Effective Data Mining Using Neural Networks *

Hongjun Lu Rudy Setiono Huan Liu

Department of Information Systems Computer Science

National University of Singapore

Kent Ridge, Singapore 119260

{luhj,rudys,liuh}@iscs.nus.sg

Abstract

Classification is one of the data mining problems receiving great attention recently in the database community. This paper presents an approach to discover symbolic classification rules using neural networks. Neural networks have not been thought suited for data mining because how the classifications were made is not explicitly stated as symbolic rules that are suitable for verification or interpretation by humans. With the proposed approach, concise symbolic rules with high accuracy can be extracted from a neural network. The network is first trained to achieve the required accuracy rate. Redundant connections of the network are then removed by a network pruning algorithm. The activation values of the hidden units in the network are analyzed, and classification rules are generated using the result of this analysis. The effectiveness of the proposed approach is clearly demonstrated by the experimental results on a set of standard data mining test problems.

1 Introduction

One of the data mining problems is *classification*. Various classification algorithms have been designed to tackle the problem by researchers in different fields such as mathematical programming, machine learning, and statistics. Recently, there is a surge of data mining research in the database community. The classification problem is re-

*This work is partly supported by the National University Research Grant RP950660. An early version of this paper appeared in VLDB'95 Proceedings

examined under the context of large databases. Unlike researchers in other fields, database researchers pay more attention to the issues related to the volume of data. They are also concerned with the effective use of the available database techniques, such as efficient data retrieval mechanisms. With such concerns, most algorithms proposed are basically based on decision trees. The general impression is that the neural networks are not well suited for data mining. The major criticisms include the following:

1. Neural networks learn the classification rules by multiple passes over the training data set so that the learning time of a neural network is usually long.

2. A neural network is usually a layered graph with the output of one node feeding into one or many other nodes in the next layer. The classification process is buried in both the structure of the graph and the weights assigned to the links between the nodes. Articulating the classification rules becomes a difficult problem.

3. For the same reason, available domain knowledge is rather difficult to be incorporated to a neural network.

On the other hand, the use of neural networks in classification is not uncommon in machine learning community [5]. In some cases, neural networks give a lower classification error rate than the decision trees but require longer learning time [7, 8]. In this paper, we present our results from applying neural networks to mine classification rules for large databases [4] with the focus on articulating the classification rules represented by neural networks. The contributions of our study include the following:

- Different from previous research work that excludes the neural network based approaches entirely, we argue that those approaches should have their position in data mining because of its merits such as low classification error rates and robustness to noise.

- With our rule extracting algorithms, symbolic classification rules can be extracted from a neural network. The rules usually have a comparable classification error rate to those generated by the decision tree based methods. For a data set with a strong relationship among attributes, the rules extracted are generally more concise.

- A data mining system based on neural networks is developed. The system successfully solves a number of classification problems in the literature.

Our neural network based data mining approach consists of three major phases:

1. *Network construction and training*

   This phase constructs and trains a three layer neural network based on the number of attributes and number of classes and chosen input coding method.

2. *Network pruning*

   The pruning phase aims at removing redundant links and units without increasing the classification error rate of the network. A small number of units and links left in the network after pruning enable us to extract concise and comprehensible rules.

3. *Rule extraction*

   This phase extracts the classification rules from the pruned network. The rules generated are in the form of "if $(a_1 \theta v_1)$ *and* $(x_2 \theta v_2)$ *and* ... *and* $(x_n \theta v_n)$ then C_j" where a_i's are the attributes of an input tuple, v_i's are constants, θ's are relational operators $(=, \leq, \geq, <>)$, and C_j is one of the class labels.

Due to space limitation, in this paper we omit the discussion of the first two phases. Details of these phases can be found in our earlier work [9, 10]. We shall elaborate in this paper the third phase. Section 2 describes our algorithms to extract classification rules from a neural network and uses an example to illustrate how the rules are generated using the proposed approach. Section 3 presents some experimental results obtained. Finally, Section 4 concludes the paper.

