Theory and Methodology

Financial credit-risk evaluation with neural and neurofuzzy systems

Selwyn Piramuthu 1

351 Bus, P.O. Box 117169, Graduate School of Business, Decision and Information Sciences, University of Florida, Gainesville FL 32611-7169, USA

Received 18 December 1996; accepted 26 August 1997

Abstract

Credit-risk evaluation decisions are important for the financial institutions involved due to the high level of risk associated with wrong decisions. The process of making credit-risk evaluation decision is complex and unstructured. Neural networks are known to perform reasonably well compared to alternate methods for this problem. However, a drawback of using neural networks for credit-risk evaluation decision is that once a decision is made, it is extremely difficult to explain the rationale behind that decision. Researchers have developed methods using neural network to extract rules, which are then used to explain the reasoning behind a given neural network output. These rules do not capture the learned knowledge well enough. Neurofuzzy systems have been recently developed utilizing the desirable properties of both fuzzy systems as well as neural networks. These neurofuzzy systems can be used to develop fuzzy rules naturally. In this study, we analyze the beneficial aspects of using both neurofuzzy systems as well as neural networks for credit-risk evaluation decisions. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Neural networks; Neurofuzzy systems; Credit-risk evaluation

1. Introduction

Credit plays an important role in the lives of many people and in almost all industries that involve monetary investment in some form (Chirinko et al., 1991; Koch, 1995). Obtaining credit is inevitable for smooth and effective operation of industries. The value of credit depends on the need and urgency in obtaining the required credit. It is especially critical when the current worth of that credit is multifold when compared to what it would be worth in the future. In other words, if that credit is not obtained immediately, its worth decreases precipitously due to the loss of opportunities for which the credit was required in the first place. This is more pronounced in industries that operate in areas where there is a tremendous growth in technology and where, loosely speaking, today’s technology would be outdated tomorrow. Credit is also essential for acquisition of capital-intensive investments that would be hard to obtain otherwise.

1 Tel.: 352 392 9600; fax: 352 392 5438.
In the process of evaluating credit-risk, the party involved in making the decision investigates factors that may lead to default in the repayment of loan. The characteristics considered while evaluating credit may vary depending on the type of loan in question. Some of the common differences in these characteristics are whether the loan involves short or intermediate or long term, the purpose of the loan, type of borrower, collateral, source of repayment, and interest rate, among others.

Credit-risk evaluation decisions are inherently complex due to the various forms of risks involved. The numerous and varied risks in approving credit stem from the many factors that can lead to the non-payment of obligation when they become due (Reed et al., 1980). Due to these complexities, the chances of making an error in credit-risk evaluation decision is large even if done carefully. Given a choice, the party approving the credit is better off erring on the safe side resulting in approval of credit for most of the deserving clients and none of the risky ones, while possibly denying credit for some deserving clients. This is especially evident when the stakes associated with a wrong decision are high.

Although there is no fail-proof method currently available for credit-risk evaluation, decisions involving evaluation of credit-risk are done quite often that it requires extremely careful scrutiny. The fact that the risks associated with clients are inherently different depending on a multitude of factors, past experience can only be used as a guideline for future decisions. The payoff is usually high when a right credit-risk evaluation is made. Any improvement in the methods that are currently being used would result in significant payoffs for the party involved in the credit-risk evaluation.

We study the performance of neural networks and neurofuzzy systems, methods that were developed under the rubric of artificial intelligence, for financial credit-risk evaluation. There has been an increased interest in using neural networks for various purposes over the last decade, resulting in a spate of applications in the financial domain. Financial credit-risk evaluation certainly has its share in these recently developed methods. In fact, neural networks and related methods are being used quite commonly in credit evaluation decisions. Although neurofuzzy systems have been in existence only for a few years, both neural networks and fuzzy systems have been studied for a few decades already. Neurofuzzy systems are still being evaluated for plausible applications. Both neural networks and neurofuzzy systems have their inherent advantages. In this paper, we study their performance using three credit-risk evaluation data sets.

This paper is organized as follows: Section 2 provides an overview of neural networks and neurofuzzy systems, covering their primary characteristics as well as their advantages and disadvantages. Experimental results using both neural networks and neurofuzzy systems are given in Section 3, and Section 4 concludes the paper with a brief discussion of preliminary lessons learned and future extensions of this study.

2. Neural networks and neurofuzzy systems

In this section, we briefly discuss the characteristics of both neural networks as well as neurofuzzy systems. We also compare and contrast the major advantages and disadvantages of both these systems.

