is considered the best found rule set and is reported to the user. The quality of this rule set is evaluated on the test set, separate from the training and validation sets.

To compute the complexity of a rule set we have borrowed a formula from [6]. This formula is shown in equation (3):

$$Complexity = 2 * R + C \,, \qquad (3)$$

where R is the number of extracted rules and C is the total number of rule conditions in the extracted rules. However, since we want to combine the Acc and the Complexity measures into a single fitness function, is necessary to normalize the value of the Complexity measure, so that it returns a value between 0 and 1. (The value of Acc in equation (2) is already normalized.) By using normalized version of the Complexity measure, our fomula for the comprehensibilty of a rule set is as shown in equation (4).

$$Comprehensibilty = 1 - \frac{2 * \frac{R}{Max\_R} + \frac{C}{Max\_C}}{3} \qquad (4)$$

where R is the number of rules, C is the total number of rule conditions, Max_R is the maximum number of rules extracted from an individual, among all individuals generated so far, and Max_C is the maximum number of rule conditions extracted from an individual, among all individuals generated so far.

Finally, we are now ready to specify our fitness function combining both the accuracy rate (Acc) and the Comprehensibility of an extracted rule set. Since the ENZO tool minimizes the fitness of individuals (i.e. the lower the value of the fitness function, the better the individual), we have used the following fitness function:

$$Fitness = 1 - W_{acc}Acc + W_{comp}Comprehensibility \qquad (5)$$

where W_{acc} and W_{comp} are user-defined weights for the Acc and Comprehensibility terms. Hence, the user is free to set the values of W_{acc} and W_{comp} in such a way to assign greater importance to the predictive accuracy or the comprehensibility of the extracted rule set, depending on his/her interest and on the application domain.

Activation value discretization via clustering:

(a) Let ε (0,1). Let D be the number of discrete activation value in the hidden node.
Let δ_1 be the activation value in the hidden node for the first pattern in the training set. Let $H(1) = \delta_1$, $count(1) = 1$, $sum(1) = \delta_1$ and set $D = 1$.

(b) For all patterns i = 2,3,4...k in the training set:
- Let δ be its activation value.
- If there exists an index J such that
$$|\delta - H(J)| = \min_{j \in \{1,2,3,...D\}} |\delta - H(J)| \, and \, |\delta - H(J)| \leq \varepsilon,$$
then set count(J) := count(J) + 1,
sum(J) := sum (J) + δ
else D = D +1, H(D) =δ.

(c) Replace H by the average of all activation values that have been clustered into this cluster:
$$H(j) := sum(j) / count(j),\ j = 1,2,...,D.$$

1. Enumerate the discretized activation values and compute the network output. Generate perfect rules that have a perfect cover of all the tuples from the hidden node activation values to the output values.
2. For the discrtized hidden node activation values appeared in the rules found in the above step, enumerate the input values that lead to them, and generate perfect rules.
3. Generate rules that relate the input values and output values by rule substitution based on the results of the above two steps.

Figura 4: Algorithm of rules extraction adapted (RX).

In all the experiments reported in this paper W_{acc} and W_{comp} were set to 0.75 and 0.25, respectively.

4.2. Generation of initial population and selection operator

The method used to generate the initial population tries to determine the minimum number N of neurons in the hidden layer that leads to a good generalization ability. Once this number is determined, individuals are generated by having in their hidden layer a number of nodes randomly chosen between N and the maximum number of nodes chosen by user.

The selection method is based on ranking, i.e. the individuals are ranked in decreasing order of fitness, so that the higher the ranking of an individual the higher the probability that it will be selected for reproduction.

Two crossover operators were used. The first one inserts into the child all the connections occurring in both parents. For each inserted connection, its weight in the child is the average of the weights of the connection in the parents. If a given connection occurs only in one parent, that connection is inserted into the child with a prespecified crossover probability. The second crossover operator inserts a randomly-chosen hidden unit of the fittest parent into the child, with the appropriate connections according to the current topology of the child.

Two mutation operators were used. The first one inserts a new unit to an individual or remove one of its units. The second mutation operator inserts a new input unit to an individual, together with all the appropriate connections according to the current topology of the individual, or remove one of its units. Additional information about the crossover and mutation operators can be found in [33].

4.3. Data Sets and Experiments

We have evaluated our rule extraction algorithm on three data sets available from the UCI Machine Learning repository (http://www.ics.uci.edu/AI/Machine-Learning.html). The data sets in question are Iris,Wine and Monks 1.

