Credit Risk Evaluation Using A Multilayered Feedforward Neural Network with Backpropagation Learning Rule

Innocent Sizo Duma, ‡ Bhekisipho Twala

Department of Electrical and Electronic Engineering Science
University of Johannesburg, P.O. Box 524, Auctland Park, 2006, South Africa
sizoduma@yahoo.com

Abstract

In this study we propose a multilayered feedforward neural network (MFNN) with Backpropagation Learning Rule Incorporating Bayesian Regularization, and apply it to the credit risk evaluation problem domain using a real world data set from a financial services company in England. We choose the MFNN because of its broad applicability to many problem domains of relevance to business: principally prediction, classification, and modelling. We employ two different methods to determine their prowess in identifying the true positives, that is, defaults. We analyzed the effect of making the number of observed bad equal the number of observed good in the data by over sampling of the minority class (bad obligors) by resampling without replacement, and compare this to the dimensionality reduction of the input vector space using Principal Component Analysis. Overall results indicate that using the Receiver Operating Characteristic as a measure of discriminatory power, over sampling of the minority class has been found to be effective in identifying the true positives.

1. Introduction

With the capital investment increasing in a great many enterprises, credit risk assessment in commercial banks is a crucial area which can have an important impact on stability of bank operation. Therefore discriminating good creditors or obligors from bad ones with high accuracy is of critical importance. With effective classification among clients, the commercial banks can divide customers into different levels by their information, and so the banks can decide whether to provide a loan to clients according to the classification. Of paramount importance is the necessity to maximize the True Positives (TP).

Predictive modeling defaulter risk is one of the important problems in financial risk management, especially for any credit-granting institution such as commercial banks. The need for reliable models that can predict defaults accurately is imperative, in order to enable the interested parties to take either preventive or corrective measures.

Credit risk assessment models based upon statistical techniques usually require strong assumptions about the data, such as normality, and continuous. Moreover, they generally cannot deal efficiently with the implicit nonlinear relations between the attributes and output results. Backpropagation neural networks have shown the ability to learn this nonlinear relationship. However, BPNNs do not usually report good performance in forecasting new examples. For example, overfitting problem widely exists in BPNNs, actually, this algorithm mainly minimize the objective function by using gradient descent method to train network weights, so it results in bad generalized performance. In particular, classification accuracy will be affected to a large extent with minor training samples. It is worth noting that backpropagation is a method for calculating the first derivative, or gradient, of the error function required by some optimization method. Instead of using gradient descent optimization methods to train our network, we envisage that by using cross-validation together with Bayesian Regularization we can overcome the problem of overfitting to a certain extent.

Amongst the objectives of this study is to improve the accuracy, probability that the neural network correctly classifies the obligors, using MFNN on the method presented in [1], which used a backpropagation neural network (BPNN) on the same data set. We envisage that by eliminating outliers and reducing the noise, by using weight decay training (which also account for overfitting), in the data we can achieve a robust neural network which is insensitive to noise, thus improved accuracy. Experimental results has shown that weight decay training is as good as standard backpropagation in noisy situations and, in some data sets, weight decay training outperforms standard backpropagation by a significant difference [2].


2. Methodology Formulation

Given a training data set \( D = \{ x_n, c_n \} (n = 1, \ldots, N) \) where \( x_n \in R^N \) is the nth input pattern and \( c_n \) is the corresponding observed outcome or target, and is a binary variable. In credit risk evaluation models, \( x_n \) denotes the attributes of applicants or creditors; \( c_n \) is the observed result of timely repayment. If the customer defaults its debt, \( c_n = 1 \), else \( c_n = 0 \). The basic assumption we make is that:

\[
Pr(c_n = 1|x) = f(x)
\]  \hspace{2cm} (1)

that is, the probability that an applicant will default depends in some (as yet specified) way on his or her characteristics, \( x \).

This is a two class classification problem, so our neural network model only needs a single output, \( y(x; w) \), where \( w \) is the vector of network weights.

\[
y(x; w) = Pr(c_1|x)
\]  \hspace{2cm} (2)
By definition \( Pr(c_2|x) = 1 - y(x; w) \) is the probability of non-default.