2.1. Neural networks

Neural networks have been gaining popularity recently due to their beneficial properties such as excellent generalizability, ease of modelling without need for specifying the model (e.g., as in regression applications) up-front, no assumptions about underlying distributions of data used, ability to learn any function, among others. These applications span a wide range of areas. The most commonly used architectures are the feedforward neural network and its variants, for classification problems.

A feedforward neural network consists of layers of processing units, where the input signals are fed forward from the input layer through the network to the processing units in output layer. Various
methods are used to learn weights in the links in these networks. The backpropagation algorithm has been one of the most popular learning algorithms in these feedforward neural networks due to its simplicity in terms of development. The backpropagation algorithm learns through propagating error in the network backwards, until learning (convergence) occurs.

The number of processing units in the input layer of these feedforward neural networks are decided by the number of inputs (independent variables) in the data. The number of processing units in the output layer corresponds to the number of output variables and their range of values. For example, a 0-1 (2 class) classification problem needs a 1-output-unit network; an output variable with eight different values can be modeled using three (resulting in $2^3$ different combinations) output units; two binary output variables need two output units.

There is no structured practical means to determine the number of hidden layers and the number of processing units in these layers (e.g., Baum and Hausler, 1989; Huang and Huag, 1991; Kung and Hwang, 1988), although studies have shown that a hidden layer is sufficient to learn any function (e.g., Hornik et al., 1989; Hornik, 1991). This has resulted in the selection of these by trial and error or by heuristics. For example, NeuralWorks (1996) recommends $(\text{number of inputs} + \text{outputs}) \times (2/3)$ as the number of hidden units using only one hidden layer, though they also recommend thorough experimentation by increasing/decreasing the number of units.

There have also been studies where hidden units are added one after another while testing the performance of the network (e.g., Fahlman and Lebiere, 1990). Several other researchers have studied variations on this theme (e.g., Frean, 1990; Hirose et al., 1991; Mohraz and Protzel, 1996; Wermter and Meurer, 1996). The addition of new processing units in the hidden layer is stopped when the performance of the neural network stops improving or improves less than a predetermined threshold. A drawback with these methods of adding hidden units incrementally is that they assume the number of hidden units to be a simple convex function of the performance (e.g., classification, generalization) of the neural network. However, it is possible that this function is multimodal with several local optima and a good probability of ending up in a local optimum.

The primary advantage in using feedforward neural networks is its excellent generalizability property. This property is especially useful when dealing with incomplete data, i.e., the data does not cover all possible range of variable values and heretofore unseen cases are expected to be processed later. This occurs frequently in financial domains such as credit-risk evaluation, where data used to evaluate credit-risk of a new firm need not be similar to the cases seen thus far. Although no method is fail-proof, performing well in a majority of the cases proves to be more than adequate in these domains where the stakes are high and any improvement over methods used previously can be beneficial.

Feedforward neural networks are also noise tolerant due to the distributed representation of knowledge amongst the various links. Also, common problems in statistical analyses such as explicit assumptions on distributions of data and interactions among the variables are not major problems in feedforward neural networks. Unlike some other learning methods, real-valued inputs can be used in neural networks. Feedforward neural networks and the algorithms used to train these networks are inherently parallel in structure, and hence can be implemented in parallel machines to improve learning time.

The speed of convergence of most learning algorithms used in these feedforward neural networks is a major drawback. However, since the need for learning is infrequent in most circumstances, it does not pose much of a problem. There are implementations of faster methods that circumvent this problem (see for example, Fahlman, 1988), by assuming that the error surface is convex and continuous. The result of learning (weights in the links) in these neural networks are cryptic, which is a problem if the user wants to know the reasoning that led to a certain conclusion. There have been studies that deal with extracting rules from neural networks (e.g., Fu, 1989; Sun, 1992; Towell, 1991), although these methods make compromises and assumptions to obtain the rules,
which do not represent the knowledge learned in the neural networks accurately. The number of hidden units and various learning parameters (e.g., momentum, learning rate) need to be set by the user, usually from past experience or by trial and error. As mentioned before, there are no structured means to determine the best set of these values. For a detailed overview of feedforward neural networks, the reader is referred to Rumelhart et al. (1986).

2.2. Neurofuzzy systems

Both neural networks and fuzzy systems have their own individual merits. We discussed some of the merits of neural networks in the previous subsection. Fuzzy systems are advantageous when dealing with imprecise information which cannot be handled with regular probability measures. A degree of membership is assigned to these quantities, which do not fall into binary category so as to either belong or not belong to a certain classification. A neurofuzzy system combines the desirable properties of both neural networks and fuzzy systems to form a system that is easy to use, with good performance.

