8-2010

Early Stopping of a Neural Network via the Receiver Operating Curve.

Daoping Yu
East Tennessee State University

Early Stopping of a Neural Network via the Receiver Operating Curve

---

A thesis

presented to

the faculty of the Department of Mathematics

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Mathematical Sciences

---

by

Daoping Yu

August 2010

Jeff R. Knisley, Ph.D., Chair

Robert M. Price Jr., Ph.D.

Robert B. Gardner, Ph.D.

Keywords: ANNs, Classifiers, Sampling, ROC, AUC, Early Stopping.

ABSTRACT

Early Stopping of a Neural Network via the Receiver Operating Curve

by

Daoping Yu

This thesis presents the area under the ROC (Receiver Operating Characteristics) curve, or abbreviated AUC, as an alternate measure for evaluating the predictive performance of ANNs (Artificial Neural Networks) classifiers. Conventionally, neural networks are trained to have total error converge to zero which may give rise to over-fitting problems. To ensure that they do not over fit the training data and then fail to generalize well in new data, it appears effective to stop training as early as possible once getting AUC sufficiently large via integrating ROC/AUC analysis into the training process. In order to reduce learning costs involving the imbalanced data set of the uneven class distribution, random sampling and k-means clustering are implemented to draw a smaller subset of representatives from the original training data set. Finally, the confidence interval for the AUC is estimated in a non-parametric approach.

DEDICATION

To my parents Mengli Yu and Aichun Yu

ACKNOWLEDGEMENTS

I gratefully acknowledge my advisor Professor Jeff Knisley who has supervised the thesis after introducing me into the appealing field of predictive modeling. I thank Professor Robert Price for his kind support and inspiring discussions on the tightness of the estimated confidence interval for the AUC, possible feature selection, and suggestions for insurers marketing team. I appreciate Professor Robert Gardner who has provided the thesis template in LaTex and kindly corrected errors in my thesis. I would like to thank Teshome, Pan, and Eric with whom I have discussed and shared a lot such as how to include graphics in a LaTex file after properly exporting figures from Maple worksheets. I wish to acknowledge nice people standing by me in times of need, and for the very nice atmosphere for study and work in the Department of Mathematics and Statistics and the Graduate School at East Tennessee State University.

CONTENTS

ABSTRACT 2

DEDICATION 4

ACKNOWLEDGMENTS 5

LIST OF TABLES 7

LIST OF FIGURES 8

1 INTRODUCTION 9

1.1 ANNs in Actuarial Science 10

2 NEURAL NETWORKS AS BINARY CLASSIFIERS 12

2.1 Back-Propagation Multi-Layer Feed-Forward Neural Networks 13

2.2 Advantages and Limitations of ANNs 19

3 ROC ANALYSIS 21

3.1 ROC Curves 21

3.2 Confidence Interval Estimation for the AUC 26

4 IMBALANCED DATA PRE-PROCESSING 29

4.1 Random Sampling and k-means Clustering 29

5 EXPERIMENTAL RESULTS AND DISCUSSIONS 32

5.1 Experiments and Results 33

5.2 Discussion 38

BIBLIOGRAPHY 40

APPENDIX: Maple Code 46

VITA 64

LIST OF TABLES

1 Main Advantages and Limitations of Neural Networks 19

2 Test Performance on Average 34

3 Train 2 Iterations With the Original Training Set 37

LIST OF FIGURES

1 The Sigmoid Function 13

2 An Artificial Neuron 14

3 A Three-Layer Feed-Forward Neural Network 14

4 2×2 Confusion Matrix 22

5 ROC Curves, Best Threshold Closest to the Top Left Corner 24

6 Training Error and ROC Curve 33

1 INTRODUCTION

Who will take direct mail from insurers more seriously to advertise their products or services and show more interest in a particular product or service? The marketing team would effectively decrease expenses if they could more accurately identify who potential customers are and select to whom to mail designed marketing bulletins. Actuaries hired by insurers can assist in detecting potential customers by taking advantage of predictive modeling to analyze factual data which can lead to improved marketing decisions. Predictive modeling encompasses a variety of techniques from mathematics, statistics and machine learning, which is widely used in business for decision making. It aims to extract subtle informative patterns by capturing relationships between explanatory variables and response variables from historical and current data to make predictions about future events.

For predictive modeling, various classifiers such as Naive Bayes [31, 37], k-nearest neighbors [32], Support Vector Machines (SVMs) [6], logistic regression [13], *Artificial Neural Networks* (ANNs) [10] are all employable. This thesis focuses on ANNs as binary classifiers for prediction of customers' propensity to have a caravan insurance policy. ANNs, biologically inspired and incorporating neuroscience advances, are non-linear models that are able to approximate sophisticated functions. ANNs can be applied to problems of classification, prediction and control in a wide spectrum of fields such as actuarial science, neuroscience, bioinformatics, and engineering. It is appropriate to employ ANNs when there is little information available about the probability distribution of a population or how response variables are associated with

explanatory variables [17].

In particular, we apply a back-propagation learning algorithm to train multilayer feed-forward (MLF) neural networks and incorporate analysis of the *Receiver Operating Characteristics* (ROC) curve into the training process for the early stopping purpose. Because the real data usually has an uneven two-class distribution as potential customers just occupy a small proportion of the population, it may be necessary to pre-process the data to achieve a certain degree of balance between two distinct classes.

1.1 ANNs in Actuarial Science

ANNs classifiers have a wide range of applications to analyze insurance data in such settings as underwriting [18], rate making [30], loss reserving [26], and marketing to make some sort of classification and prediction. Risk classification is fundamental to insurance business, of vital importance in properly identifying and segmenting risk groups for correct underwriting, and appropriate rating to counter effects of adverse selection [1]. Also comparison studies between ANNs and *Generalized Linear Models* (GLMs, widely used by actuaries) could be conducted from a theoretical perspective and with practical examples based on real insurance data [23].

The work in this thesis falls into the realm of insurance marketing prediction. Marketing analyzes customers to predict response rates (sales volume) and thus profitability of the marketing activity. Actuaries could play an active role in the marketing process helping marketers with response prediction to maximize the return per dollar

invested in acquiring customers.

2 NEURAL NETWORKS AS BINARY CLASSIFIERS

Given a set of training patterns $\{\langle \mathbf{p}^i, \mathbf{t}^i \rangle, i = 1, ..., N\}$ of input patterns $\mathbf{p}^i = (p_1^i, p_2^i, ..., p_r^i)$ and target outputs $\mathbf{t}^i = (t_1^i, t_2^i, ..., t_n^i)$, machine learning algorithms learn some initially unknown function

$$f : P \rightarrow T$$

such that

$$\mathbf{t}^i = f\left(\mathbf{p}^i\right), \tag{1}$$

where f is a classifier mapping from a discrete or continuous r-dimensional feature space P to a discrete set of labels T, $\mathbf{p}^i \in P$, $\mathbf{t}^i \in T$.

From the mathematical point of view, a classifier partitions feature space P so that each subset in the partition contains points corresponding to only one label in T. Training corresponds to using a set of training patterns with a priori classified functional relationship to approximate such a partition. Classification corresponds to using the approximate partition to make predictions about a pattern whose label is not known [19]. If there are only two labels, as dealt with in this thesis where consumers are classified as either interested or not interested in purchasing a caravan insurance policy, the goal is to establish a binary classifier to partition a r-dimensional space into two distinct subsets.

2.1 Back-Propagation Multi-Layer Feed-Forward Neural Networks

Prior to the introduction of basic concepts of neural networks, let us begin with logistic regression which is a useful way of describing the relationship between multiple explanatory variables and a binary response variable. It is a generalized linear model (GLM) used for prediction of the probability of occurrence of an event by fitting data to a logistic function. The logistic function is sigmoid, and the *sigmoid function* is a real function $\sigma\colon \mathbb{R} \to (0,1)$, defined by the expression

$$\sigma(x) = \frac{1}{1+e^{-\kappa x}} \tag{2}$$

where $x = -b + w_1 x_1 + \ldots + w_r x_r$, and the parameter $\kappa > 0$ can be selected arbitrarily and sigmoid means differentiable and non-decreasing from 0 up to 1. The shape of the sigmoid changes according to the value of κ, as can be seen in Figure 1.