2 Extracting rules from a trained neural network

Network pruning results in a relatively simple network. However, even with a simple network, it is still difficult to find the explicit relationship between the input tuples and the output tuples. A number of reasons contribute to the difficulty of extracting rules from a pruned network. First, even with a pruned network, the links may be still too many to express the relationship between an input tuple and its class label in the form of *if* ... *then* $\cdots$ rules. If a network still has n input links with binary values, there could be as many as 2^n distinct input patterns. The rules could be quite lengthy or complex even for a small n. Second, the activation values of a hidden unit could be anywhere in the range [-1,1] depending on the input tuple. It is difficult to derive an explicit relationship between the continuous activation values of the hidden units and the output values of a unit in the output layer.

2.1 A rule extracting algorithm

The rule extraction algorithm, RX, consists of the four steps given below.

Rule extraction algorithm (RX)

1. Apply a clustering algorithm to find clusters of hidden node activation values.
2. Enumerate the discretized activation values and compute the network outputs. Generate rules that describe the network outputs in terms of the discretized hidden unit activation values.
3. For each hidden unit, enumerate the input values that lead to them and generate a set of rules to describe the hidden units' discretized values in terms of the inputs.
4. Merge the two sets of rules obtained in the previous two steps to obtain rules that relate the inputs and outputs.

The first step of **RX** clusters the activation values of hidden units into a manageable number of discrete values without sacrificing the classification accuracy of the network. After clustering, we obtain a set of activation values at each hidden node. The second step is to relate these discretized activation values with the output layer activation values, i.e., the class labels. And the third step is to relate them with the attribute values at the nodes connected to the hidden node. A general purpose algorithm X2R was developed and implemented to automate the rule generation process. It takes as input a set of discrete patterns with the class labels and produces the rules describing the relationship between the patterns and their class labels. The details of this rule generation algorithm can be found in our earlier work [3].

To cluster the activation values, we used a simple clustering algorithm which consists of the following steps:

1. Find the smallest integer d such that if all the network activation values are rounded to d-decimal-place, the network still retains its accuracy rate.
2. Represent each activation value α by the integer nearest to $\alpha \times 10^d$. Let $\mathcal{H} = \mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_H$ be the set of these discrete representations. Set $i = 1$.

3. Sort the set $\mathcal{H}$ such that the values of $\mathcal{H}_i$ are in increasing order.

4. Find a pair of distinct adjacent values $h_{i,j}$ and $h_{i,j+1}$ in $\mathcal{H}_i$ such that if $h_{i,j+1}$ is replaced by $h_{i,j}$, no conflicting data will be generated.

5. If such values exist, replace $h_{i,j+1}$ by $h_{i,j}$ and repeat Step 4. Otherwise, set $i = i + 1$, if $i \leq H$ go to Step 3.

This algorithm tries to merge as many as possible values into one interval in every step as long as the merge does not produce conflicting data, i.e. two sets of activation values that are the same but they represent tuples that belong to different classes. The algorithm is order sensitive, that is, the resulting clusters depend on the order in which the hidden units are clustered.

2.2 An illustrative example

In [1], ten classification problems are defined on datasets having nine attributes: salary, commission, age, elevel, car, zipcode, house-value, house-years, and loan. We use one of the problems, Function 3 as an example to illustrate how classification rules can be generated from a network. It classifies tuples as Group A according to the following definition:

Group A: $((\mathbf{age} < 40) \wedge (\mathbf{elevel} \in [0..1])) \vee ((40 \leq \mathbf{age} < 60) \wedge (\mathbf{elevel} \in [1..3])) \vee ((\mathbf{age} \geq 60) \wedge (\mathbf{elevel} \in [2..4]))$

All tuples that do not satisfy the conditions for membership in Group A are labeled B.

To solve the problem, a neural network was first constructed. Each attribute value was coded as a binary string for use as input to the network. The length of the strings used to represent the attribute values are summarized in Table 1. The thermometer coding scheme was used for the binary representations of the continuous attributes. Each bit of a string was either 0 or 1 depending in which subinterval the original value was located. For example, a salary value of 140k would be coded as $\{1,1,1,1,1,1\}$ and a value of 100k as $\{0,1,1,1,1,1\}$. For the discrete attribute, **elevel**, for example, an **elevel** of 0 would be coded as $\{0,0,0,0\}$, 1 as $\{0,0,0,1\}$, etc.