Neurofuzzy systems (Buckley and Hayashi, 1994; Gupta and Rao, 1994; Kuncheva and Mitra, 1994; Nauck et al., 1993) are similar to feedforward neural networks, in structure. The primary difference lies in the fuzzification and defuzzification of data in neurofuzzy systems. Fuzzification is the process of converting deterministic (also known as crisp) values (or range of values) to fuzzy (e.g., low, medium, high) values, and defuzzification is the inverse process of fuzzification used to convert fuzzy values to their corresponding crisp values. The process of fuzzification/defuzzification in a Neurofuzzy system occurs at various levels – pre-processing stage (Klimasauskas, 1995), fuzzy weights (Yamakawa, 1990), both fuzzy input as well as fuzzy weights (Buckley and Hayashi, 1992; Nauck et al., 1993). In this study, we consider the latter, i.e., a neurofuzzy system with fuzzy weights, activations, and processing. Crisp inputs are fuzzified and fed into these neurofuzzy systems generating fuzzy outputs which are then defuzzi-fied in the neurofuzzy systems used. These systems generate rules in IF-THEN form as learning proceeds. Similar to the weights that are learned in neural networks, rules in IF-THEN form are learned by neurofuzzy systems.

The decision with regards to the number of hidden units to be used as well as setting parameter values such as learning rate, etc., still needs to be done by the user, usually by trial and error, where the number of hidden units can be set to the maximum number of IF-THEN rules desired. The number of IF-THEN rules changes dynamically as learning proceeds in these networks. Hence, a number \(n\) is chosen such that it lies close to the maximum number of rules that are generated during the initial few iterations (epochs). If the actual number of IF-THEN rules exceeds this number, the best \(n\) of these rules are kept during each epoch. We do not want to throw away too many of these IF-THEN rules that are generated, while also not keeping everything that is generated. There needs to be a tradeoff between avoiding overfitting the data and throwing away valuable learned information in the form of IF-THEN rules.

The problem with interpreting the learned results from a neural network, in the form of cryptic weights, is avoided in a neurofuzzy system. Learning in neurofuzzy systems result in natural language rules in IF-THEN form which are easily understood by humans. These are also extremely useful when trying to explain the reasoning behind any output by a neurofuzzy system. The IF-THEN rules generated by a neurofuzzy system are simple enough that they could be used in an expert system, if necessary.

The generation of IF-THEN rules in these neurofuzzy systems enables the use of both qualitative as well as quantitative data as input to the system. In addition to the rules that are generated by the neurofuzzy system, external ‘hints’ can be introduced as IF-THEN rules. The system uses both these sets – the ones generated from training examples and the ones provided by the user as ‘hints’ – for classification purposes. The process of providing these ‘hints’ is extremely useful in situations where additional knowledge is provided by domain expert(s). Usually, in real-world situa-
tions, these ‘hints’ provide information that is not available in the training data set, and they are the result of the expert’s many years of experience in dealing with the domain of interest. The process of providing ‘hints’ is not easy in neural networks, although possible through appropriate modification of weights that involves much work by the user. The modification of a weight in a feedforward neural network has ramifications in the layers that follow the layer of interest, hence the complication is even multiplied. Whereas, in a neurofuzzy system, new knowledge in the form of IF-THEN rules. The neurofuzzy system then selects the rules that it generated as well as the rules that were provided by the user, and proceeds with its next iteration.

As in neural networks, both real-valued as well as nominal-valued variables are used as input in neurofuzzy systems. Due to fuzzification of the inputs, the input variables have few fuzzified values (e.g., small, medium, large) instead of the whole range of real-numbered values. This results in a huge reduction in learning time for neurofuzzy systems.

The neurofuzzy system in this study is fuzzified in terms of weights, inputs to the network, and the activations of the output units. The learning algorithm (Halgamuge and Glesner, 1994; Nauck et al., 1993) for the neurofuzzy system used in this study can be described as follows:

1. **Initialize**: Set number of units in input (\(N\)), output (\(M\)), and maximum number of rule (hidden) units (\(H\)). The rule units are units in the hidden layer(s) of the neurofuzzy system. Select learning rates for parameters of fuzzy set – e.g., \(\sigma_a\), \(\sigma_b\), and \(\sigma_c\), if triangular fuzzy sets are used with parameters \(a\), \(b\), and \(c\). Here, \(a\), \(b\), and \(c\) are the \(x\)-coordinate values of a variable corresponding to the triangular function representation, with \(a\) and \(c\) representing the two vertices of the triangle that lie along the \(x\)-axis and \(b\) the projection of the third vertex of the triangle on the \(x\)-axis. \(\sigma_a\), \(\sigma_b\), and \(\sigma_c\) are the corresponding learning rates for \(a\), \(b\), and \(c\), respectively. Initialize the system with any manually input IF-THEN rules.