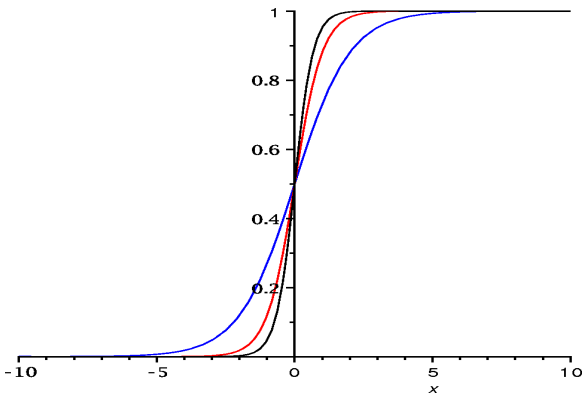

Figure 1: The Sigmoid Function

The sigmoid function is useful because it can take as an input x any value from $-\infty$ to ∞, whereas the output $\sigma(x)$ is confined to values between 0 and 1, which

matches up with the range of probability of an event. A logistic regression classifier can be interpreted as one single artificial neuron with the logistic function as the *activation function*, $w_1, ..., w_r$ as *synaptic weights* and b as a *bias*, which is shown in Figure 2.

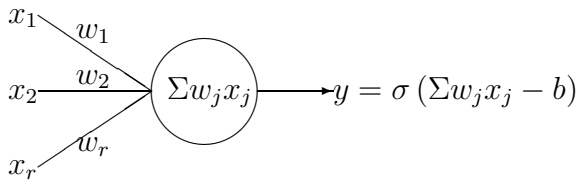

Figure 2: An Artificial Neuron

A *multi-layer feed-forward neural network* (MLF) consists of multiple layers of artificial neurons, and each neuron in one layer has connections to all the neurons in the previous and subsequent layers but not to any other neuron in the layer itself. We will consider networks with a single *hidden layer* (intermediate layer), the *input layer* (the first layer) and the *output layer* (the last layer) as shown in Figure 3.

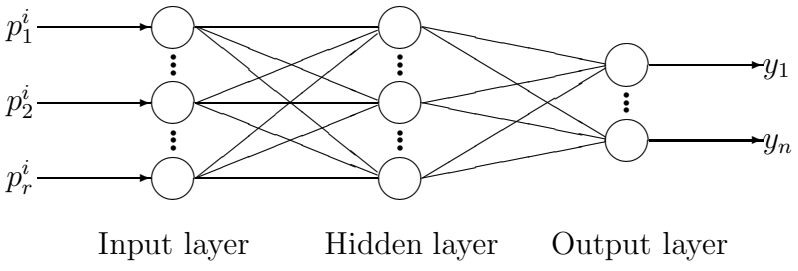

Figure 3: A Three-Layer Feed-Forward Neural Network

In the feed-forward neural network, the information moves in only one direction, *forward*, from the input neurons, through the hidden neurons and to the output neurons, without loops in the network.

Suppose that a three-layer feed-forward neural network has r input neurons connected to m hidden layer neurons that are connected to n neurons in the output layer. In the *feed-forward* phase, the ith input patterns $p_1^i, p_2^i, ..., p_r^i$ are presented to the input layer neurons, and their activations are calculated as

$$x_l = \sigma\left(p_l^i - \theta_l\right), \tag{3}$$

for $l = 1, ..., r$ and where θ_l denotes the bias for the lth input neuron.

Those input neurons activations are multiplied by the synaptic weights w_{kl} between the lth input neuron and the kth hidden neuron and used to calculate the activations of the hidden layer neurons as

$$h_k = \sigma\left(\sum_{l=1}^{r} w_{kl}x_l - \theta_k\right), \tag{4}$$

for $k = 1, ..., m$ and where θ_k denotes the bias for the kth hidden neuron.

Similarly, those hidden neurons activations are multiplied by the synaptic weights α_{jk} between the kth hidden neuron and the jth output neuron and are used to calculate the activations from the output layer neurons as

$$\begin{aligned} y_j &= \sigma\left(\sum_{k=1}^{m} \alpha_{jk}h_k - b_j\right) \\ &= \sigma\left(\sum_{k=1}^{m} \alpha_{jk}\sigma\left(\sum_{l=1}^{r} w_{kl}x_l - \theta_k\right) - b_j\right), \end{aligned} \tag{5}$$

for $j = 1, ..., n$ and where b_j denotes the bias for the jth output neuron.

The produced output y_j is different in general from the target t_j. What we want is to make y_j and t_j identical for $j = 1, ..., n$, by using a learning algorithm. More precisely, we want to minimize the total squared error function of a neural network in the weight space for a given training set, defined as

$$E\left(\mathbf{w}_1, ..., \mathbf{w}_k, ..., \mathbf{w}_m; \boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_j, ..., \boldsymbol{\alpha}_n\right) = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - t_j\right)^2, \tag{6}$$

where $\mathbf{w}_k = (w_{k1}, ..., w_{kl}, ..., w_{kr})$ denotes the vector of weights between the input layer and the kth-hidden neuron, and $\boldsymbol{\alpha}_j = (\alpha_{j1}, ..., \alpha_{jk}, ..., \alpha_{jm})$ denotes the vector of weights between the hidden layer and the jth-output neuron.

We want to train a three-layer feed-forward neural network until we have closely approximated

$$\frac{\partial E}{\partial w_{kl}} = 0, \tag{7}$$

and

$$\frac{\partial E}{\partial \alpha_{jk}} = 0 \tag{8}$$

at the inputs $\mathbf{p}^i$ for all $l = 1, ..., r$; $k = 1, ..., m$; and $j = 1, ..., n$. Because these equations cannot be solved directly, a *back-propagation algorithm* with an iterative process of gradient descent is used instead to find appropriate synaptic weights.

Because this numerical optimization method requires computation of the gradient of the error function with respect to synaptic weights at each iteration step, one must guarantee the differentiability (differentiability implies continuity) of the error function. Because a neural network itself computes only function compositions, in order to have the error function differentiable, one has to use a differentiable activation function. A popular and simple one is the sigmoid function defined in the equation

(2). The derivative of the sigmoid with respect to x, needed later on in this section, is

$$\begin{aligned} \frac{d\sigma}{dx} &= \frac{\kappa e^{-\kappa x}}{(1+e^{-\kappa x})^2} \\ &= \kappa\sigma(x)(1-\sigma(x)). \end{aligned} \tag{9}$$

Thus, E is a continuous and differentiable function of the weights w_{kl}, α_{jk} in a neural network. One can thus minimize E by using an iterative process of gradient descent, for which one needs to calculate the gradient

$$\nabla E = \left(..., \frac{\partial E}{\partial w_{kl}}, ..., \frac{\partial E}{\partial \alpha_{jk}}, ... \right). \tag{10}$$

Each weight is updated using the increment

$$\Delta w_{kl} = -\lambda \frac{\partial E}{\partial w_{kl}} \tag{11}$$

and

$$\Delta \alpha_{jk} = -\lambda \frac{\partial E}{\partial \alpha_{jk}}, \tag{12}$$

where λ is a parameter called the *learning rate* which defines the step length of each iteration in the negative gradient direction.

The derivation of $\frac{\partial E}{\partial \alpha_{jk}}$ is given by

$$\begin{aligned} \frac{\partial E}{\partial \alpha_{jk}} &= \frac{\partial \left(\frac{1}{2} \sum_{j=1}^{n} (y_j - t_j)^2 \right)}{\partial \alpha_{jk}} \\ &= (y_j - t_j) \frac{\partial y_j}{\partial \alpha_{jk}} \\ &= (y_j - t_j) \frac{\partial \sigma \left(\sum_{k=1}^{m} \alpha_{jk} h_k - b_j \right)}{\partial \alpha_{jk}} \\ &= (y_j - t_j) \kappa y_j (1 - y_j) h_k \end{aligned} \tag{13}$$

The weights α_{jk} are adjusted using

$$\alpha_{jk} \leftarrow \alpha_{jk} + \lambda \delta_j h_k,$$

where h_k is given in the equation (4), and

$$\delta_j = \kappa y_j \left(1 - y_j\right) \left(t_j - y_j\right). \tag{14}$$

We show the derivation of $\frac{\partial E}{\partial w_{kl}}$ below.