With the coding scheme shown in Table 1 we had a total of 37 binary inputs. The 38th input was added to the network to incorporate the bias in each hidden unit. As the tuples were classified into two classes only, a single unit at the output layer was sufficient. The target output was 1 if the tuple belonged to Group A, and 0, otherwise. The number of the hidden units was initially set as six. The training data set consisted of 2000 tuples. One pruned

Table 1: Coding of the attributes for neural network input.

Attribute	No. of inputs	Subintervals
salary	6	$[20k, 25k), [25k, 50k), [50k, 75k), [75k, 100k), [100k, 125k), [125k, 150k]$
commission	3	$[0, 25k), [25k, 50k), [50k, 75k]$
age	6	$[20, 30), [30, 40), [40, 50), [50, 60), [60, 70), [70, 80]$
elevel	4	$[0], [1], [2], [3], [4]$
car	4	$[1, 5], [6, 10], [11, 15], [15, 20]$
zipcode	3	$[1, 3], [3, 6], [6, 9]$
hvalue	3	$[0.450k), [450k, 900k), [900k, 1350k]$
hyears	3	$[1, 10], [10, 20], [20, 30]$
loan	5	$[0, 100k), [100k, 200k), [200k, 300k), [300k, 400k), [400k, 500k]$

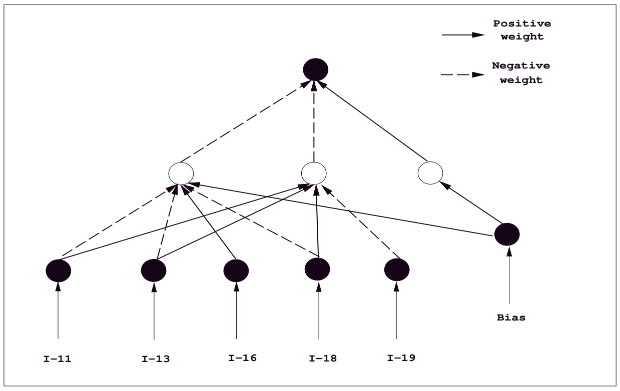

Figure 1: A pruned network for function 3.

network that achieved 100% accuracy on the test data set is depicted in Figure 1. Recall that the initial network had 38 input units, 6 hidden units, 1 output unit, and therefore 234 links. The pruned network is much simpler.

To extract the rules from the pruned network, the hidden unit activation values are first clustered. There were two clusters of activation values found at each of the two hidden units. Tuples were split into two sets by the first hidden unit: those with activation values in the subinterval $[-1, 0.46)$ and those in $[0.46, 1)$. The subintervals of the second hidden unit were $[-1, 0.81)$ and $[0.81, 1]$. Let $\alpha_j = 1 \;\; or \;\; 2$ for $j = 1 \;\; or \;\; 2$ represent an activation value of a pattern at hidden unit j located in its first or second subinterval. When all tuples were grouped according to their activation values at the two hidden units, it was found that there were only 3 different groups. These were patterns with $(\alpha_1, \alpha_2) = (1, 2), (2, 1),$ or $(2, 2)$. The rules that distinguish tuples in Group A from those in Group B are:

Rule 1. If $\alpha_1 = 1$ or if $\alpha_2 = 1$, then Group A.

Default rule. Group B.

The activation values of a tuple at the two hidden units were determined by the relevant network's weights and the values of attributes **age** and **elevel** as follows (cf. Fig. 1). At hidden unit 1, we find $\alpha_1 = 1$ if and only if $\delta(-1.71I_{11} - 1.38I_{13} + 1.30I_{16} - 3.95I_{18} + 4.53) < 0.46$, where $\delta(.)$ is the hyperbolic tangent function. At hidden unit 2, we have $\alpha_2 = 1$ if and only if $\delta(1.07I_{11} + 1.87I_{13} + 3.99I_{18} - 1.81I_{19}) < 0.81$.