2. **Generate rule units**: Until all examples have been selected (i.e., used for generating rules):
   a. Select next example, consisting of input/output vectors \(i_1, \ldots, i_N, o_1, \ldots, o_M\).
   b. Calculate membership functions for each input feature \(x_i (i = 1 \ldots N)\), with each \(x_i\) consisting of \(q\) fuzzy sets \(\mu_{1}^{(q)} \ldots \mu_{q}^{(q)}\):
      \[\mu_{i}^{(q)}(p_i) = \max_{j \in \{1, \ldots, q\}} \{\mu_{j}^{(q)}(x_i)\},\]
      where \(\mu_{1}^{(q)} \ldots \mu_{q}^{(q)}\) are fuzzy sets corresponding to each input feature.
   c. If there is no rule node \(H_i\) with \(W'(H_i, H) = \mu_{1}^{(1)} \ldots W'(H_i, H) = \mu_{q}^{(q)}\), create this node and connect it to output node if class = 1. \(W'\) defines links in the network and maps on to \(F(\mathcal{R})\), which is the set of all fuzzy subsets of \(\mathcal{R}\).

3. **Select rule units**: Select the best \(H\) rule units and discard the remaining rule units from further consideration.

4. **Determine actual output**: Select next example and determine actual output \((a_i)\) by propagating the example through the network.

5. **Update \(\delta\) (a measure proportional to the difference between the actual and target values) of the values**:
   \[\delta = a (1 - a) \sum_i W'(H, N)(o_i - a_i),\]
   where \(o_i\) is the target concept.

6. **Determine \(\sigma\) (learning rate) values**: For example, using \(\delta\), calculate \(\sigma_a\), \(\sigma_b\), and \(\sigma_c\), if triangular fuzzy sets are used with parameters \(a\), \(b\), and \(c\).

7. **Repeat**: Go to step 4 until a stopping criteria is reached.

It should be noted here that the rules generated in these neurofuzzy systems are not cryptic, and are not hidden in a black box as in neural networks. In step 1, the neurofuzzy system either starts without any rules or it can be initialized using prior knowledge in the form of IF-THEN rules. The prior knowledge used in these neurofuzzy systems can be either qualitative or quantitative. Qualitative data can be input directly during step 1. Quantitative data is first pre-processed at step 1 to fit the shape/parameters of fuzzy...
sets used. For example, if a triangular fuzzy set is used, the input data is processed according to the parameters \( \sigma_a, \sigma_b, \) and \( \sigma_c \) of the fuzzy set. As new rules are created and integrated with these set of rules in step 2, no distinction is made as to the origin (i.e., manually input to the neurofuzzy system or generated by the neurofuzzy system) of the rules in later steps (3–7). Although as many rules the data demands are generated during step 2, only the best \( H \) rules are kept for further evaluation during step 3.

The maximum number of rule (hidden) units \( (H) \) is selected by the user based on experimentation. Also, the final number of rules in the system may not equal \( H \) if that many are not input/created during steps 1 and 2.

The final stopping criteria (step 7 in the algorithm given above) could be (1) a predetermined number of epochs (iterations), (2) a predetermined error value of the learned concepts, (3) a predetermined number of iterations during which no improvement in the error value occurs. The algorithm stops if any of these criteria is satisfied.

3. Experimental results

There have been very few studies comparing the performance of neural networks and neurofuzzy systems. Herrmann et al. (1995) used medical diagnosis data to study the classification performance of feedforward neural networks using backpropagation, radial basis function (RBF) networks, dynamic vector quantization (DVQ), and neurofuzzy systems. For detailed discussion on RBF and DVQ the reader is referred to Jang and Sun (1993) and Halgamuge et al. (1995) as well as the references given therein, respectively. The best classification results were obtained using DVQ. They also state the advantages of using neurofuzzy systems: human readability of the structure, and possibility of including a priori knowledge. Tuma et al. (1996) used fuzzy expert systems, neural networks as well as neurofuzzy systems to develop emission oriented production control strategies. Their results indicate that neural network control resulted in further increase in production compared to that using neurofuzzy control.