For any \( D = \{x_n, c_n\} (n = 1, \ldots, N) \) the neural network problem consists of finding the optimal combination of weights so that:

\[
|f(x_n; w) - t_n| \to 0
\]  

(3)

where

\[
t_n = \begin{cases} 
1 & \text{when } x_n \text{ is in class } c_1, \\
0 & \text{when } x_n \text{ is in class } c_2.
\end{cases}
\]  

(4)

The likelihood of observing the data is thus given by:

\[
\mathcal{L} = \prod_{n=1}^{N} p(x_n, t_n)
\]  

(5)

Since we are given \( x_n \) and we wish to specify \( t \). We can rewrite equation (5) in terms of the conditional distribution \( p(t|x) \) which gives us complete information regarding \( t \) for a given \( x \), that is

\[
\mathcal{L} = \prod_{n=1}^{N} p(t_n|x_n)p(x_n)
\]  

(6)

Maximizing the likelihood is equivalent to minimizing the negative log-likelihood, this gives us an equivalent error measure:

\[
E = -\sum_{n=1}^{N} \ln p(t_n|x_n) - \sum_{n=1}^{N} \ln p(x_n)
\]  

(7)

Since this is classification problem, we are interested in modeling \( p(t|x) \), and thus the second term on the right hand side \( \ln p(x) \) is independent of any parameter we wish to utilize in our model. Therefore maximizing the model likelihood over parameter values is equivalent to maximizing:

\[
E = -\sum_{n=1}^{N} \ln p(t_n|x_n)
\]  

(8)

We consider the target values \( t_n \) to have been generated from some deterministic generating function \( h(x_n) \) with additive zero-mean Gaussian noise,

\[
t_n = h(x_n) + \epsilon_n
\]  

(9)

with \( \epsilon \sim N(0, \sigma^2) \), where \( \sigma^2 \) is the noise variance which is usually independent of \( x \). Thus

\[
p(\epsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{\epsilon^2}{2\sigma^2} \right\}
\]  

(10)

Assuming that the data was generated by our model, which by combining equations (9) and (10) then gives us the conditional distribution \( p(t|x, w) \)

\[
p(t|x, w) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{[t - y(x; w)]^2}{2\sigma^2} \right\}
\]  

(11)

which is a Gaussian distribution in \( t \) centred at \( y(x; w) \). From equation (8) the corresponding Error function is

\[
E(w) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} (t_n - y(x_n; w))^2 + \frac{N}{2} \ln(2\pi\sigma^2)
\]  

(12)

Dropping out the last term which is independent of the parameter \( w \) and the scaling factor \( \sigma^2 \), gives the sum-of-squares error measure.

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} (t_n - y(x_n; w))^2
\]  

(13)

The average squared error (MSE) per example is then,

\[
E(w) = \frac{1}{N} \sum_{n=1}^{N} (t_n - y(x_n; w))^2
\]  

(14)

The advantages of using equation (14) is that it is a convex, non-negative function which makes it viable in search of the global minimum, it penalizes large errors, and is differentiable.

The NN operates in two modes: training and prediction mode. For the training of the NN and for the prediction using NN we need two data sets, the training set and the set we want to predict (the test set). The training mode begins with arbitrary values of weights (in our study, random numbers) and proceeds iteratively. Each iteration of the complete training set is called an epoch. In each epoch the network adjust the weights in the direction that minimizes equation (14) with respect to \( w \) over all the training examples. After minimizing this function for the training set, new unknown attributes \( x \) are presented to the network and we expect it to classify them accordingly (prediction mode).

Let us assume we have trained our model on a data set of size \( N \) and we now wish to make a prediction for some new example \( x \). The prediction error (PE) for the new example is thus given by:

\[
PE(x) = \left\{ E_D \left[ y(x, w) \right] - \{ t|x, w \} \right\}^2
\]

\[\begin{array}{c}
\quad + E_D \left\{ \left( y(x, w) - E_D \left[ y(x, w) \right] \right)^2 \right\}
\end{array}\]

(15)

where \( E_D[\cdot] \) is expectation with respect to all possible data sets \( D \) that we might use. \( \{ t|x, w \} \) is the conditional average which is the best predictor of the data in the squared-error sense.

The bias and variance terms in equation (15) provide useful information on how the estimates differs from the desired target values. The model bias measures the extent to which the average of the estimates over all possible data sets with the same size differs from the desired target values. The model variance, on the other hand, measures the sensitivity of the estimates to the training data set. Although it is desirable to have both low bias and low variance, we can not reduce both at the same time for a given data set because these goals are conflicting. A model that is less dependent on the data tends to have low variance but high bias if the model is correct. On the other hand, a model that fits the data well tends to have low bias but high variance when applied to different data sets. NN that fits the training data well have low bias, but the potential risk is the overfitting that causes the high variance in generalization. That is, the NN gives a low MSE value for examples from the training data set, but fails for examples from the test set. Severe overfitting can occur with noisy data, even when there are many more training examples than weights. Given a fixed amount of training data, there are some effective approaches to avoiding overfitting, and hence getting good generalization. A more principled solution to overfitting makes us of regularization. Regularization involves modifying the cost function (14) by adding a weight decay penalty.
term to the original cost function. Because small weights can be used by the network to code noisy patterns, this weight decay mechanism is considered to be important in dealing with noise in the data set. The new cost function $E(w)$ then becomes:

$$E(w) = E_0(w) + \frac{1}{2}\mu \sum_{ij} w_{ij}^2 \quad (16)$$

where $\mu$ is the weight decay constant and $w_{ij}$ is the weight connection from node $j$ to node $i$. This additional term favours small values of $w$ and decreases the tendency of a model to overfit fine details of the data [2, 4, 5, 6].