Note that the rule extraction algorithm requires that the data being mined have discrete values. However, two data sets used in our experiments have continuous attributes. We have discretized the continuous attributes of these data sets by running the C4.5-Disc algorithm as a pre-processing step (before evolving the neural network). C4.5-Disc has been shown to be an effective discretization algorithm [34], and it is essentially a modified version of the well-known C4.5 decision tree algorithm [35]. C4.5-Disc works as follows. C4.5 is applied to each continuous attribute separately. Hence, C4.5 builds a decision tree where the internal nodes contain only binary partitions on that continuous attribute's values and then applies tree pruning to find an appropriate number of nodes in the tree - i.e. the appropriate number of discretization intervals. After the tree is pruned we simply use the threshold values at each internal tree node for a discretization of the continuous attribute. In our experiments we used all the default values for the parameters of C4.5.

The first data set used in our experiments is the Iris data set. This data set has 150 examples, 4 predicting attributes and a goal attribute which can take on 3 classes. The predicting attributes were continuous, but they were discretized as explained above, for rule extraction purposes.

In our experiments with the Iris data set the initial population of genetic algorithm individuals consisted of a fully-connected feedforward neural network with 13 neurons in the input layer, between N and 4 neurons in the hidden layer (where N is determined by the algorithm, as explained in the beginning of section 4.3) and 3 neurons in the output layer.

The second data set used in our experiments is the Wine data set. This data set has 178 examples, 13 predicting attributes and a goal attribute which can take on 3 classes. The predicting attributes were continuous, but they were discretized as explained above, for rule extraction purposes.

In our experiments with the Wine data set the initial population of genetic algorithm individuals consisted of a fully-connected feedforward neural network with 43 neurons in the input layer, between N and 4 neurons in the hidden layer and 3 neurons in the output layer.

The third data set used in our experiments is the Monks-1 data set. This data set has 432 examples, 6 predicting attributes and a goal attribute which can take on 2 classes. The predicting attributes were nominal.

In our experiments with the Monk-1 data set the initial population of genetic algorithm individuals consisted of a fully-connected feedforward neural network with 15 neurons in the input layer, between N and 4 neurons in the hidden layer and 1 neurons in the output layer.

We used the same genetic algorithm parameters for all the data sets. More precisely, the population consisted of 30 individuals evolving during 30 generations. The probabilities of crossover and mutation were set to the default values in the ENZO software.

5 Computational Results

We have evaluated our rule extraction method with respect to both the classification accuracy rate and the comprehensibility of the extracted rules. More precisely, we have made two kinds of comparison: (a) We have compared the accuracy rate and the comprehensibility of the rule set extracted by our method against the accuracy rate and the comprehensibility of the rule set extracted by C4.5, a well-known decision-tree algorithm [35]; (b) We have compared the accuracy rate of the rule set extracted by our method against the accuracy rate of a neural network evolved by ENZO without using any rule-extraction algorithm. This latter comparison allows us to evaluate how much accuracy rate we are sacrificing in order to gain comprehensibility.

Our experiments have used the well-known methodology of cross validation [31], with a cross-validation factor of five. In other words, the data set is partitioned into five data subsets and the rule extraction algorithm is run five times. In each run a distinct data subset is used as the test set and the remaining four partitions are used as the training set. After the cross-validation procedure was performed, we merged all the five partitions into a single data set again, in order to generate the final rules to be shown to the user. (This is necessary because, of course, we cannot compute an "average rule set" over the rule sets discovered by the five iterations of the cross-validation procedure.) The graphs and rules presented in this section were obtained through the application of the proposed method using 70% of the examples for training and 30% for validation.

The results are reported in Table 1. This table shows the results of a five-factor cross-validation experiment for the three data sets. The first column indicates the data set used. The second column indicates the accuracy rate (on the test set) of the rule set extracted from the evolved neural network by using our method. The third column indicates the complexity of the rule set extracted by our method, as measured by equation (3). The fourth and fifth columns indicate respectively the accuracy rate (on the test set) and complexity of the rule set found by C4.5. Finally, the sixth column indicates the accuracy rate of a neural network evolved by ENZO without using any rule-extraction algorithm.

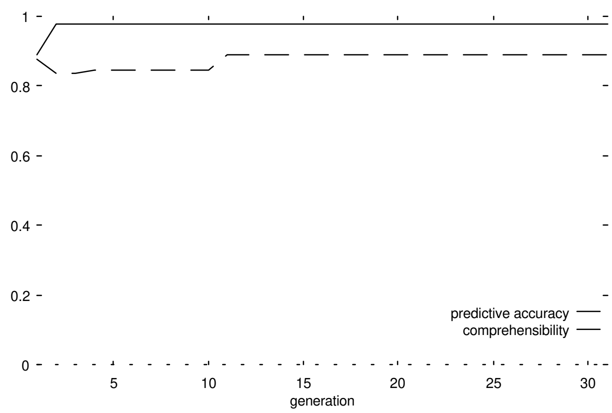

Figure 5: Evolution of rule accuracy and rule comprehensibility in the Iris data set.