We study the performance of neural networks as well as neurofuzzy systems using three real-world applications data. These data sets have been used in previous studies, and involve financial credit-risk evaluation in various forms: credit approval, loan default, and bank failure prediction.

3.1. Credit approval data

We use the credit approval data that was used in Quinlan (1987) in this study. The data set was cleaned to remove examples with missing attribute values. This data is from a large bank. Each of the examples in this data corresponds to a credit card application, with nine discrete and six real attributes. The discrete attributes have anywhere from 2 through 14 possible values. This is a binary classification data, corresponding to positive and negative decisions. There were 690 examples in this data set. We removed the incomplete examples and ended up with 653 examples of which 296 belong to positive class and 357 belong to negative class, where the classes correspond to whether or not credit was approved. This data set is also known to be noisy.

To study the generalizability property of both neural networks and neurofuzzy systems, we split the data into a training and a testing (holdout data) set. The training set consisted of 490 examples, of which 268 and 222 examples corresponded to positive and negative class values, respectively. The testing set consisted of 163 examples, of which 89 and 74 examples corresponded to positive and negative class values respectively. The 10 training and testing samples used in the 10 runs of both neural networks and neurofuzzy systems were all randomly selected from the whole data set.

For the neural network the learning rates were set at 0.19, since we found this to be a good number from experimentation as well as past experience. These networks were allowed to run until the total sum of squares of error terms (tss) reached 0.04, or until the number of iterations (epochs) reached 2000, or if minimum error is not decremented for more than 10 epochs. In order to be consistent with previous studies, the limit on the number of epochs was set at 2000. None of these
neural networks converged, and were run until 2000 epochs were reached. The number of input units were set at 15, corresponding to the number of input attributes. The number of output units was set at 1, corresponding to the binary output class values. The number of hidden units in a hidden layer were set at 8. Based on past experience, we chose this as the average of input and output units since there is no available method to analytically and/or experimentally determine the optimal number of hidden units.

The learning rates \( (\alpha_a, \alpha_b, \text{ and } \alpha_c) \) of the triangular membership functions were set at 0.015, again from experimentation as well as past experience in dealing with neurofuzzy systems. One of the possible reasons for these low learning rates, compared to those used in neural networks, is the large number of units used in these networks. As knowledge is distributed among these large number of units, the rate of learning is kept low for better learning results. For each input unit, the same number (3) of fuzzy partitioning is defined by equally distributed triangular membership functions. These systems were allowed to run until the tss reached 0.04, or the number of epochs reached 2000, or if minimum error is not decremented for more than 10 epochs. Weighted sum was used as the aggregate function for the units in the output layer in the neurofuzzy system, where the activations of output units are the mean of the connected rule unit activations.

All these neurofuzzy systems converged by 50 epochs, based on non-decrement of minimum error for more than 10 epochs. These networks had 15 input units corresponding to the 15 input attributes and an output unit corresponding to the binary output class values. By trial and error, we set the number of hidden units to be 250, based on the maximum initial number of rules that were created in these systems. Although there could have been more than 250 rules that were created at any given epoch as learning proceeded, we set the maximum at 250. Hence, only the 250 best rules were used at any given epoch.

The results using credit approval data in neural networks and neurofuzzy systems described above are given in Tables 1 and 2, respectively.

<table>
<thead>
<tr>
<th>#</th>
<th>Net configuration</th>
<th>Epochs</th>
<th>Time (s)</th>
<th>Classification training (%)</th>
<th>Classification testing (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15-8-1</td>
<td>2000</td>
<td>2642</td>
<td>95.9183</td>
<td>97.546</td>
</tr>
<tr>
<td>2</td>
<td>15-8-1</td>
<td>2000</td>
<td>2754</td>
<td>95.3061</td>
<td>87.116</td>
</tr>
<tr>
<td>3</td>
<td>15-8-1</td>
<td>2000</td>
<td>2576</td>
<td>96.3265</td>
<td>79.754</td>
</tr>
<tr>
<td>4</td>
<td>15-8-1</td>
<td>2000</td>
<td>3135</td>
<td>95.7142</td>
<td>93.251</td>
</tr>
<tr>
<td>5</td>
<td>15-8-1</td>
<td>2000</td>
<td>2648</td>
<td>95.5102</td>
<td>80.981</td>
</tr>
<tr>
<td>6</td>
<td>15-8-1</td>
<td>2000</td>
<td>2926</td>
<td>94.6938</td>
<td>85.889</td>
</tr>
<tr>
<td>7</td>
<td>15-8-1</td>
<td>2000</td>
<td>2886</td>
<td>95.7142</td>
<td>79.141</td>
</tr>
<tr>
<td>8</td>
<td>15-8-1</td>
<td>2000</td>
<td>2994</td>
<td>94.6938</td>
<td>76.687</td>
</tr>
<tr>
<td>9</td>
<td>15-8-1</td>
<td>2000</td>
<td>2880</td>
<td>96.1224</td>
<td>82.822</td>
</tr>
<tr>
<td>10</td>
<td>15-8-1</td>
<td>2000</td>
<td>2424</td>
<td>95.9183</td>
<td>72.392</td>
</tr>
</tbody>
</table>