$$\begin{aligned}
\frac{\partial E}{\partial w_{kl}} &= \frac{\partial \left(\frac{1}{2} \sum_{j=1}^{n} \left(y_j - t_j\right)^2 \right)}{\partial w_{kl}} \\
&= \sum_{j=1}^{n} \left(y_j - t_j\right) \frac{\partial y_j}{\partial w_{kl}} \\
&= \sum_{j=1}^{n} \left(y_j - t_j\right) \frac{\partial \sigma \left(\sum_{k=1}^{m} \alpha_{jk} h_k - b_j \right)}{\partial w_{kl}} \\
&= \sum_{j=1}^{n} \left(y_j - t_j\right) \kappa y_j \left(1 - y_j\right) \frac{\partial \sum_{k=1}^{m} \alpha_{jk} h_k - b_j}{\partial w_{kl}} \\
&= \sum_{j=1}^{n} \left(y_j - t_j\right) \kappa y_j \left(1 - y_j\right) \alpha_{jk} \frac{\partial h_k}{\partial w_{kl}} \\
&= \sum_{j=1}^{n} \left(y_j - t_j\right) \kappa y_j \left(1 - y_j\right) \alpha_{jk} \frac{\partial \sigma \left(\sum_{l=1}^{r} w_{kl} x_l - \theta_k \right)}{\partial w_{kl}} \\
&= \sum_{j=1}^{n} \left(y_j - t_j\right) \kappa y_j \left(1 - y_j\right) \alpha_{jk} \kappa h_k \left(1 - h_k\right) x_l
\end{aligned} \tag{15}$$

The weights w_{kl} are adjusted using

$$w_{kl} \leftarrow w_{kl} + \lambda \rho_k x_l,$$

where x_l is given in the equation (3), and

$$\rho_k = \kappa h_k \left(1 - h_k\right) \sum_{j=1}^{n} \alpha_{jk} \delta_j. \tag{16}$$

with h_k given in the equation (4), and δ_j given in the equation (14).

2.2 Advantages and Limitations of ANNs

Consider strengths and weaknesses of ANNs. One nice property of MLF neural networks is that they are known as universal classifiers according to the *Cybenko's Theorem* [9] that says *choosing a hidden layer sufficiently large with the appropriate selection of synaptic weights, there exists a neural network to uniformly approximate any absolutely integrable function f on a compact set in $\mathbb{R}^n$ to within any $\epsilon > 0$.*

Therefore, in practice the number of hidden layer neurons may necessarily be large, thus contradicting the desire to use smaller hidden layers to better avoid *over-training* and *over-fitting* problems [19]. There are several other typical problems of neural networks including the speed of convergence, and the possibility of getting stuck in a *local minimum* of the error function.

Advantages	Limitations
High accuracy	Poor transparency
Universality	Undesirable Local Minima
Noise tolerance	Over-fitting

Table 1: Main Advantages and Limitations of Neural Networks

The back-propagation algorithm will eventually converge, albeit it may converge slowly or converge to local minima. The learning rate is a significant factor to determine the speed of convergence. Setting a too small learning rate may require too many iterations until convergence, but setting a too large learning rate may end up with oscillation around a minimum. One typical way to deal with undesirable local minima is to randomly reinitialize the weights to different starting values, which approach ends up with a set of neural networks and then the one with the best per-

formance will be chosen. Even if a neural network just converges to local minima, if it has good performance on the training set and can be successfully validated it will often still generalize well on new data.

ANNs can be over-trained to the training set. Over-training corresponds to the "memorizing"of the training set, thus leading to poor recognition for noisy patterns not in the training set. This issue is often addressed using the cross-validation training method in which a part of the original training set is randomly removed as the validation set, and a neural network is trained with the remaining training patterns, then the classifications of the removed patterns are predicted. When the performance on both the training set and the validation set becomes good enough, the network training could stop.

Similar to other non-linear models, ANNs can over-fit the training data. Typically, there are small variations in the values within each feature, so that if there are too many parameters (for example, too many neurons in the hidden layer) then training may lead to an "interpolation" of the slightly flawed training set at the expense of poor generalization of the training set and predictions become meaningless [19].

3 ROC ANALYSIS

ROC analysis provides a decision tool to select possibly optimal models, originally developed for analysis of radar images where complex and weak signals are to be distinguished from a noisy background. Often the ROC analysis is used to find an optimal threshold value for decision making, which is widely used in medicine [4], radiology [29], bioinformatics [35], and machine learning research [34].

Recently, in both theoretical and empirical studies, the area under the ROC curve (AUC) has been suggested [5] as the alternative metric of classifier performance, and many existing learning algorithms have been modified in order to seek a classifier with maximum AUC. It has been shown that the AUC is a better (statistically consistent and more discriminating) measure than accuracy in the test of learning algorithms [22].

We provide a mathematical introduction to ROC analysis. Important concepts involved in the correct use and interpretation of this analysis, such as generation of ROC curves, estimation of the AUC in parametric, semi-parametric and non-parametric methods, and the *confidence interval* estimation of the AUC are discussed.

3.1 ROC Curves

Among available binary classifier performance measures, overall accuracy

$$\frac{TP + TN}{TotalCounts}$$

and mis-classification rate

$$\frac{FP + FN}{TotalCounts},$$

TPR (true positive rate)

$$\frac{TP}{TP + FN}$$

and FPR (false positive rate)

$$\frac{FP}{FP + TN}$$

can be empirically calculated from a 2×2 confusion matrix as shown in Figure 4.

		actual value **P**	actual value **N**
prediction outcome	$\mathbf{P}'$	True Positive	False Positive
	$\mathbf{N}'$	False Negative	True Negative

Figure 4: 2×2 Confusion Matrix

Such measures require that all predicted cases be divided into true positives (actually positive and correctly classified as such), false negatives (actually positive but erroneously classified as negative), true negatives (actually negative and correctly classified as such), and false positives (actually negative but erroneously classified as positive).

Assuming that we have only two classes, labeled P (actually positive) and N (actually negative). Let Y be the prediction outputs of a binary classifier with cumulative

distribution functions G for actual positives and F for actual negatives, respectively. For a particular threshold value t, the true positive rate (TPR, or sensitivity) and the false positive rate (FPR, or 1−specificity) of the classifier are defined as

$$TPR = P\left(Y > t \,|P\right) = 1 - G\left(t\right), \tag{17}$$

and

$$FPR = P\left(Y > t \,|N\right) = 1 - F\left(t\right) \tag{18}$$

respectively.

The ROC curve for a binary classification problem, which is defined as a plot of the true positive rate as the vertical coordinate versus the false positive rate as the horizontal coordinate for all possible threshold levels. The ROC curve can be mathematically expressed by

$$R\left(x\right) = 1 - G\left(F^{-1}\left(1 - x\right)\right), \tag{19}$$

where F^{-1} is the inverse function of F. $R\left(x\right)$ is indeed the TPR of the classifier when the FPR is at level x. The area under the ROC curve (AUC), defined as

$$A = \int_0^1 R\left(x\right) dx, \tag{20}$$

is a single-number measure of the overall performance across the entire range of all possible threshold levels.

A single pair of TPR and FPR values, corresponding to selecting a single threshold for classification, is insufficient to describe the full spectrum of performance. To overcome that insufficiency, in practice we scan over the entire range of threshold levels. Each specific pair of TPR and FPR values for a particular threshold level

corresponds to a discrete point on the graph called an *operating point*. An empirical ROC curve can be estimated or constructed from these discrete points, either by making the assumption that the prediction results follow an approximating distribution, or by connecting all points.