The first row of Table 1 shows the results for the Iris data set. As can be seen in the table, our system was able to extract rules with high predictive accuracy and high comprehensibility (low complexity). The accuracy rate of the discovered rules was 93.33%, slightly smaller than the accuracy rate of the neural network (94%) and slightly smaller than the accuracy rate of the rules discovered by C4.5 (94.68%). In the context of data mining, this minor reduction in accuracy rate is a small price to pay for the large gain in the comprehensibility of the discovered knowledge. The rule set obtained for the Iris data set is as follows.

Table 1: Results

data set	Accuracy rate of the rule set (proposed method)	Complexity of the rule set (proposed method)	Accuracy rate of the rule set (C4.5)	Complexity of the rule set (C4.5)	Accuracy rate of the neural network
Iris	93.33%	10.6	94.68%	16.2	94.00%
Wine	85.32%	88.6	82.46%	18.4	100.00%
Monks-1	99.77%	34.2	100%	107	100%

Rule 1: (a3 ≤ 1.9) AND (a4 ≤ 0.6) ⇒ Class 1.
Rule 2: (a3 ≤ 1.9) AND (a4 > 1.7) ⇒ Class 1.
Rule 3: (a3 > 1.9) AND (a4 ≤ 0.6) ⇒ Class 3.
Rule 4: (a3 > 1.9) AND (a4 > 1.7) ⇒ Class 3.
Default class => Class 2

Figure 5 presents the evolution of rule accuracy and rule comprehensibility in the Iris data set. The curves refer to the performance of the best individual of each generation.

The second row of Table 1 shows the results for the Wine data set. Unlike the results for the Iris data set, the table shows that the extracted rules have an accuracy rate (85,32%) significantly smaller than the neural network's accuracy rate (100%), but larger than the accuracy rate of the rules discovered by C4.5 (82.46%). Although the complexity of the rule set discovered by our method (88.6) is significantly larger than the complexity of the rule set discovered by C4.5 (18.4), our method still has the advantage of discovering a somewhat comprehensible rule set, which is more useful for human decision making than the output of the neural network without any rule extraction. The rule set obtained for the Wine data set is as follows.

Rule 1: (a5 ≤ 132) AND (a11 > 0.78) AND (a12 > 2.47) AND (a13 > 750) ⇒ Class 1.
Rule 2: (a5 ≤ 132) AND (a11 > 0.78) AND (a11 ≤ 0.97) AND (a12 ≤ 2.47) AND (a13 ≤ 750) ⇒ Class 3.
Rule 3: (a5 ≤ 132) AND (a11 ≤ 0.78) AND (a12 ≤ 2.11) ⇒ Class 3.
Default class ⇒ Class 2.

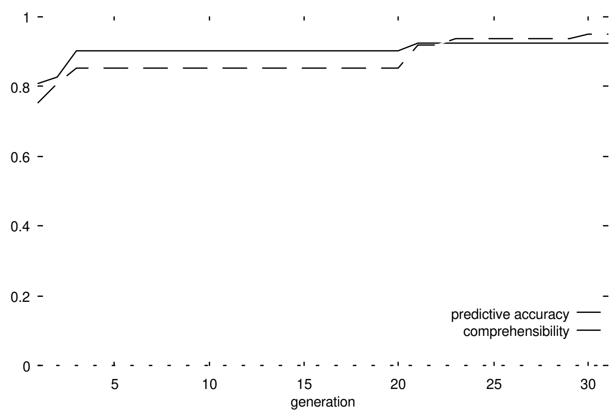

Figure 6: Evolution of rule accuracy and rule comprehensibility in the Wine data set.

Figure 6 presents the evolution of rule accuracy and rule comprehensibility in the Wine data set. The curves refer to the performance of the best individual of each generation.

The third row of Table 1 shows the results for the Monks-1 data set. As can be seen in the table, our system was able to extract rules with high predictive accuracy and high comprehensibility (low complexity). The accuracy rate of the discovered rules was 99.77%, slightly smaller than the accuracy rate of the neural network (100%) and slightly smaller than the accuracy rate of the rules discovered by C4.5 (100%). In the context of data mining, this minor reduction in accuracy rate is a small price to pay for the large gain in the comprehensibility of the discovered knowledge. The rule set obtained for the Monks-1 data set is as follows.