Average 2786 (204.80) 95.59 (0.53) 83.56 (7.22)

The standard deviation values are given in parentheses.
the data is noisy, and it is hard for any method to learn to generalize noisy examples. The classification results using the training examples were more or less consistent, based on the standard deviation values for the training examples being small.

Table 2 provides the results using neurofuzzy systems using the same data as in Table 1. Each of the rows in both Tables 1 and 2 correspond to the same set of training and testing example sets, respectively. The time taken by the neurofuzzy system was less than that taken by the neural network. It should be noted that neurofuzzy systems were run using a 66 MHz IBM-compatible PC whereas the neural networks were run in a SUN-4 machine. In spite of the difference in machines, the neurofuzzy systems were faster using a slower machine. This could be because of the fewer number of epochs, although the neurofuzzy systems were slower on a per epoch basis due to the large number of hidden units. If we use connection updates\(^3\) as a measure to compare the methods, we still would have the neurofuzzy system performing in lesser number of connection updates compared to the neural networks. The neurofuzzy systems converged after fewer epochs whereas the neural networks did not converge even after 2000 epochs.

In terms of classification accuracies, both the training as well as the testing classification performance of the neurofuzzy systems were worse than those using the neural networks. The classification accuracy on testing data using neurofuzzy system was significantly worse (at 0.1 level of significance using the two-tailed t-test) compared to those using neural networks. This could be attributed to the approximations of both the inputs as well as the output made by the neurofuzzy system, as it fuzzified the inputs and defuzzified the output. The neural networks did not use any such approximations. The classification accuracies using neurofuzzy system is also influenced by the overlap in the way the range of values of a given attribute is split into its various categories (e.g., range of values for small, medium, and large). Again, these are pitfalls associated with the mechanisms used for both fuzzification and defuzzification of input and output data, respectively. Currently, there is no optimal means to fuzzify/defuzzify data. These problems could be alleviated as progress is made in this area.

### 3.2. Loan default data

This data has been used in previous studies (e.g., Abdel-Khalik and El-Sheshai, 1980), to classify a set of firms into those that would default and those that would not default on loan payments. The source of this data is the Index of Corporate Events in the 1973–1975 issues of

---

\(^3\) Connection updates are measured by the product of the number of connections in the network times the number of epochs.
Disclosure Journal. Sixteen defaulted firms were matched with 16 non-defaulted firms to obtain data for the study. These are used for training the neural systems. Another (holdout) set of 16 examples, all belonging to the non-default case, were used to test the learned concepts. There are 18 variables in this data: (1) net income/total assets (NITA), (2) net income/sales (NIS), (3) total debt/total assets (TDTA), (4) cash flow/total debt (CFTD), (5) long-term debt/net worth (LDNW), (6) current assets/current liabilities (CACL), (7) quick assets/sales (QAS), (8) quick assets/current liabilities (QACL), (9) working capital/sales (WCS), (10) cash at year-end/total debt (CYTD), (11) earnings trend (ET), (12) sales trend (ST), (13) current ratio trend (CT), (14) trend of L.T.D./N.W. (TLN), (15) trend of W.C./sales (TWS), (16) trend of N.I./T.A. (TNT), (17) trend of N.I./sales (TNS), and (18) trend of cash flow/T.D. (TFT).

For detailed description of this data, the reader is referred to Abdel-Khalik and El-Sheshai (1980). Since the data set was already split into training and testing sets, we decided to go with the same split of the data as in Abdel-Khalik and El-Sheshai (1980). We ran the neural networks 10 times, due to the random initial weight settings and took the average of the results. The neurofuzzy system was run only once since this does not involve the use of any random initial weights. As before, the learning rate was set at 0.19 for the neural networks. The neural networks were also allowed to run for 2000 iterations or until convergence to a tss value of 0.04 or if the minimum error was not decremented for 10 continuous iterations, if that occurred before 2000 iterations were completed. Corresponding to the 18 input variables, we selected the neural network to have 18 input units and one output unit for the binary (default/non-default) output. The number of hidden units, in a hidden layer, was chosen as 10 (average of number of input and output units). The neurofuzzy system was allowed to run until convergence or until 2000 iterations. There was no real improvement in classification performance of the neurofuzzy system after 500 iterations. After experimentation, the configuration of the neurofuzzy system was set at 18 input units, 1 output unit and 30 hidden (rule) units.