A set of rules that did not involve the network's weights were generated by X2R for each hidden unit. For hidden unit 1, X2R took as its input 4-bit binary strings $(I_{11}, I_{13}, I_{16}, I_{18})$. Because of the coding scheme used (cf. Table 2), there were not 16, but only 9 possible combinations of 0-1 values for these strings. The class label for each of these input strings was either 1 or 2, depending on the activation value. Three strings labeled 1 corresponded to original input tuples that had activation values in the interval $[-1, 0.46)$. The remaining six strings were labeled 2 since they represented inputs that had activation values in the second subinterval, $[0.46, 1]$. The rules generated were as follows:

Rule 1. If $(I_{11} = I_{13} = I_{18} = 1)$, then $\alpha_1 = 1$; Rule 2. If $(I_{13} = I_{18} = 1$ and $I_{16} = 0)$, then $\alpha_1 = 1$.

Default rule. $\alpha_1 = 2$.

Similarly, for hidden unit 2, the input for X2R were the binary string $(I_{11}, I_{13}, I_{18}, I_{19})$. There were also 9 possible combinations of 0-1 values that can be taken by the four inputs. Three input strings were labeled 1, while the rest 2. X2R generated the following rules:

Rule 1. If $(I_{11} = I_{18} = 0$ and $I_{19} = 1)$, then $\alpha_2 = 1$; Rule 2. If $(I_{13} = I_{19} = 0)$, then $\alpha_2 = 1$.

Default rule. $\alpha_2 = 2$.

The conditions of the rules that determine the output in terms of the activation values can now be rewritten in terms of the inputs. After removing some redundant conditions, the following set of rules for Function 3 were obtained:

R1. If $I_{11} = I_{18} = 1$, then **Group A**.

R2. If $I_{13} = I_{18} = 1$ and $I_{16} = 0$, then **Group A**.

R3. If $I_{11} = I_{18} = 0$ and $I_{19} = 1$, then **Group A**

Table 2: The coding scheme for attributes **age** and **elevel**

Age	I_{10}	I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	**elevel**	I_{16}	I_{17}	I_{18}	I_{19}
$[20, 30)$	0	0	0	0	0	1	0	0	0	0	0
$[30, 40)$	0	0	0	0	1	1	1	0	0	0	1
$[40, 50)$	0	0	0	1	1	1	2	0	0	1	1
$[50, 60)$	0	0	1	1	1	1	3	0	1	1	1
$[60, 70)$	0	1	1	1	1	1	4	1	1	1	1
$[70, 80]$	1	1	1	1	1	1					

R4. If $I_{13} = I_{19} = 0$, then **Group A**

Default rule. **Group B**

Hence from the pruned network, we extracted 5 rules with a total of 10 conditions. To express the rules in terms of the original attribute values, we refer to Table 2 which shows the binary representations for attributes **age** and **elevel**. Referring to Table 2, the above set of rules is equivalent to:

R1. If **age** ≥ 60 and **elevel** $\in [2, 3, 4]$, then **Group A**.

R2. If **age** ≥ 40 and **elevel** $\in [2, 3]$, then **Group A**.

R3. If **age** < 60 and **elevel** $= 1$, then **Group A**.

R4. If **age** < 40 and **elevel** $= 0$, then **Group A**.

Default rule. **Group B**.

It is worth to highlight the significant difference between the decision tree based algorithms and the neural network approach for classification. Figure 2 depicts the decision boundaries formed by the network's hidden units and output unit. Decision tree algorithms split the input space by generating two new nodes in the context of binary data. As the path between the root node and a new node gets longer, the number of tuples becomes smaller. In contrast, the decision boundaries of the network are formed by considering all the available input tuples as a whole. The consequence of this fundamental difference of the two learning approaches is that a neural network can be expected to generate fewer rules than do decision trees. However, the number of conditions per rule will generally be higher than that of decision trees' rules.