Rule 1: (a1 = 1) ⇒ Class 2
Rule 2: (a2 = 3) AND (a5 = 1) ⇒ Class 2
Rule 3: (a2 = 2) AND (a5 = 1) ⇒ Class 2
Default class ⇒ Class 1

Figure 7 presents the evolution of rule accuracy and rule comprehensibility in the Monks-1 data set. The curves refer to the performance of the best individual of each generation.

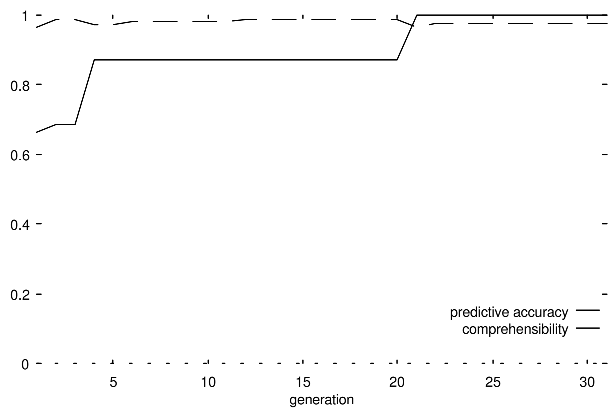

Figure 7: Evolution of rule accuracy and rule comprehensibility in the Monks-1 data set.

6 Conclusions

We have introduced a system to extract comprehensible rules from a neural network whose topology is evolved by a genetic algorithm, and have evaluated its performance on three public domain data sets. The computational results have shown that in two of the data sets the system extracted a very compact, comprehensible rule set without unduly reducing the accuracy rate, in comparison with the accuracy rate of the rule set discovered by the well-known C4.5 algorithm. In the other data set, however, there was a significant decrease in accuracy rate when the neural network was converted into a rule set. However, in this data set our rule extraction method achieves an accuracy rate which is larger than the one achieved by the rule set discovered by C4.5.

We should emphasize that in the experiments reported in this paper the genetic algorithm's initial population consisted of neural networks with a small number of neurons in the hidden layer. This design decision was made not only to facilitate the discovery of comprehensible rules but also to save computational time. However, one might ask whether this has significantly hindered the search for a good network topology. The answer seems to be no, since in general the number of neurons in the hidden layer was further reduced during the evolutionary process, so that the neural networks in the last generation of individuals had an even smaller number of neurons in the hidden layer.

In any case, future research will include a more extensive set of experiments with other data sets, in order to better validate the results reported in this paper.

Bibliography

[1] R.J. Henery. Classification. In: D. Michie et al. (Eds.) *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[2] M. Bohanec and I. Bratko. Trading accuracy for simplicity in decision trees. *Machine Learning 15,* 223-250. 1994.

[3] J. Catlett. Overpruning large decision trees. *Proc. 12th Int. Joint Conf. on AI (IJCAI-91).* Sidney, 1991.

[4] L.A. Brewlow and D.W. Aha. Simplifying decision trees: a survey. *The Knowledge Eng. Review 12(1)*, 1997, 1-40.

[5] L. Fu. Neural Networks in computer intelligence. McGraw-Hill, 1994.

[6] Lu, H., Setiono, H., Liu, H. NeuroRule: a connectionist approach to data mining. *Proc. 21st Conf. on Very Large Databases*. Zurich, 1995.

[7] M. L. Vaughn. Interpretation and Knowledge discovery from a multilayer perceptron network: opening the black box. Neural Comput & Applic. 4, 72-82, 1996.

[8] R. Andrews, J. Diederich, and A.B. Tickle. A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks. Site: *http://157.225.15.98/Ken_pubs.html*, Brisbane, Australia, 1998.

[9] G.G. Towell, and J.W. Shavlik. The Extraction of Refined Rules from Knowledge-Based Neural Networks. *Machine Learning*, v. 31, n. 1, p. 71-101, 1993.

[10] C. McMillan, M.C. Mozer, and P. Smolensky. The Connectionist Scientist Game: Rule Extraction and Refinement in a Neural Network. In: *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1991.

[11] R. Andrews, and S. Geva. Rule Extraction from a Constrained Error Back-Propagation MLP. In: Proceedings of the 6th Australian Conference on Neural Networks, p. 9-12, Brisbane Queensland, 1994.

[12] E. Pop, R. Hayward, and J. Diederich. RULENEG: Extracting Rules from a Trained ANN by Stepwise Negation. *QUT NRC*, 1994.