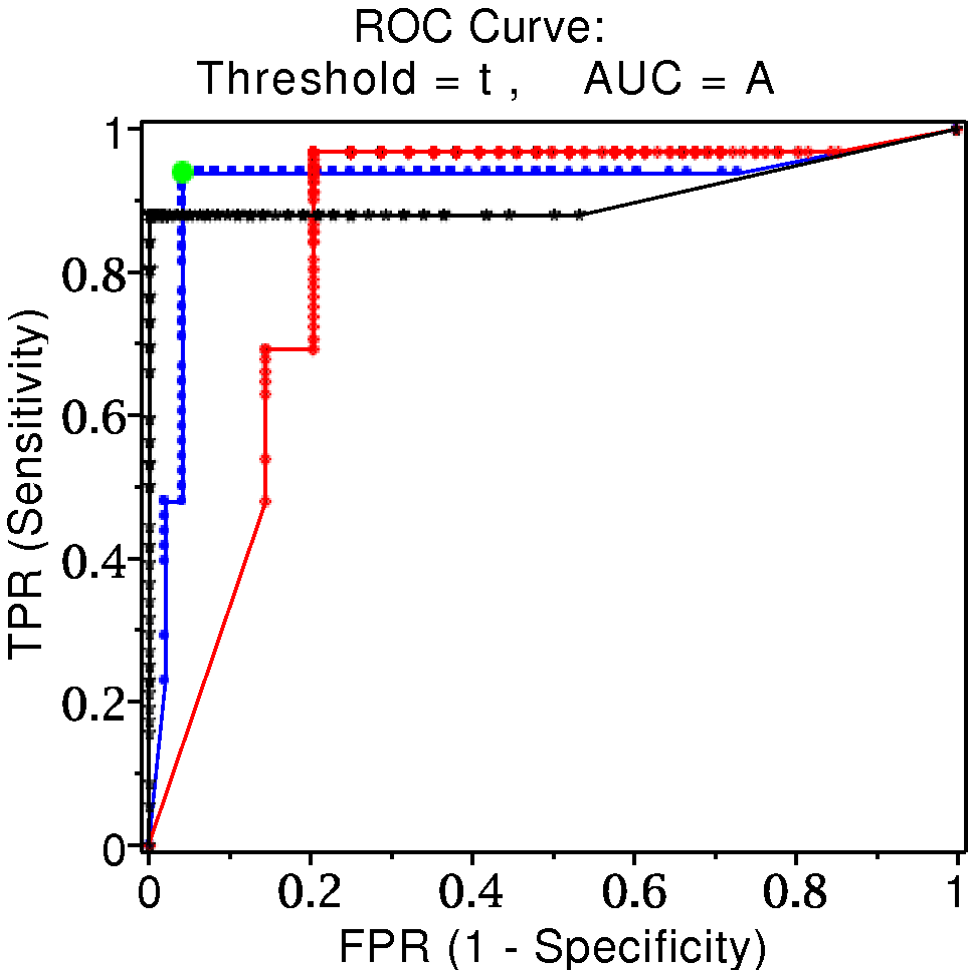

Figure 5: ROC Curves, Best Threshold Closest to the Top Left Corner

ROC curves begin from the bottom-left corner (threshold set at 1) and rise to the top-right corner (threshold set at 0), and the closer to the top-left corner, the larger

area under the ROC curve, and the better classification capability. The practical lower bound for the AUC is 0.5 (random guessing performance with the ROC curve along the diagonal line), and the ideal upper bound for the AUC is 1 (perfect classification performance).

Many approaches have been proposed to do inference about the ROC curve, including parametric [25], non-parametric [14] and semi-parametric [11] approaches. The AUC can be estimated both parametrically, fitting prediction outcomes to a model with maximum likelihood estimates (MLE) with the assumption that either prediction outcomes themselves or some transformation of prediction outcomes follows certain (for example, bi-normal) distribution, and non-parametrically from the empirical ROC curve with no or weak assumptions made about the distributions of prediction outcomes. Semi-parametric [11] approaches are also available. It has been suggested that for a wide range of distributions, concern about bias or imprecision of the estimates of the AUC should not be a major factor in choosing between the non-parametric and parametric approaches [15].

There are different non-parametric methods for an estimate of the AUC. One is to use the Wilcoxon-Mann-Whitney statistic [16] that is a distribution-independent statistic. This method does neither necessarily require actually graphing the ROC curve nor make assumptions on the distributions of outcomes. Let $y_1^+, ..., y_m^+$ be the outputs of a classifier on the actual positives and $y_1^-, ..., y_n^-$ its outputs on the actual negatives. Then, the AUC of that classifier is given by

$$A = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \left(I_{y_i^+ > y_j^-} + \frac{1}{2} I_{y_i^+ = y_j^-} \right)}{mn}, \tag{21}$$

that is the value of the Wilcoxon-Mann-Whitney statistic [16]. Another is the sum-

mation of the areas of the trapezoids formed by connecting all the operating points on the ROC curve. The latter method is used in this thesis.

3.2 Confidence Interval Estimation for the AUC

The AUC is often presented along with its confidence interval (CI). If one performs the same classifier on a different sample, the AUC obtained may be different. Parametric methods based on the bi-normal model were proposed [25] to estimate confidence intervals for the AUC. Empirical likelihood (EL) based approach [28, 33] were proposed to derive non-parametric confidence intervals for the AUC and compared with normal approximation based intervals and bootstrap intervals for the AUC. Cortes et al. [8] derived relatively tight confidence intervals for the AUC based on a statistical and combinatorial analysis using only a small number of readily available parameters such as the mis-classification number k_0, the number of positives m and the number of negatives n.

Assume that all classifications with fixed k mis-classifications are equally probable. For a given classification, there may be x, $0 \leq x \leq k$, false positives. Then, the expectation of the AUC, A, over all classifications with k mis-classifications is given by

$$\mathbf{E}\left[A_k\right] = 1 - \frac{k}{m+n} - \frac{(m-n)^2\,(m+n+1)}{4mn}\left(\frac{k}{m+n} - \frac{\sum_{x=0}^{k-1}\binom{m+n}{x}}{\sum_{x=0}^{k}\binom{m+n+1}{x}}\right) \qquad (22)$$

and the variance of the AUC is given by

$$\sigma^2(A_k) = \frac{kQ_0}{144m^2n^2} - \frac{(m+n+1)^2(m-n)^4 Z_1^2}{16m^2n^2} - \frac{(m+n+1)Q_1Z_1}{72m^2n^2} + \frac{(m+n+1)(m+n)T(m^2-nm+3km-5m+2k^2-nk+12-9k)Z_2}{48m^2n^2} + \frac{(m+n+1)(m+n)(m+n-1)T((m+n-2)Z_4-(2m-n+3k-10)Z_3)}{72m^2n^2} \quad (23)$$

with

$$Z_i = \frac{\sum_{x=0}^{k-i}\binom{m+n+1-i}{x}}{\sum_{x=0}^{k}\binom{m+n+1}{x}}, \quad (24)$$

$$T = 3\left((m-n)^2+m+n\right)+2, \quad (25)$$

$$Q_0 = \left(-3m^2+7m+10n+3nm+10\right)T - 4(3mn+m+n+1) + \left(\left(-3n^2+3mn+3m+1\right)T - 12(3mn+m+n) - 8\right)k + (m+n+1)Tk^2, \quad (26)$$

$$Q_1 = \left(-3m^2+7(m+n)+3mn\right)T - 2(6mn+m+n) + \left(\left(-3n^2+3mn-3m+8\right)T - 6(6mn+m+n)\right)k + +3(m-1)Tk^2 + Tk^3. \quad (27)$$

$\mathbf{E}[A_k]$ and $\sigma(A_k)$ depend only on k, m, and n [8].

Assume that a binary classifier follows a binomial law (use the normal approximation of the binomial law when $m+n$ is large), then, for any ϵ, $0 \leq \epsilon \leq 1$, one can construct I_A, the confidence interval for the AUC at the confidence level $1-\epsilon$ defined by

$$I_A = \left[\min_{k\in I_k}\left\{\mathbf{E}[A_k] - \frac{\sigma(A_k)}{\sqrt{\epsilon_k}}\right\}, \max_{k\in I_k}\left\{\mathbf{E}[A_k] + \frac{\sigma(A_k)}{\sqrt{\epsilon_k}}\right\}\right], \quad (28)$$

where

$$\epsilon_k = 1 - \sqrt{1-\epsilon}, \quad (29)$$

and

$$I_k = \left[k_0 - \frac{\sqrt{N}\Phi^{-1}(\frac{1-\sqrt{1-\epsilon}}{2})}{\sqrt{2}}, k_0 + \frac{\sqrt{N}\Phi^{-1}(\frac{1-\sqrt{1-\epsilon}}{2})}{\sqrt{2}} \right] \tag{30}$$

is the confidence interval for the mis-classifications number k at the confidence level $\sqrt{1-\epsilon}$ with

$$\Phi(u) = \int_u^{\infty} \frac{e^{\frac{-x^2}{2}}}{\sqrt{2\Pi}} dx. \tag{31}$$

I_A depends only on ϵ, k_0, m, and n [8].

4 IMBALANCED DATA PRE-PROCESSING

Information about customers consists of the target variable (caravan policy holders or not) and 85 explanatory variables describing customer attributes including product usage data and socio-demographic data derived from postal zip codes. The training set contains 5822 customers, with values of target variable known. A test set contains 4000 customers with values of target variable to be determined (actual values removed and stored somewhere else) [36].