Although the number of hidden units for the neurofuzzy system were set at 30, only 26 rules were actually generated upon convergence. To illustrate the structure of these rules, only one of these rules are given below due to space considerations. The other 25 rules are similar in structure.

IF (NITA is large) and (NIS is small) and (TDTA is medium) and (CFTD is small) and (LDNW is small) and (CACL is medium) and (QAS is small) and (QACL is small) and (WCS is large) and (CYTD is small) and (ET is medium) and (ST is small) and (CT is medium) and (TLN is small) and (TWS is large) and (TNT is small) and (TNS is medium) and (TFT is medium) THEN non-default.

As can be seen, all 18 variables in the data are included in the generated rules. Although some pruning of these rules could be done, this was not attempted in this study. There is a vast amount of literature that deal with reducing the number of rules through feature selection, verification and validation, among others. Feature selection deals with pre-processing data, before being used as input to neurofuzzy (here) system. This would reduce the number of conjunctions used in the rules generated by these neurofuzzy systems. For an excellent review of recent developments in the feature selection literature, the reader is referred to Kohavi and John (1995) and the references given therein. Whereas feature selection in this context is done before the data is processed through neurofuzzy systems, verification and validation is done after the rules are generated by the neurofuzzy system. The verification and validation process is used to check the knowledgebase for consistency and correctness, non-redundancy, and completeness. Hamilton et al. (1991) provide an overview of verification and validation of knowledge-based systems. Since, the purpose of this study is not to show how to generate compact rule sets using neurofuzzy systems, we do not attempt to do that in this study.

The results using neural networks and neurofuzzy system are given in Tables 3 and 4 respectively. Again, the results using neural networks
outperformed those using neurofuzzy system. Although the classification performance of both neural networks as well as neurofuzzy system are comparable (different only at 0.4 level of significance using the two-tailed t-test) on testing data, the performance of neurofuzzy system on training data was 50%, which is as good as deciding on the class value by tossing a fair coin. Since classification results on testing data are more useful in most real-world situations, these differences are not of major concern.

3.3. Bank failure prediction data

This data was used in Tam and Kiang (1992), among others. Texas banks that failed during 1985–1987 were the primary source of this data. Data from a year and two years prior to their failure were used. Data from 59 failed banks were matched with 59 non-failed banks, which were comparable in terms of asset size, number of branches, age, and charter status. Tam and Kiang had also used holdout samples for both the 1 and 2 year prior cases. The 1 year prior case consists of 44 banks, 22 of which belong to failed and the other 22 to non-failed banks. The 2 year prior case consists of 40 banks, 20 of which belong to failed and the other 20 to non-failed banks. The data describes each of these banks in terms of 19 financial ratios. For a detailed overview of the data set, the reader is referred to Tam and Kiang (1992).

As in the case of Loan Default data, these data sets were already split into training and testing sets. To be consistent with previous studies, we decided to use the same split and did not randomly select training/testing samples from entire data sets. The neural networks were run 10 times, due to the random initial weight settings and the average of the results were taken. The neurofuzzy system was run only once, as before. The learning rate was set at 0.19 for the neural networks. The neural networks were also allowed to run for 2000 iterations or until convergence to a tss value of 0.04 or if the minimum error was not decremented for 10 continuous iterations, if that occurred before 2000 iterations were completed. Corresponding to the 19 input variables, we selected the neural network to have 19 input units and one output unit for the binary output. The number of hidden units, in a hidden layer, was chosen as 10 (average of number of input and output units). Based on experimentation, the configuration of the