2.1. The Two-Layer Feed-Forward Network

We shall denote the input values to the network by $x_i$ where $i = 1, \ldots, d$. The first layer of the network forms $M$ linear combinations of these inputs to give a set of intermediate variables $a^{(1)}_j$, where

$$a^{(1)}_j = \sum_{i=1}^{d} w_{ij}^{(1)} x_i + b^{(1)}_j \quad (17)$$

$j = 1, \ldots, M$. Each one variable $a^{(1)}_j$ is associated with each hidden unit. Here $w_{ij}^{(1)}$ represents the elements of the first-layer weight matrix and $b^{(1)}_j$ are the bias parameters associated with the hidden units.

The variables $a^{(1)}_j$ are then transformed by the non-linear activation functions of the hidden layer. In our study

$$z_j = \frac{1}{1 + \exp(-a^{(1)}_j)} \quad (18)$$

$j = 1, \ldots, M$ which has the property that

$$\frac{dz_j}{da^{(1)}_j} = \frac{\exp(-a^{(1)}_j)}{(1 + \exp(-a^{(1)}_j))^2} \quad (19)$$

The $z_j$ are then transformed by the second layer of weights and biases to give second-layer activation values $a^{(2)}_k$ with

$$a^{(2)}_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b^{(2)}_k \quad (20)$$

$k = 1, \ldots, \hat{c}$, where $\hat{c}$ is the total number of outputs.

Finally, these values are passed through to the output unit activation function to give output values $y_k$. For classification problems involving multiple independent attributes we consider the logistic sigmoid activation functions applied to each of the outputs independently, so that

$$y_k = \frac{1}{1 + \exp(-a^{(2)}_k)} \quad (21)$$

So, we might redefine our model as:

$$g(x; w) = \frac{1}{1 + \exp[-g(x; w)]}, \quad (22)$$

where $g(x; w)$ is an arbitrary-valued function with poetically infinite range.

While the logistic sigmoid correctly enforces $0 \leq y(x; w) \leq 1$, this particular choice of squashing function may appear arbitrary, that is, we are effectively free to devise any function with the appropriate range e.g. the gaussian error function. We may however obtain further justification for the use of this particular function when predicting posterior probabilities, as we now see.

If we invert the sigmoid we obtain

$$g(x; w) = \log \frac{y(x; w)}{1 - y(x; w)} \quad (23)$$

If we now write

$$g(x; w) = Pr(c_1 | x) \quad (24)$$

From equations (23) and (24) and the fact that $Pr(c_2 | x) = 1 - Pr(c_1 | x)$ we can write:

$$g(x; w) = \log \frac{Pr(x|c_1)}{Pr(x|c_2)} + \log \frac{Pr(c_1)}{Pr(c_2)} \quad (25)$$

equation (25) shows that the neural network is in effect modeling the log likelihood ratio of the two classes.

For neural network training, learning rules are used to update the weights and to minimize the error function. The learning process repeats until termination condition is met. The activation function introduces a degree of nonlinearity to the model and prevents the output from reaching very large values that can paralyze NN model and inhibit training [3, 7].

2.1.1. Backpropagation Algorithm

The goal of a learning algorithm is to minimize $E(w)$ for a particular set of training examples $(x_n, t_n)$.

1. Create a feed-forward network with $n_{in}$ inputs, $n_{h}$ hidden units, and $n_{out}$ output units.

2. Initialize all network weights and biases to small random numbers.

3. Until the termination condition is met, do:

   - For $(x_n, t_n)$ in training example do
   
     (a) Propagate the input forward through the network:
     
     - Input the instance $x_n$ and compute the output of every unit in the network. That is, we evaluate the activations $z_n^m$ of the hidden units and the activations $y_n^k$ of the outputs for each pattern $n$ in the data set.

     (b) Propagate the errors backwards through the network:
     
     - For each network output unit $k$, calculate its error term. Because of the canonical choice of the error function and its corresponding activation function, the partial derivative of the error with respect to $a^{(2)n}_k$, the output activations is the same: $\frac{\partial E}{\partial a^{(2)n}_k} = y^{(2)n}_k - t^{(2)n}_k$. Thus, in order to evaluate the derivatives of $E$ with respect to the weight and bias parameters we first evaluate the quantities $\delta_k^{(2)n}$ given by $\delta_k^{(2)n} = y^{(2)n}_k - t^{(2)n}_k$ which represent the errors for the output units on pattern $n$. Here $y^{(2)n}_k$ denotes $y_k(x^n; w)$. 