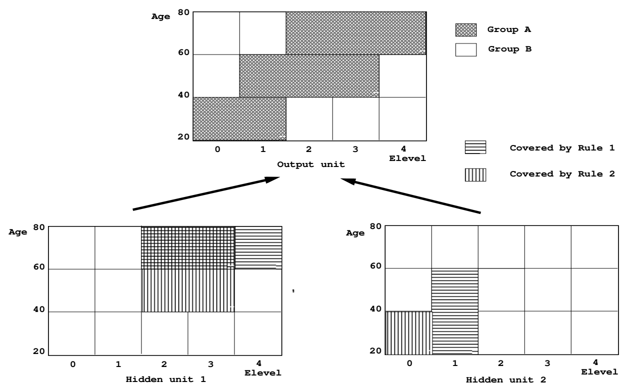

Figure 2: Decision boundaries formed by the units of the pruned network for Function 3.

3 Experimental results

To test the neural network approach more rigorously, a series of experiments were conducted to solve the problems defined by Agrawal *et al.* [1]. Among 10 functions described, we found that functions 8 and 10 produced highly skewed data that made classification not meaningful. We will only discuss functions other than these two.

The values of the attributes of each tuple were generated randomly according to the distributions given in [1]. Following Agrawal *et al.* [1], we also included a perturbation factor as one of the parameters of the random data generator. This perturbation factor was set at 5 percent. For each tuple, a class label was determined according to the rules that define the function. For each problem, 3000 tuples were generated. We used three-fold cross validation to obtain an estimate of the classification accuracy on the rules generated by the algorithm.

Table 3 summarizes the results of the experiments. In this table, we list the average accuracy of the extracted rules on the test set, the average number of rules extracted, and the average number of conditions in a rule. These averages are obtained from 3×10 neural networks. Error due to perturbation in the test data set was subtracted from the total error, and the average accuracy rates shown in the table reflect this correction.

For comparison, we have also run C4.5 [6] for the same data sets. Classification rules were generated from the trees by C4.5rules. The same binary coded data for neural networks were used for C4.5 and C4.5rules. Figures 3-5 show the accuracy, number of rules, and the average number of conditions in the rules generated by two approaches. We can see that the two approaches are comparable in accuracy for most functions except for Function 4. On the other hand, while the average numbers of conditions in both rule sets are almost the same for all the functions, the

Table 3: Averages of accuracy rates on the test, the number of rules, and the average conditions per rule obtained from 30 networks. Standard deviations appear in parentheses.

Func.	Accuracy	No. of rules	No. of conditions
1	99.91 (0.36)	2.03 (0.18)	2.23 (0.50)
2	98.13 (0.78)	7.13 (1.22)	4.37 (0.66)
3	98.18 (1.56)	6.70 (1.15)	3.18 (0.28)
4	95.45 (0.94)	13.37 (2.39)	4.17 (0.88)
5	97.16 (0.86)	24.40 (10.18)	4.68 (0.87)
6	90.78 (0.43)	13.13 (3.72)	4.61 (1.02)
7	90.50 (0.92)	7.43 (1.76)	2.94 (0.32)
9	90.86 (0.60)	9.03 (1.65)	3.46 (0.36)

number of rules generated by the neural network approach is much less than that of C4.5 rules. For functions 1 and 9, C4.5 generated as much as 5 times number of rules. For all other functions except for function 5, C4.5 also generated as twice as much rules compared to the neural network based approach.

4 Conclusion

In this paper we present a neural network based approach to mining classification rules from given databases. The approach consists of 3 phases: (1) constructing and training a network to correctly classify tuples in the given training data set to required accuracy; (2) pruning the network while maintaining the classification accuracy; and (3) extracting symbolic rules from the pruned network. A set of experiments was conducted to test the proposed approach using a well defined set of data mining problems. The results indicate that, using the proposed approach, high quality rules can be discovered from the given test data sets.