Notice that in the training set there are only 348 customers who have a caravan insurance policy, a very small proportion just around 6%, similarly to the test set. A classifier that predicts that "no one will buy" has about 94% classification accuracy but is useless for selecting potential customers. In such case, the class imbalance problem occurs, and standard classifiers tend to be overwhelmed by the majority class and treat the minority class as noise. Class imbalance problems have been well-studied with many approaches such as over-sampling [2, 3, 21], down-sampling [2, 21], uncertainty sampling [20], informative sampling [12, 24], and pre-clustering [27].

4.1 Random Sampling and k-means Clustering

For the reason of lowering the cost involving training on a highly imbalanced data set, one could conduct either random sampling from the majority class of the population or k-means clustering to adjust to the size of the minority class. Simple random sampling (SRS) is a process of drawing simple random samples from a population

such that all samples of the same size have an equal chance of being selected from the population. In statistics and machine learning, k-means clustering is an algorithm employing an iterative refinement approach to partition a given data set into k clusters in which each data point belongs to the cluster with the nearest mean, also referred to as Lloyd's algorithm.

Given a set of data points $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$, where each data point is a d-dimensional real vector, the k-means clustering aims to partition the set of n data points into k clusters $(k < n)$ $\mathbf{C} = \{C_1, C_2, ..., C_k\}$ so as to achieve

$$\arg\min_{\mathbf{C}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of all data points in cluster C_i.

Given an initial set of k means $\mu_1^{(1)}, ..., \mu_k^{(1)}$ as the initial k cluster representatives (centroids), which may be selected by random sampling from the data set, the algorithm proceeds by alternating between two steps till convergence:

Assignment step: Associate each data point $\mathbf{x}_j$ with its closest (the default measure of closeness is the Euclidean distance) centroid μ_i resulting in a partitioning of the data set.

$$C_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - \mu_i^{(t)} \right\| \leq \left\| \mathbf{x}_j - \mu_{i^*}^{(t)} \right\| \text{ for all } i^* = 1, \ldots, k \right\}$$

Relocation step: The centroid for each cluster is relocated to the mean of all data points assigned to it.

$$\mu_i^{(t+1)} \leftarrow \frac{1}{|C_i^{(t)}|} \sum_{\mathbf{x}_j \in C_i^{(t)}} \mathbf{x}_j$$

When the assignments no longer change, the algorithm converges and convergence is guaranteed in a finite number of iterations. Note that each iteration needs $n \times k$

comparisons, which determines the time complexity of one iteration. The number of iterations required for convergence varies and may depend on n and k. With regard to the computational intensity and the convergence speed of k-means clustering implemented in this thesis, we set tolerable sufficiently small shift of centroids as the stopping criterion rather than until no changes would be made within each cluster.

There is no guarantee that k means algorithm will converge to the global optimum, instead, only to a local optimum. The local optima problem can be countered to some extent by running the algorithm multiple times with different initial centroids. Also, k-means clustering algorithm is sensitive to the presence of outliers, and *mean* is not a robust statistic.

5 EXPERIMENTAL RESULTS AND DISCUSSIONS

Divide the data into two groups (positives/negatives), and rescale the data by normalization before clustering or random sampling. Design a neural network structure with 85 input neurons (one for each attribute in the input pattern), and 2 output neurons (the number of output patterns we want to recognize), and the number of hidden layer neurons is set to be 170, hoping to balance efficiency and having enough weights to store all patterns. Split the training data into a training set and a validation set. After a neural network is trained on the training set, the classifications on the validation set are used to validate the neural network which will be used to predict outputs on the test set.

The objective function is still the total squared error function to achieve weight optimization using an iterative gradient descent algorithm. The reason not to directly optimize the AUC as an objective function is that the AUC statistic as a function is non-differentiable, whereas a differentiable approximation to the AUC require assumptions on data distributions. The AUC may not necessarily monotonically increase as the total squared error goes downward direction. Nevertheless, the AUC close to 1 may be achieved during the training process before error drops to 0. Hence we can set the stopping criterion as the AUC sufficiently large instead of total error approaching 0. Once the network can successfully separate two classes, in other words, the separation performance rises to some pre-specified target AUC value, the network training stops as early as possible, not caring about how low the error falls. The reason for early stopping is to some extent to overcome over-fitting problems

which lead to neural networks useless in terms of generalization to new data.

5.1 Experiments and Results

Firstly, randomly draw negatives of a sample size n and positives of the same sample size from the original training data set for training a neural network and use the remaining to validate neural networks. For a small n, the training error drops significantly, not necessarily down to 0, within a finite number of iterations, but it still comes up with a good AUC estimated from a ROC curve as shown in Figure 6.

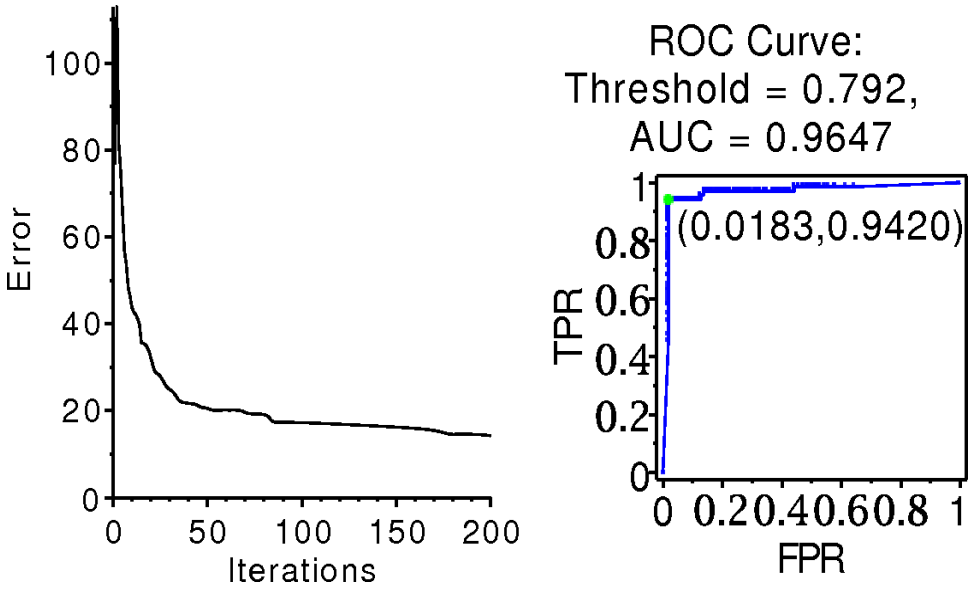

Figure 6: Training Error and ROC Curve

Vary the sample size (70, or 278, or 1112, or 4380). Initialize a neural network with initial value 0.3 (or 0.11) and learning rate 0.09 (or 0.1) stopping training at 100-500 iterations ending up with the AUC above 0.75-0.95 (depending on the sample

size, because it tends to be harder to obtain a high AUC value when sample size gets larger). How could the sample size be 1112 or 4380 for positives considering that there are only 348 positives in the original training set? The way to solve that is to cluster 348 positives into 174 centroids again and again and combine those thousands of centroids with the original 348 to get a new positive training subset from which random samples will be drawn.

The average performance on the test set is calculated over a cumulative number of trials as shown in Table 2. If more (at least one half) neural networks among all trials predict some pattern as positive, then that pattern would eventually be classified as positive, otherwise eventually it would be classified as negative if more trials output negative predictions.

Sample Size	Trials	Average Performance (TPR, FPR)	95% C.I. for AUC
70	1	(.571, .309)	[.553, .707]
70	25	(.609, .288)	[.583, .735]
70	302	(.689, .358)	[.588, .740]
70	327	(.672, .347)	[.585, .737]
278	1	(.630, .306)	[.585, .737]
278	5	(.603, .251)	[.599, .750]
278	301	(.676, .334)	[.593, .745]
278	331	(.681, .337)	[.595, .747]
278	400	(.685, .343)	[.593, .745]
1112	1	(.588, .283)	[.575, .728]
1112	9	(.605, .247)	[.602, .753]
1112	25	(.538, .222)	[.580, .733]
4380	1	(.521, .197)	[.585, .737]
4380	32	(.529, .205)	[.585, .737]
4380	84	(.508, .192)	[.580, .737]
ALL	836	(.647, .316)	[.588, .740]

Table 2: Test Performance on Average

For one single trial, increasing the sample size may result in improved performance. For a small sample size n, certain improvement could be achieved if average performance is taken on several repeated trials even though the average performance oscillates as more and more trials are conducted, but improvement in that sense of average performance on multiple trials becomes more difficult when the sample size gets larger. There might be, sporadically, some trials with AUC values below that pre-specified target level, but still having good or even a little bit better classification performance. That gives an inspiration that it might have better average performance of a set of neural networks with training stopping as early as possible.