- For each hidden unit $h$, calculate its error $\frac{\partial E}{\partial w_k^{(2)}} = \sum_{n=1}^{N} \delta_k^{(2)n} z_j^n$. The derivatives for the output-unit biases are given by $\frac{\partial E}{\partial b_k} = \sum_{n=1}^{N} \delta_k^{(2)n}$. In order to find the corresponding derivatives for the first-layer parameters, the errors $\delta_k^{(2)n}$ must first be back-propagated through the second-layer weights to obtain error signals for the hidden units. The back-propagation equations take the form

$$\delta_j^{(1)n} = (1 - z_j^n)^2 \sum_{k=1}^{c} w_{kj}^{(2)} \delta_k^{(2)n}$$

(26)

where the sum runs over all output units.

- Update each network weight $w_{ji}$ as follows $w_{ji}^{(1)n} \leftarrow w_{ji}^{(1)n} + \eta \delta_j^{(1)n} x_i$, where $\eta$ is the learning rate.

In this study we used the MATLAB learning function LEARNHD (Hebb with decay weight learning rule) to update the weights. In this case the learning rule behaves like a smoothing filter, remembering the most recent inputs more clearly: $w_{ji}^{(1)n} \leftarrow w_{ji}^{(1)n} + \eta \delta_j^{(1)n} x_i$, becomes $w_{ji}^{(1)n} \leftarrow \eta \delta_j^{(1)n} x_i + (1 - \gamma) w_{ji}^{(1)n}$, $\gamma$ is the weight decay rate which is a positive constant less than one. As $\gamma \to 0$, the learning law becomes the standard learning rule. As $\gamma \to 1$, the learning law quickly forgets old inputs and remembers only the most recent patterns. This keeps the weight matrix from growing without bounds. The maximum weight value $w_{j}^{MAX}$ is determined by $\gamma$. The main objective of the Hebb with decay learning is to filter the weight changes so that convergence towards the minima is smooth. In this study we set $\eta = 0.5$ and $\gamma = 0.1$. The above parameters were obtained by trial and error [7, 8].

### 2.1.2. Bayesian Regularization

Bayesian framework proposed by [9] overcome the problem of noisy data, and thus overfitting, and also allows to estimate the number of parameters actually used by the model, the number of weights actually needed to solve a particular problem. Bayesian regularization expands the cost function given by equation (16) to search not only the minimal error, but for the minimal error using the minimal weights. It works by introducing two Bayesian hyperparameters, $\alpha$ and $\beta$, to tell which direction (minimum error or minimal weights) the learning should proceed. The cost function then becomes:

$$C = \beta E_D + \alpha E_W$$

(27)

where $E_D$ is the sum of squared errors, $E_W$ is the sum of squared weights, $\alpha$ is the regularizing constant, and $\beta$ is the second cost function parameter. The relative size of these cost function parameters dictates the emphasis for training: if $\alpha \ll \beta$, then the training algorithm will drive the errors smaller, if $\beta \ll \alpha$, training will emphasize weight size reduction at the expense of the network errors, plus producing a smoother network response. The main objective with implementing regularization is setting the correct values for the cost function. Here are the steps required for Bayesian optimization of the regularization parameters, with the Gauss-Newton approximation to Hessian matrix:

1. Initialize $\alpha$, $\beta$ and the weights.

2. Take one step of the Levenberg-Marquardt algorithm to minimize $C'(w) = \beta E_D + \alpha E_W$.

3. Compute the effective number of parameters

$$\hat{\gamma} = N - 2\alpha tr(J)$$

(28)

making use of the Gauss-Newton approximation to the Hessian:

$$H = 2\beta J^T J + 2\alpha I_N,$$

(29)

where $J$ is the Jacobian matrix of the training set errors.

4. Compute new estimates for the cost function parameters

$$\alpha = \frac{\hat{\gamma}}{2E_w(w)}$$

(30)

and

$$\beta = \frac{n - \hat{\gamma}}{\alpha E_D(w)}$$

(31)

5. Now iterate steps 1 through 3 until convergence.

Where

- $N$ is the total number of parameters in the network
- $\hat{\gamma}$ is a measure of how many parameters in the NN are effectively used in reducing the error function. It can range from zero to $N$.
- $H = \beta \nabla^2 E_D + \alpha \nabla^2 E_W$

Bear in mind that with each reestimate of the cost function parameters the cost function is changing; therefore, the minimum point is moving. If traversing the performance surface generally moves toward the next minimum point, then the new estimates for the cost function parameters will be more precise. Eventually, the precision will be good enough that the cost function will not significantly change in subsequent iterations. Thus, we will obtain convergence [10]. The above procedure is implemented by the MATLAB function TRAINBR. When using TRAINBR, it is important to let the algorithm run until the effective number of parameters has converged. The training will stop with message Maximum MU reached. This is typical, and is a good indication that the algorithm has truly converged.