The work reported here is our attempt to apply the connectionist approach to data mining to generate rules closer to that of decision trees. A number of related issues are to be further studied. One of the issues is to reduce the training time of neural networks. Although we have been improving the speed of network training by developing fast algorithms, the time required to extract rules by our neural network approach is still longer than the time needed by the decision tree based approach, such as C4.5. As the long initial training time of a network may be tolerable in some cases, it is desirable to have incremental training and rule extraction during the life time of an application database. With an incremental training that requires less time, the accuracy of rules extracted can be improved along with the change of database contents. Another possibility to reduce the training time and improve the classification accuracy is to reduce the number of input units of the networks by feature selection.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6), December 1993.

[2] J.E. Dennis Jr. and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice Hall, Englewood Cliffs, NJ, 1983.

[3] H. Liu and S.T. Tan. X2R: A fast rule generator. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics.* IEEE, 1995.

[4] H. Lu, R. Setiono, and H. Liu. Neurorule: A connectionist approach to data mining. In *Proceedings of VLDB'95*, 478–489, 1995.

[5] D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood Series in Artificial Intelligence, 1994.

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

[7] J.R. Quinlan. Comparing connectionist and symbolic learning methods. In S.J. Hanson, G.A. Drastall, and R.L. Rivest, editors, *Computational Learning Theory and Natural Learning Systems*, volume 1, 445–456. A Bradford Book, The MIT Press, 1994.

[8] J.W. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.

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

[10] R. Setiono. A penalty function approach for pruning feedforward neural networks. *Neural Computation*, 8(8), 1996.

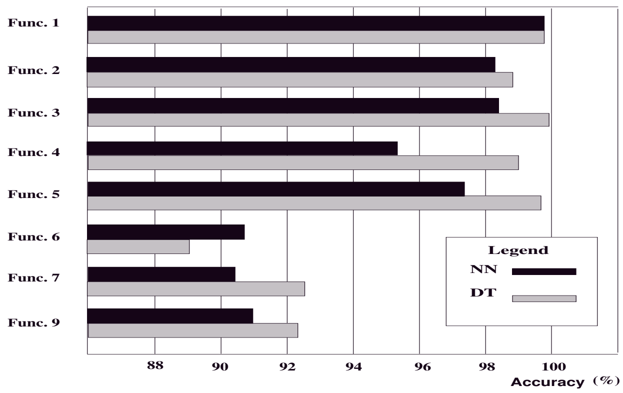

Figure 3: Accuracy of the rules extracted from neural networks (NN) and by C4.5rule (DT).

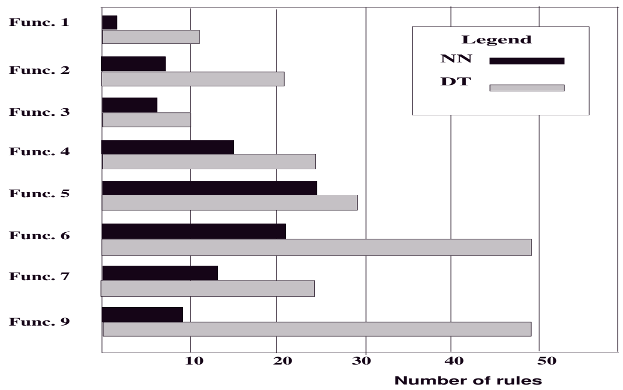

Figure 4: The number of the rules extracted from neural networks (NN) and by C4.5rule (DT).

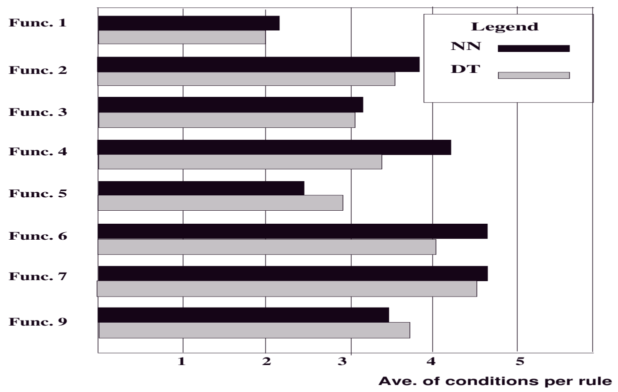

Figure 5: The number of conditions per neural network rule (NN) and C4.5 rule (DT).