A tighter 95% confidence interval for the AUC could be obtained if it is estimated from better validation performance on a larger validation set. The reason is that the confidence interval (its estimation in the aforementioned method only depends on the number of positives m, the number of negatives n, and the mis-classifications number k) tends to become tighter as m and n increase and k decreases. The test AUC appears to more often lie within the 95% confidence interval computed based on the validation performance than that based on the training performance. And the test performance appears strongly positively associated with the validation performance, so that it is good to estimate the confidence interval for the AUC based on either one. Also, experiments indicate that the aforementioned method for confidence interval estimation is only applicable for an equal number of positives and negatives ($m = n$). Therefore, if estimating the confidence interval for the AUC from the test performance, multiply the sum of the false negative rate and the false positive rate by the minority class size 586 (combine 348 positives in the training set and 238 positives

in the test set) to get the mis-classifications number k ($k = 586\,(FNR + FPR)$) and set both m and n equal to 586.

Secondly, partition the subset of 5474 negatives into groups and then partition each group into clusters, because it involves intensive computation to directly partition the original training subset of 5474 negatives into 348 clusters. Find a corresponding data point from the subset of 5474 negatives with a closest distance to each data point in the original training subset of 348 positives, respectively. Here the distance is measured by the angle defined by

$$\theta = \arccos\left(\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|}\right)$$

between two data points $\mathbf{a}$ and $\mathbf{b}$. The first-round gave 348 negative data points to constitute the $1st$ group which is then removed from the subset of negative training patterns; repeat the same process to construct the $2nd$ up to the $15th$ group and the last remaining 254 negatives form the $16th$ group. Then, cluster each of the $1st$ up to the $15th$ negative groups into 22 centroids, and cluster the $16th$ negative group into 18 centroids and combine all centroids to get a set of 348 representatives of negative training patterns. Then, randomly remove 70 from the original 348 positives and 70 from the 348 negative representatives and use the remaining 278 positives plus 278 negatives for training. The average performance over seven such trials is measured by: TPR .672, FPR .438, and the estimated 95% confidence interval for the AUC [.538, .693].

Thirdly, try to train a neural network with the original highly imbalanced data set. Restart neural networks using different initial values and learning rates. It has been found that an initial value in a range of $0.0055 - 0.011$ with a learning rate

$0.005 - 0.01$ (10/11 proportional to the initial value) ends up with relatively better AUC (0.72-0.76) right at the beginning (for example, 2 iterations) of the training process. As training goes on up to 128 iterations, the error decreases slightly but the AUC goes down from 0.732 to 0.654 and most operating points become denser toward the left-bottom corner. In other words, it turns out low FPR at most threshold levels but it seems hard to improve TPR, which is consistent with the theory that ANNs tend to ignore the minority class as noise. It has as good performance on the validation set as on the test set, indicating that a model properly fitting the training data and successfully validated by new data does have a capability to predict patterns in the test set. Then, train a set of neural networks with the original imbalanced data and figure out the average performance on 111 trials (initialize 0.011, learning rate 0.01, 20% for Validation): TPR .634, FPR .306, and the estimated 95% confidence interval for the AUC $[.586, .739]$.

Trials	Initialize	Learning Rate	Training AUC	Validation AUC	Test AUC
1	0.011	0.01	.732	.732	.707
2	0.0099	0.009	.721	.742	.698
3	0.0088	0.008	.724	.763	.706
4	0.0077	0.007	.733	.771	.709
5	0.0066	0.006	.735	.775	.700
6	0.0055	0.005	.729	.764	.701

Table 3: Train 2 Iterations With the Original Training Set

5.2 Discussion

Apparently, there appears to be a certain degree of information loss to draw a smaller subset of representatives from the original imbalanced data for training, and a neural network trained in that respect even with as high AUC as 0.95 does not necessarily outperform another neural network trained with the original data just with a relatively low AUC such as 0.73. Random sampling appears to work slightly better than k-means clustering. Average performance over several trials seems better than that on a single trial more obviously for a small sample size.

The maximum number of policy holders that could be found is 238 and random selection results in 48 policy holders if the task is to find the set of 800 customers in the test set that contains the most caravan policy holders. In the CoIL Challenge 2000 report [7], the winning model selected 121 policy holders (Naive Bayes approach), and neural networks selected 105 policy holders. In this thesis, a slightly improved result with respect to neural networks (selecting 124 policy holders from a set of 865 customers, approximately 115 out of 800, and $115 > 105$) has been obtained on one single trial with the best performance (trained by over sampling the minority class with a sample size 4380 to adjust to the sample size 4380 of the majority class).

One issue to resolve for clustering is how to better measure distance (heterogeneity/homogeneity) in a high dimensional space. Besides the default Euclidean distance, other alternatives such as the angle and the KL-divergence for information-theoretic clustering could be considered as well. Data points within a cluster should ideally be as homogeneous (closest distance) as possible, but there should be heterogene-

ity among cluster representatives. The statistical analysis of clusters must take into account the intra-cluster correlation and variation, and variations among clusters.

In all, we need to improve the sufficiency of the selected representative data to best approximate the underlying probability distribution of the original data. A certain degree of improvement tends to be made if random samples of size k (for example, 348) are drawn from the original subset of 5474 negatives repeatedly sufficiently many times, although it might oscillate especially at the beginning. In conclusion, a sufficiently large AUC as an early stopping criterion is able to measure the overall performance of a neural network with the mitigation of over-training and over-fitting problems.

BIBLIOGRAPHY

[1] G. A. Akerlof, The Market for "Lemons": Quality Uncertainty and the Market Mechanism, *The Quarterly Journal of Economics*, **84**(1970), 488-500.

[2] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data, *ACM SIGKDD Explorations Newsletter, SPECIAL ISSUE: Special Issue on Learning from Imbalanced Datasets*, ACM, New York, NY, **6**(2004), 20-29.

[3] M. Beigi and A. Zell, SPSO: Synthetic Protein Sequence Oversampling for Imbalanced Protein Data and Remote Homology Detection, *Biological and Medical Data Analysis: 7th International Symposium, ISBMDA 2006, Thessaloniki, Greece, December 2006 Proceedings*, eds. N. Maglaveras, I. Chouvarda, V. Koutkias, and R. Brause, Springer-Verlag Berlin Heidelberg, 2006, 104-115.

[4] C. Bowd, K. Chan, L. M. Zangwill, M. H. Goldbaum, T. W. Lee, T. J. Sejnowski, and R. N. Weinreb, Comparing Neural Networks and Linear Discriminant Functions for Glaucoma Detection Using Confocal Scanning Laser Ophthalmoscopy of the Optic Disc, *Investigative Ophthalmology & Visual Science*, **43**(2002), 3444-3454.

[5] A. P. Bradley, The use of the area under the ROC curve in the test of machine learning algorithms, *Pattern Recognition*, **30**(1997), 1145-1159.

[6] F. Chu, G. Jin, and L. Wang, Cancer Diagnosis and Protein Secondary Structure Prediction Using Support Vector Machines, *Support Vector Machines: Theory and Applications*, ed. L. Wang, Springer-Verlag Berlin Heidelberg, 2005, 343-363.

[7] The CoIL Challenge 2000 Report, *The Insurance Company Case*, available at `http://www.liacs.nl/~putten/library/cc2000/report2.html`.

[8] C. Cortes and M. Mohri, Confidence Intervals for the Area under the ROC Curve, *Advances in Neural Information Processing Systems 17: Proceedings of the 2004 Conference, Vol. 17*, eds. L. K. Saul and Y. Weiss and L. Bottou, MIT Press, Cambridge, MA, 2005, 305-312.

[9] G. Cybenko, Approximation by Super-positions of a Sigmoid Function, *Mathematics of Control, Signals and Systems*, **2**(1989), 303-314.

[10] M. Chetouani, A. Hussain, M. Faundez-Zanuy, and B. Gas, Non-linear Predictive Models for Speech Processing, *Artificial Neural Networks: Biological Inspirations - ICANN 2005: 15th International Conference, Warsaw, Poland, September 2005 Proceedings, Part II*, eds. W. Duch, J. Kacprzyk, E. Oja, and S. Zadrozny, Springer-Verlag Berlin Heidelberg, 2005, 779-784.