### 3. Related Work

Various approaches have been used in credit risk modeling which includes amongst others, Structural models and Machine learning algorithms (or Artificial Intelligence - AI). Among intelligence techniques, artificial neural networks (NN) is the most widely used. The NN model has developed out of the fields of artificial intelligence and brain modelling, and contains mathematical and algorithmic elements that mimic the biological neural networks of the human nervous system. The method considers an interrelated group of artificial neurons and processes information associated with them using a so-called connectionist approach, where network units are connected by a flow of information. The structure of NN models changes based upon external or internal information that flows through the network during the learning phase and uses nonlinear function approximation tools to test the relationship between explanatory factors [11].

West [12] investigated the accuracy of linear discriminant analysis (LDA), logistic regression analysis (LRA), the multilayer perceptron (MLP), mixture of experts (MOE), radial basis functions (RBF), learning vector quantization and fuzzy adaptive resonance, k-nearest neighbor (k-NN), classification...
and regression trees (CART) on a German and Austrian credit scoring data sets. Both data sets were provided by the University of Hamburg. The results indicated that NN could improve the credit scoring accuracy. He also suggested that LRA is a good alternative to NN while LDA, k-Nearest Neighbor (k-NN), and CART did not produce encouraging results.

Armingier et al. [13] tested three credit rating models (NN, CART, and LDA) on a data set which was provided by a major bank in Germany. The data sample was found to be biased in the sense that faulty loans were overrepresented, i.e., the percentage of defaults in the sample was about 50 percent. Whereas on average faulty loans in consumer credit varies between 1 and 7 percent. They claimed that, the NN model outperformed both the other models in classifying good credit obligors from bad ones in both validation and test samples.

Desai et al. [14] explored the capabilities of NN, LDA, and LRA on three sets of data. They claimed that NN shows a promise if the performance measure is the percentage of bad loans accurately classified. However, if the performance measure is the percentage of good and bad loans accurately classified, LRA is as good as NN. The percentage of bad loans correctly classified is an important performance measure for credit scoring models since the cost of granting a loan to a defaulter is much larger than that of rejecting a good applicant.

Hand [15] reviewed several credit rating methods, that is LDA, LRA, mathematical programming, recursive partitioning, expert systems, neural networks, smoothing non-parametric, and time varying models. Their assessment concluded that there is no overall best method. What is best will depend on the details of the problem: data structures, the variables used, the extent to which it is possible to separate the classes by using those variables and the objective of the classification. However, NN were found that with proper training are well suited to situations where there is a poor understanding of the data structure.

Twala [16] compared the predictive power of different ensemble of five individual classifiers, the NN, Decision Tress (DT), Naive Bayes Classifier (NBC), k-Nearest Neighbor (k-NN), and Logistic Discrimination (LG) using four credit data set. Overall results showed that at lower levels of noise, the DT achieved the highest accuracy rates as a classifier for handling attribute noise in the training set. For attribute noise in the testing data, the highest accuracy rate was achieved by NBC. The worst performer being the k-NN. Overall results showed that the predictive power of the ensemble of classifiers is much more superior to that of individual classifiers.

Boyacioglu [17] applied various machine learning techniques, viz., NN techniques, support vector machines (SVM) and multivariate statistical methods to bank failure prediction problem in a Turkish case. In the case of NNS, four different architectures viz., multi-layer perceptron (MLP), competitive learning, self-organized map (SOM) and learning vector quantization (LVQ) were employed. The multivariate statistical methods: multivariate discriminant analysis, k-means cluster analysis and logistic regression analysis were tested. The results were evaluated with respect to the correct accuracy performance of techniques. The data set they used was obtained from the annual publication of the Banks Association of Turkey (BAT). The t-test was used as a measure of accuracy. The results showed that the SVM outperformed most all the other techniques employed in the research. Although MLP and LVQ NN architectures are found as the most successful prediction models in the experiments, the prediction performances of SVMs and multivariate statistical methods are also satisfying.