<p>| Table 3 Results using neural networks for loan default data |</p>
<table>
<thead>
<tr>
<th>#</th>
<th>Net configuration</th>
<th>Epochs</th>
<th>Time (s)</th>
<th>Classification training (%)</th>
<th>Classification testing (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18-10-1</td>
<td>2000</td>
<td>256</td>
<td>98.87</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>18-10-1</td>
<td>530</td>
<td>67</td>
<td>100</td>
<td>81.25</td>
</tr>
<tr>
<td>3</td>
<td>18-10-1</td>
<td>2000</td>
<td>275</td>
<td>98.87</td>
<td>68.75</td>
</tr>
<tr>
<td>4</td>
<td>18-10-1</td>
<td>2000</td>
<td>258</td>
<td>98.87</td>
<td>68.75</td>
</tr>
<tr>
<td>5</td>
<td>18-10-1</td>
<td>2000</td>
<td>238</td>
<td>98.87</td>
<td>75</td>
</tr>
<tr>
<td>6</td>
<td>18-10-1</td>
<td>2000</td>
<td>241</td>
<td>98.87</td>
<td>75</td>
</tr>
<tr>
<td>7</td>
<td>18-10-1</td>
<td>363</td>
<td>51</td>
<td>100</td>
<td>75</td>
</tr>
<tr>
<td>8</td>
<td>18-10-1</td>
<td>432</td>
<td>58</td>
<td>100</td>
<td>75</td>
</tr>
<tr>
<td>9</td>
<td>18-10-1</td>
<td>551</td>
<td>68</td>
<td>100</td>
<td>75</td>
</tr>
<tr>
<td>10</td>
<td>18-10-1</td>
<td>2000</td>
<td>250</td>
<td>98.87</td>
<td>68.75</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>176.2 (94.64)</td>
<td>98.12 (1.53)</td>
<td>73.75 (3.75)</td>
<td></td>
</tr>
</tbody>
</table>

<p>| Table 4 Results using neurofuzzy system for loan default data |</p>
<table>
<thead>
<tr>
<th>Net configuration</th>
<th>Epochs</th>
<th>Time (s)</th>
<th>Classification training (%)</th>
<th>Classification testing (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>18-30-1</td>
<td>500</td>
<td>130</td>
<td>50</td>
<td>68.75</td>
</tr>
</tbody>
</table>
neurofuzzy unit was set at 19 input units, 1 output
unit and 30 hidden (rule) units for the 1 year prior
data and 40 hidden (rule) units for the 2 year prior
data respectively. The neurofuzzy system was al-
lowed to run until convergence or until 2000 iter-
ations. After 50 iterations, there was no real
improvement in classification performance of the
neurofuzzy systems.

The results using neural networks and neuro-
fuzzy systems are given in Table 5. Again, the
neural networks performed at least as good as the
neurofuzzy systems for both 1 year prior as well as
2 year prior cases, on holdout data. The results on
holdout data, for both 1 and 2 year prior cases,
were not statistically significant using the two-
tailed t-test. Although not of much concern, the
classification performance of neurofuzzy systems
on training data were somewhat poor compared to
those of neural networks. The time taken by the
neurofuzzy systems were an order of magnitude
less for comparable classification results using
testing (holdout) data.

4. Discussion

We studied the performance of neural networks
and neurofuzzy systems using three real-world
credit-risk evaluation data. Although the results
provided in this paper are only preliminary, we
believe that this is a good first step in under-
standing the dynamics of using neural networks
versus neurofuzzy systems for credit-risk evalua-
tion decisions.

This study illustrates the classic tradeoff be-
tween classification performance results and un-
derstandability of results obtained. The learning
results obtained using neurofuzzy systems are un-
derstandable by any user since they are in IF-
THEN rule form. Any decision made by these
neurofuzzy systems can be analyzed using these
rules. Thus these systems have an in-built reason-
ing mechanism. However, in a neural network, all
the user can do is to take the output given by the
neural network as the most appropriate output
without any explicit reasoning. While evaluating
credit of a customer, in real-world situations, the
credit risk evaluator may be required to clarify
why a certain credit approval/denial decision was
made. Under these circumstances, the use of a
neural network for this purpose is questionable.

Overall, neural networks performed somewhat
better than neurofuzzy systems in terms of classi-
fication accuracy, on both training as well as testing
data. This result is not surprising given the various
approximations that are made while dealing with
fuzzification/defuzzification and also the approxi-
mations that are made in fuzzy arithmetic, while
learning the rules in the neurofuzzy system.

Hence, from this study, we understand that
neural networks are probably better for credit-risk
evaluation decisions only if we are not interested in
knowing how a particular conclusion was made.
On the other hand, although the classification
performance of neurofuzzy systems are not good
at the moment, the benefits associated with gen-
erating a set of IF-THEN rules need to be evalu-
ated in light of the fact that the user probably
needs to know the line of reasoning behind a de-
cision.

References

choice and utilization in an experiment on default
prediction. Journal of Accounting Research, Autumn,
325–342.