[11] A. Erkanli, M. Sung, E. J. Costello, and A. Angold, Bayesian semi-parametric ROC analysis, *Statistics in Medicine*, **25**(2006), 3905-3928, Published Online at www.interscience.wiley.com, DOI: 10.1002/sim.2496.

[12] S. Ertekin, J. Huang, and C. L. Giles, Active Learning for Class Imbalance Problem, *Proceedings of the 30th Annual International ACM SIGIR Conference*

on Research and Development in Information Retrieval, ACM, New York, NY, 2007, 823-824.

[13] A. A. V. Eye, P. Mair, and E. Y. Mun, Advances in Configural Frequency Analysis, *The Guilford Press*, New York, NY, 2010, 65-93.

[14] J. Gu, S. Ghosal, and A. Roy, Bayesian bootstrap estimation of ROC curve, *Statistics in Medicine*, John Wiley & Sons, Ltd., 2008, Published Online at www.interscience.wiley.com, DOI: 10.1002/sim.3366.

[15] K. O. Hajian-Tilaki, J. A. Hanley, L. Joseph, and J. P. Collet, A Comparison of Parametric and Non-parametric Approaches to ROC Analysis of Quantitative Diagnostic Tests, *Medical Decision Making*, **17**(1997), 94-102.

[16] J. A. Hanley and B. J. McNeil, The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve, *Radiology*, **143**(1982), 29-36.

[17] J. Kamruzzaman, R. Begg, and R. A. Sarker, Artificial Neural Networks in Finance and Manufacturing, *Idea Group Inc.*, 2006, 44-45.

[18] F. L. Kitchens, Financial Implications of Artificial Neural Networks in Automobile Insurance Underwriting, *International Conference on Fuzzy Sets and Soft Computing in Economics and Finance (FSSCEF), Saint-Petersburg, Russia, June 2004 Proceedings, Volume II*, eds. I. Batyrshin, J. Kacprzyk, and L. Sheremetov, 2004, 386-393.

[19] D. Knisley and J. Knisley, Graph Theoretic Models in Chemistry and Molecular Biology, *Handbook of Applied Algorithms: Solving Scientific, Engineering, and*

Practical Problems, eds. A. Nayak and I. Stojmenovic, John Wiley & Sons, Inc., Hoboken, New Jersey, 2008, 85-113.

[20] D. D. Lewis and J. Catlett, Heterogeneous Uncertainty Sampling for Supervised Learning, *Machine Learning: Proceedings of the Eleventh International Conference*, eds. W. W. Cohen and H. Hirsh, Morgan Kaufmann Publishers, San Francisco, CA, 1994, 148-156.

[21] T. W. Liao and E. Triantaphyllou, Recent Advances in Data Mining of Enterprise data: Algorithms and Applications, *Series on Computers and Operations Research, Vol.6*, World Scientific Publishing Co. Pte. Ltd., Singapore, 2007, 394.

[22] C. X. Ling, J. Huang, and H. Zhang, AUC: a statistically consistent and more discriminating measure than accuracy, *Proceedings of the Eighteenth International Joint Conference of Artificial Intelligence*, 2003, 519-526.

[23] J. Lowe and L. Pryor, Neural Networks v. GLMs in Pricing General Insurance, *General Insurance Convention 1996, Stratford, UK*, 1996, 417-438.

[24] Z. Lu, A. I. Rughani, B. I. Tranmer, and J. Bongard, Informative Sampling for Large Unbalanced Data Sets, *Genetic and Evolutionary Computation Conference (GECCO) 2008, Atlanta, Georgia, July 2008*, ACM, 2008, 2047-2053.

[25] D. K. McClish, Analyzing a Portion of the ROC Curve, *Medical Decision Making*, **9**(1989), 190-195.

[26] P. Mulquiney, Artificial Neural Networks in Insurance Loss Reserving, *Proceedings of the 2006 Joint Conference on Information Sciences (JCIS 2006)*, Kaohsi-

ung, Taiwan: Atlantis Press, 2006, Online Proceedings at http://www.atlantis-press.com/publications/aisr/jcis-06/, DOI:10.2991/jcis.2006.67.

[27] G. H. Nguyen, A. Bouzerdoum, and S. L. Phung, A Supervised Learning Approach for Imbalanced Data Sets, *International Conference on Pattern Recognition 08*, Tampa, Florida, 2008, 1-4, DOI: 10.1109/ICPR.2008.4761278.

[28] A. Owen, Empirical Likelihood Ratio Confidence Regions, *Annals of Statistics*, **18**(1990), 90-120.

[29] S. H. Park, J. M. Goo, and C. H. Jo, Receiver Operating Characteristic (ROC) Curve: Practical Review for Radiologists, *Korean J Radiol*, **5**(2004), 11-18.

[30] R. Pelessoni and L. Picech, Some applications of unsupervised neural networks in rate making procedure, *Proceedings of the General Insurance Convention and XXIX Astin Colloquium, Glasgow, Scotland, Vol. 2*, 1998, 550-567.

[31] Y. Peng, G. Kou, A. Sabatka, J. Matza, Z. Chen, D. Khazanchi, and Y. Shi, Application of Classification Methods to Individual Disability Income Insurance Fraud Detection, *Computational Science - ICCS 2007: 7th International Conference, Beijing, China, May 2007, Proceedings, Part III*, eds. Y. Shi, G. D. V. Albada, J. Dongarra, and P. M. A. Sloot, Springer-Verlag Berlin Heidelberg, 2007, 852-858.

[32] M. Prijs, L. Peelen, P. Bresser, and N. Peek, A Nearest Neighbor Approach to Predicting Survival Time with an Application in Chronic Respiratory Disease, *Artificial Intelligence in Medicine: 11th Conference on Artificial Intelligence in*

Medicine, AIME 2007, Amsterdam, The Netherlands, July 2007 Proceedings, eds. R. Bellazzi, A. Abu-Hanna, and J. Hunter, Springer-Verlag Berlin Heidelberg, 2007, 77-86.

[33] G. Qin and X. Zhou, Empirical Likelihood Inference for the Area Under the ROC Curve, *Biometrics*, **62**(2006), 613-622.

[34] S. Rosset, Model Selection via the AUC, *Proceedings of the Twenty-First International Conference on Machine Learning, Banff, Alberta, Canada*, 2004, 89-96.

[35] P. Sonego, A. Kocsor, and S. Pongor, ROC Analysis: Applications to the Classification of Biological Sequences and 3D Structures, *Briefings in Bioinformatics Advance Access*, **9**(2008), 198-209, DOI:10.1093/bib/bbm064.

[36] The Insurance Company Benchmark (CoIL 2000), *The UCI KDD Archive*, available at `http://kdd.ics.uci.edu/databases/tic/tic.html`.

[37] H. Zhang and C. X. Ling, An Improved Learning Algorithm for Augmented Naive Bayes, *Advances in Knowledge Discovery and Data Mining: 5th Pacific-Asia Conference, PAKDD 2001, Hong Kong, China, April 2001 Proceedings*, eds. D. Cheung, G. J. Williams, and Q. Li, Springer-Verlag Berlin Heidelberg, 2001, 581-586.