Jardin [18] analyzed the influence of variable selection techniques on model accuracy, particularly the fit of the evaluation criteria commonly used in the financial literature and a NN, to study the relationship that may exist between the structure of a model and its ability to correctly classify failing companies. In his literature review, to explain the research question at hand, he found out that of the 60 authors who’s papers he reviewed, 45 used a backpropagation MLP, whereas the MLP is used in almost all experiments. He finally made a conclusion that a NN based model for predicting bankruptcy performs significantly better when designed with appropriate variable selection techniques than when designed with methods commonly used in the financial literature.

Zhang et al. [19] compared between NN and LR using a five-fold cross-validation method. The NN significantly outperformed the LR with an accuracy of 88% compared with 78.6% for the LR.

Tam [20] compared between MDA, LR, KNN, single-layer, and multilayer network (MLP) in a one-year-ahead bank failure prediction experiment. When they used a leave-one-out procedure instead of hold-out sample, the MLP was the outright winner.

4. Experiments and Analysis

4.1. The Data Set

In this study a real-world credit dataset is used to test the performance of MFNN. The data set is from the financial services company in England, obtainable from accessory CD-ROM of Thomas, Eldeman and Crook [21]. Every applicant includes the following 14 attributes: year of birth, number of children, number of other dependents, is there a home phone, residential status, value of home, mortgage balance outstanding, outgoings on mortgage or rent, outgoings on loans, outgoings on hire purchase, and outgoings on credit cards. The dataset includes detailed information of 1225 applicants, in which including 323 observed bad creditors.

We suspected that the data contained some outliers. To remove the presence of outliers we perform the following:

We calculated the interquartile range $IQR$ for each column as the difference between the first and the third sample quartiles. This gives the range of the middle 50% of the data. It is estimated as follows

$$IQR = \hat{q}_{0.75} - \hat{q}_{0.25} \quad (32)$$

Then we defined the lower limit (LL) as $LL = \hat{q}_{0.25} - 1.5 \cdot IQR$ and an upper limit (UL) as $UL = \hat{q}_{0.75} - 1.5 \cdot IQR$. The idea is that observations that lie outside these limits are possible outliers. Remember, that by definition outliers are data points that lie away from the rest of the data. This means that the data were incorrectly measured or recorded. As we have suspected, on the attribute YEAR OF BIRTH we detected and removed 13 outliers and on the attribute OUTGOINGS ON MORTGAGE OR RENT we detected and removed 27 outliers. All the outliers were from the good obligor examples.

4.2. Design of Neural Network Architecture

Here we have used a $14 - 29 - 1$ feedforward NN, input layer consisting of 14 nodes, single hidden layer containing 29 nodes with logistic sigmoid activation function, the output layer containing a single node with logistic sigmoid activation function. The input layer consists of 14 nodes representing components of the 14 dimensional input attribute vector. Based on the Kol-
mogorov theory, $2 \times 14 + 1$ nodes should be used for one hidden layer.

### 4.3. The Experimental Setup

In this method we use the whole data set to evaluate the classification rate, by means of the confusion matrix. Here we detail stepwise the procedure we have undertaken.

1. The data set $D = \{(x_1, t_1), \ldots, (x_n, t_n)\}$ is presented into the neural network.

2. The network then preprocess the data set (only the attribute vectors) automatically so that the mean is 0 and the standard deviation is 1 using the MATLAB function `mapstd`.

3. Another MATLAB function `divideint` cycles the input data examples between the training set, validation set, and test set according to percentages. That is, 80 percent of the examples are assigned to the training set, 10 percent to the validation set and 10 percent to the test set.

4. Using the neural network output, $Pr(c_k|x_n)$ we then classify the attribute vectors using the Bayes Decision Rule. The threshold on the posterior probability for the decision boundary is 0.5, since there are two classes.

<table>
<thead>
<tr>
<th></th>
<th>Actual Default</th>
<th>Actual Non-Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>Good</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>

Table 1: Contingency Table

5. Perhaps the most basic tool for understanding the performance of a default prediction model is the ‘percentage right’. A more formal way of understanding this measure is to consider the number of predicted defaults(non-defaults) and compare this to the actual number of defaults(non-defaults) experienced. A common means of representing this is a simple contingency table or confusion matrix shown in Table 1.