[13] A.B. Tickle, M. Orlowski, and J. Diederich. DEDEC: Decision Detection by Rule Extraction from Neural Networks, *QUT NRC*, 1994.

[14] Goldberg, D. E. *Genetic algorithms in search, optimization and machine learning.* Reading, MA: Addison Wesley, 1989.

[15] Pan Z. & Kang L., Evolving Both the topology and weights of neural networks, Parallel Alogrithms and Applications, Vol. 9, pp. 299-307, 1996.

[16] R. F. Albrecht, C. R. Regves, N. C. Steele, Representation and Evolution of Neural Networks., Procedings ICNNGA 93 - INNSBRUCK-Austria 1993. pp. 643-649.

[17] J. W. Shavlik & D. W. Optiz, Using Genetic Search to Refine Knowledge-Based Neural Networks, 11° International Conference of Machine Learning-1994.

[18] Y. Hayashi. A Neural Expert System Using Fuzzy Teaching Input. In: *Proceedings of The IEEE International Conference on Fuzzy Systems*, p. 485-491, San Diego, CA, 1989.

[19] Y. Hayashi and J.J. Buckley. Aproximation Between Fuzzy Expert Systems and Neural Networks. *International Journal of Aproximate Reasoning*, v. 10, n. 1, p. 63-73, 1994.

[20] R. Matsuoka, N. Watanabe, A. Kawamura, Y. Owada, and K. Asakawa. Neurofuzzy Systems – Fuzzy Inference Using a Structured Neural Network. In: *Proceedings of the International Conference on Fuzzy Logic and Neural Networks*, p. 173-177, Lizuka, Japan, 1991.

[21] H.R. Berenji. Refinement of Approximate Reasoning-based Controllers by Reinforcement Learning. In: *Proceedings of the Eighth Machine Learning Workshop*, p. 475-479, Evanston, IL, 1991.

[22] S. Horikawa, T. Foruhashi, and Y. Uchikawa. On Fuzzy Modeling Using Fuzzy Neural Networks with the Back-Propagation Algorithm. IEEE Transactions on Neural Networks, v. 3, n. 5, p. 801-806, 1992.

[23] L.M. Fu. Rule Learning by Searching on Adapted Nets. In: *Proceeding of The International Conference on Artificial Intelligence*, AAAI-91, p. 325-340, Anaheim, CA, 1991.

[24] G.G. Towell, and J.W. Shavlik. Knowledge-based Artificial Neural Networks. *Artificiail Intelligence*, v. 69, n. 1, 1994.

[25] K. Saito, and R. Nakano. Medical Diagnostic Expert System Based on PDP Model. In: *Proceedings of the International Conference on Neural Networks*, v. 1, p. 255-262, San Diego, CA, 1991.

[26] S.B. Thrun. Extracting Probably Correct Rules from Artificial Neural Networks. Relatório Técnico IAI-TR-93-5, Institut für Informatik III, Universität Bonn, Germany, 1994.

[27] M.W. Craven, and J.W. Shavlik. Extracting Tree-structured Representations of Trained Networks. *Advances in Neural Information Processing Systems*, v. 8, n. 1, 1996.

[28] K. Sestito, and T. Dillon. The Use of Sub-Symbolic Methods for the Automation of Knowledge Acquisition for Expert Systems. In: *Proceedings of the 11th International Conference on Expert Systems and Their Applications – AVIGNON'98*, p. 317-328, Avignon, France, 1991.

[29] Fayyad, V.M.et.al. From data mining to Knowledge Discovery: an overview. In:Gayyad, V.M.et.al. (Eds.) *Advances in Knowledge Discovery and Data Mining*,1-34. AAAI/MIT,1996.

[30] D. Michie, D. J. Spiegelhalter and C. C. Taylor. Machine Learning, Neural and Statistical Classification. New York: Ellis Horwood, 1994.

[31] D. Hand. Construction and assessment of classification rules. John Wiley & Sons, 1997

[32] Zell, A., Mamier, G., Vogt, M., et al., SNNS – Stuttgart Neural Network Simulator – User Manual, Version 4.1, University of Stuttgart; 1995.

[33] Braun, H., Ragg, T., *ENZO – User Manual and Implementation Guide*, Version 1.0, University of Karlsruhe; 1995.

[34] R. Kohavi & M. Sahami. Error-based and entropy-based discretization of continuous features. *Proc. 2nd Int. Conf. Knowledge Discovery & Data Mining*, 114-119. AAAI Press, 1996.

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