In the simplest case, the model produces two ratings (Bad/Good). These are shown along with the actual outcomes (default/no default) in tabular form. The cells in the table indicate the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), respectively. A TP is a predicted default that actually occurs, a TN is a predicted non-default that actually occurs. The FP is a predicted default that does not occur and a FN is a predicted non-default where the obligor actually defaults. The errors of the model are FN and FP shown on the off-diagonal. FN represents a Type I error and FP represents Type II error. A ‘perfect’ model would have zeros for both FN and FP cells, and the total number of defaults and non-defaults in the TP and TN cells, respectively, indicating that it perfectly discriminate the defaulters and non-defaulters. The rate at which the confusion matrix correctly classify the observations is given by (22):

$$P(CC) = \frac{TP}{(TP + FN)} \quad (33)$$

### 4.4. Results and Discussion

4.4.1. Results

The networks performance is shown in figure [1] for the original data sample.
Under the same training conditions we over sample the bad obliger examples by resampling without replacement to make the number of observed bad equal the number of observed good, we call this our Method II. Next, also under the same training condition, we reduce the dimensionality of the input attribute vector space using Principal Component Analysis (MATLAB built-in function *processpca*). We eliminate those principal components that contribute less than 1%, of the total variation in the data set. The objective is to observe which method could be implemented that can improve the identification of the true positives in the data: equalizing the number of obligors or reducing the attribute vector space. We call this our Methods III.

We thus used the following evaluation criteria measures:

- **Sensitivity** which is the probability that the MFNN will classify a pattern as a target (bad obligors) when it is really a target. 
  \[
  \text{Sensitivity} = \frac{TP}{TP + FN}
  \]

- **Specificity** which is the probability that the MFNN will correctly classify the true non-target (good obligors) cases. 
  \[
  \text{Specificity} = \frac{TN}{TN + FP}
  \]

- **Negative Predictivity (NP or Type I Accuracy)** is the probability that an obligor is truly bad given that he/she is predicted being bad. 
  \[
  NP = \frac{TN}{TN + FN}
  \]

- **Precision or positive predictivity (Type II Accuracy)** is the probability of predicted non-defaulters being correctly classified. 
  \[
  \text{Precision} = \frac{TP}{TP + FP}
  \]

- **F-Measure** is a measure of the classification accuracy. It considers both the Precision (positive predictivity) and the Sensitivity of the classification test to compute the score: 
  \[
  F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}
  \]

- **Accuracy/Classification rate or Power** is the probability that the MFNN correctly classifies the attributes. 
  \[
  \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
  \]

- **Error Odds Ratio** indicates if the probability of being wrongly classified is highest in the defaults or in the non-defaults group. If the error odds is higher than one the probability is highest in the default group, if the value is lower than one the probability of an incorrect classification is highest in the non-default group. It is defined as: 
  \[
  \frac{\text{Sensitivity}(1 - \text{Specificity})}{(1 - \text{Sensitivity})} \text{Sensitivity}
  \]

- **Discriminant Power or Classification effectiveness**, is a measure of how well the MFNN distinguishes between bad and good obligors. It is the sum of logs of Sensitivity and Specificity over own false proportion, scaled by the standard deviation of the logistic normal distribution curve (i.e. \(\frac{\sigma^2}{2}\)).
  - A classifier with a discriminant value of 1 is not effective in discriminating between bad and good obligors.
  - A classifier with a discriminant value of 3 is effective in discriminating between bad and good obligors.

- **Classification Bias (CB)**: A classification which shows provable and systematic differences in the results of attributes based on class membership. For example, a classification might be considered biased if attributes of one particular class consistently and systematically have statistically different results from the rest of the testing population. It is defined as 
  \[
  CB = \frac{TP}{TP + FP}
  \]

Under the hypothesis that, when a specific objective like what is set in Table 2, the MFNN overestimates the bad examples because there are more good examples than bad; If \(CB > 1\) the MFNN overestimates the bad examples because there are more good examples than bad.

<table>
<thead>
<tr>
<th>Method</th>
<th>True Positives</th>
<th>False Negatives</th>
<th>False Positives</th>
<th>True Negatives</th>
<th>Sensitivity (95% C.L.)</th>
<th>Specificity (95% C.L.)</th>
<th>Precision (95% C.L.)</th>
<th>Accuracy</th>
<th>Error Odds Ratio</th>
<th>Diagnostic Odds Ratio</th>
<th>Discriminant Power</th>
<th>Classification Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method I</td>
<td>88</td>
<td>23</td>
<td>3</td>
<td>4</td>
<td>79.3%[71.7%-86.8%]</td>
<td>57.1%[20.5%-93.8%]</td>
<td>96.7%[93.0%-100%]</td>
<td>87.1%</td>
<td>78.0%</td>
<td>5.1014</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Method II</td>
<td>56</td>
<td>33</td>
<td>15</td>
<td>79</td>
<td>62.9%[52.9%-73%]</td>
<td>84%[76.6%-91.4%]</td>
<td>78.9%[69.4%-88.4%]</td>
<td>70%</td>
<td>73.8%</td>
<td>8.9374</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Method III</td>
<td>91</td>
<td>26</td>
<td>0</td>
<td>1</td>
<td>77.8%[70.2%-85.3%]</td>
<td>100%</td>
<td>100%</td>
<td>85.5%</td>
<td>78%</td>
<td>∞</td>
<td>∞</td>
<td>0.7778</td>
</tr>
</tbody>
</table>

Table 2: Table of results for a simple contingency table, that is, Table 1. The C.I. stands for Confidence Interval

4.4.2. Discussion

The summary of our results based on the above classification criteria measures are shown in Table 2. In general classification accuracy is the generally used model validation criteria. In maximizing the true positives Method III outperforms all the other methods, however it had the the lowest true negative classification performance compared to Method I. This leads us to the hypothesis that, when a specific objective like what is set in this study is the criteria for selection of the best classification method, it may be necessary not to select any particular method based on classification accuracy alone. An interesting observation is that Method II has the highest negative predictivity accuracy, although it has the lowest classification accuracy. This indicates that the attributes are independent in the true negative space. This leads to the conclusion that classification accuracy for true positive and true negative for different methods do not vary proportionately. Also this highlight the fact that different
methods are better at identifying true positives and true negatives as separate spaces.

One of the primary objective(s) of our study was to obtain better results than the work done in [1] which used the same data set and this objective was met. The study in [1] used a three layer backpropagation neural network with 10 TANSIG neurons in the hidden layer and one PURELIN neuron in the output layer. Their network training function is the TRAINLM, with a learning rate set at 0.1 and momentum is set at 0.15. In their study it took 1, 600 epochs to obtain an overall classification rate of 72.24%. We achieved 78% at 1, 000 epochs, see Table 2, quite an improvement in Credit Risk speak. We did not generalize the contingency table using the ROC curve for the purpose of comparison, as the authors in [1] never mentioned the use of the ROC curve.

There are, however some serious issues regarding the error function we are minimizing. Remember that the MSE error measure was derived under the assumption that the binary output \( y \in \{0, 1\} \) with \( y = F(x) + \varepsilon \), where \( \varepsilon \) is the zero mean Gaussian noise, is related to a set of attribute vector \( x \). However, a Gaussian model for \( y \) is clearly not appropriate for classification. Also we should note that \( y \) characterises the possible target values we may observe for a given attribute vector \( x \). In classification when predicting posterior probabilities, the target \( y \) only takes on two values, 0 and 1. Its distribution is clearly not even approximately Gaussian. If we now model \( Pr(c_1|x) \) with \( y(x; \mathbf{w}) \), then we may re-express this more succinctly as:

\[
p(t | \mathbf{x}) = y(\mathbf{x}; \mathbf{w})^t [1 - y(\mathbf{x}; \mathbf{w})]^{1-t}
\]

This is a Bernoulli distribution (which is a special case of the binomial). And is just a shorthand way of writing:

\[
p(t = 1|\mathbf{x}) = y(\mathbf{x}; \mathbf{w})
\]

\[
p(t = 0|\mathbf{x}) = 1 - y(\mathbf{x}; \mathbf{w})
\]

If we assume that our training set comprises independently and identically distributed samples and taking into account that maximizing the model likelihood over parameter values is equivalent to minimizing:

\[
E = - \sum_{n=1}^{N} \ln[p(t_n|x_n)]
\]

\[
E = - \sum_{n=1}^{N} \ln[y(x; \mathbf{w})^t_n (1 - y(x; \mathbf{w}))^{1-t_n}]
\]

\[
E = - \sum_{n=1}^{N} [t_n \cdot \ln(y(x; \mathbf{w})) + (1 - t_n) \cdot \ln(1 - y(x; \mathbf{w}))]
\]

Equation [39] is known as the cross-entropy error function, and is the correct one to use when predicting posterior probabilities in a two-class problem, as minimizing this error maximizes the likelihood of the observations for a realistic model of the conditional target distribution [5].

5. Conclusion

Defaulting modeling is an important feat in credit risk management. The noise, and imbalance in data set pose a great challenge for all researchers. In this research two methods were evaluated and it has been observed that increasing the number of bad obligor examples by over-sampling works best for the data set at hand.

Through the practical data experiment, we have obtained good classification results, a well trained network with superior generalization. We can thus conclude that NN forms a useful tool for the prediction of credit risk in the corporate world. In our next study, we will apply Bayesian neural network techniques to the Credit Risk Evaluation problem domain. The Bayesian techniques come into their own when we are dealing with systems, where as often the case in practise, there is scarcity of data. In such cases the ability to incorporate difficult to quantify prior information, and the possibility of working simultaneously with a number of possible models, become very important tools.

Acknowledgements I would like to thank my supervisor Professor B. Twala, and Professor T. Marwala for their intellectual support. I also like to thank my present colleagues in the Department of Electrical Engineering for their open mindedness, and gratefully acknowledge the financial support from the Faculty of Engineering and Built Environment at the University of Johannesburg.

6. References