APPENDIX

Maple Code

```
> restart: with(GraphTheory); with(plots) : with(Statistics) : with(LinearAlgebra) :
    with(StringTools) : interface(displayprecision = 5);
#neural networks module
MakeANN := proc(ni :: posint, no :: posint)
 local nh, tmp, ActOnOut;

 nh := ni* no;
 if(hasoption([args[3..-1]],'hiddens','tmp') or
   hasoption([args[3..-1]],'hidden','tmp' )    ) then
   if(type(tmp,'posint')) then
     nh := tmp;
   end if:
 end if:

 ActOnOut := 1 :
 if(hasoption([args[3..-1]],'ActivationOnOutputs','tmp') ) then
  if(tmp = false or tmp = 0) then ActOnOut := 0 end if:
 end if:

 module( )
   local learnrate, numints, numouts, numhids, inputs, hiddens, outputs, delto, delth, whi,
    woh, alpha, cforward, forward, backprop, cbackprop,
      beta, errs, cnt, f, TrainingSet, Patterns, Complements, WeightsUpdated, whi_old,
    woh_old, ActivateOutputs;

   export Initialize, Get, SetAlpha, SetBeta, SetLearningRate, Forward, BackProp,
    Classify, Train, AddPattern, RemovePattern, CreateComplement,
       ResetPatterns, PredictComplements, Errors, ShowTrainingResults, ShowPatterns,
    ROCanalysis;

   learnrate := 0.1 :
   numints := ni : #This is the number of inputs in each pattern
   numouts := no : #This is the number of outputs
   numhids := nh :
   WeightsUpdated := false :
   errs := [ ] :


   cnt := 0 :
   ActivateOutputs := ActOnOut :
```

```
# Initialize arrays
inputs := Vector[column](numints, datatype = float[8]) :
hiddens := Vector[column](numhids, datatype = float[8]) :
outputs := Vector[column](numouts, datatype = float[8]) :

#the "delta's"
delto := Vector[column](numouts, datatype = float[8]) :
delth := Vector[column](numhids, datatype = float[8]) :

#weights in the form whi[hidden][input] and woh[outputs][hidden]
whi := Matrix(numhids, numints, datatype = float[8]) :#Hidden layer weights
woh := Matrix(numouts, numhids, datatype = float[8]) :#Output layer weights

#Define threshold function
alpha := 5e0:
beta := 0.0:
f := entryv → 1 / (exp(-alpha*(entryv-beta)) + 1);

Get := proc(ref)
  local item:
  item := convert(ref,'string') :
  if( item = "learnrate"  ) then return learnrate  end if:
  if( item = "numints"    ) then return numints    end if:
  if( item = "numouts"    ) then return numouts    end if:
  if( item = "numhids"    ) then return numhids    end if:
  if( item = "whi"        ) then return whi        end if:
  if( item = "woh"        ) then return woh        end if:
  if( item = "alpha"      ) then return alpha      end if:
  if( item = "beta"       ) then return beta       end if:
  if( item = "errs"       ) then return errs       end if:
  if( item = "cnt"        ) then return cnt        end if:




  if( item = "f"          ) then return f          end if:
  if( item = "fplot"      ) then return plot( f, beta-10 / alpha..beta + 10 / alpha, color
 = blue) end if:
  if( item = "TrainingSet" ) then return TrainingSet end if:
  if( item = "Patterns"   ) then return Patterns   end if:
  if( item = "Complements" ) then return Complements end if:
end proc:
```

```
SetAlpha :=proc(val :: numeric)
  local tmp:
  tmp := alpha :
  alpha := val:
  return tmp:
end proc:

SetBeta :=proc(val :: numeric)
  local tmp:
  tmp := beta :
  beta := val:
  return beta;
end proc:

SetLearningRate :=proc(val :: numeric)
  local tmp:
  tmp := learnrate :
  if( val > 0 ) then
    learnrate := val;
  end if:
  return learnrate :
end proc:

Initialize :=proc(maxamt) #initalize weights to small random values with mag
  local i, h, o :
  for h from 1 to numhids do
    for i from 1 to numints do


      whi[ h, i] := (rand( )*10^(-11)-5)*maxamt:
    end do:
  end do:
  for o from 1 to numouts do
    for h from 1 to numhids do
      woh[o, h] := (rand( )*10^(-11)-5)*maxamt:
    end do:
  end do:
  whi_old := Matrix(whi) :
  woh_old := Matrix(woh) :
  cnt := 0 :
  WeightsUpdated := true :
  errs := [ ] :
end proc:
```

```
#Define the forward function
#Compile Options

forward := proc( whi :: Matrix(datatype = float[ 8 ]), woh :: Matrix(datatype
 = float[ 8 ]), hiddens :: Vector(datatype = float[ 8 ]),
          inpat :: Vector(datatype = float[ 8 ]), outputs :: Vector(datatype
 = float[ 8 ]),
          alpha :: float[8], beta :: float[8], numints :: posint, numouts :: posint,
 numhids :: posint, ActOnOutputs :: posint )

#cforward(whi, woh, hiddens, inpat, outputs, alpha, beta, numints, numouts,
 numhids)
  local i :: posint, j :: posint;

  ##hiddens := whi.inpat
  for i from 1 to numhids by 1 do
    hiddens[i] := 0 :
    for j from 1 to numints by 1 do
      hiddens[i] := hiddens[i] + whi[i, j]*inpat[j];
    end do:
  end do:

  for i from 1 to numhids by 1 do
    hiddens[i] := 1 / (exp(-alpha*(hiddens[i]-beta)) + 1);
  end do:
  ##outputs:=woh.hiddens:
  for i from 1 to numouts by 1 do
    outputs[i] := 0 :
    for j from 1 to numhids by 1 do
      outputs[i] := outputs[i] + woh[i, j]*hiddens[j];
    end do:
  end do:
  if(ActOnOutputs = 2) then
    for i from 1 to numouts do
      outputs[i] := 1 / (exp(-alpha*(outputs[i]-beta)) + 1);
    end do:
  end if:
  return true;
end proc:

 backprop :=proc(whi :: Matrix(datatype = float[ 8 ]), woh :: Matrix(datatype
 = float[ 8 ]),
          hiddens :: Vector(datatype = float[ 8 ]),
          lrninput :: Vector(datatype = float[8]),
          lrnoutput :: Vector(datatype = float[8]), outputs :: Vector(datatype
 = float[8]),
          delto :: Vector(datatype = float[8]),
          delth :: Vector(datatype = float[8]),
          alpha :: float[8], beta :: float[8], learnrate :: float[8],
```

```
  incError := 0.0;
  for i from 1 to numouts by 1 do
    incError := incError + (outputs[i]-lrnoutput[i])*(outputs[i]-lrnoutput[i]) :
    if(ActOnOutputs = 2) then
      delto[i] := alpha*outputs[i]*(1-outputs[i])*(lrnoutput[i]-outputs[i]) :
    else
      delto[i] := lrnoutput[i]-outputs[i] :
    end if:
    for h from 1 to numhids do
      woh[i, h] := woh[i, h] + learnrate*delto[i]*hiddens[h] :
    end do:
  end do:
  for h from 1 to numhids by 1 do
    delth[h] := alpha*hiddens[h]*(1-hiddens[h])*add(woh[j, h]*delto[j], j = 1
..numouts) :
    for i from 1 to numints by 1 do
      whi[h, i] := whi[h, i] + learnrate*delth[h]*lrninput[i] :
    end do:
  end do:
  return incError;

end proc:

  cforward := Compiler:-Compile(forward);
  cbackprop := Compiler:-Compile(backprop);

Forward := proc(inpt :: Vector)
  local inpfl8;

  inpfl8 := Vector(inpt, datatype = float[8]) :
  cforward(whi, woh, hiddens, inpfl8, outputs, convert(alpha,'float[8]'), convert(beta,
'float[8]'), numints, numouts, numhids, ActivateOutputs + 1 ) :
  outputs;

end proc:

#Define the forward function

Classify :=proc(inpat :: Vector)
  LinearAlgebra[Transpose](Forward(inpat)) :
end proc:

BackProp :=proc(lrninput :: Vector, lrnoutput :: Vector)
  local tmpLinpt, tmpLoutp;
```

```
  Forward(lrninput) :
  tmpLinpt := Vector(lrninput, datatype = float[8]);
  tmpLoutp := Vector(lrnoutput, datatype = float[8]);
  WeightsUpdated := true:

  cbackprop(whi, woh, hiddens, tmpLinpt, tmpLoutp, outputs, delto, delth, alpha,
beta, learnrate, numints, numouts, numhids, ActivateOutputs + 1 ) :

end proc:

##Training Set Algorithms

Patterns := [ ]:

AddPattern := proc(inp, oup)
  local inpt, oupt, attr;

  attr := "" :
  hasoption( [args[3..-1]],'attributes','attr') :

  inpt := Vector(convert(inp,'list'), orientation = column) :#print(inpt);
  oupt := Vector(convert(oup,'list'), orientation = column) :#print(oupt);

  if(LinearAlgebra:-Dimensions(inpt) = numints and LinearAlgebra:-
Dimensions(oupt) = numouts ) then
    Patterns := [ Patterns[ ], [inpt, oupt, attr] ]:
    TrainingSet := [seq(i, i = 1..nops(Patterns)) ]:
    true:
  else
    false:
  end if:
end proc:

RemovePattern := proc(PatIndex :: posint)
  if(PatIndex ≤ nops(Patterns) ) then
    Patterns := [ Patterns[1..PatIndex-1], Patterns[PatIndex+1..-1]]:
    true:
  else
    false:
  end if:

end proc:
```

```
ShowPatterns := proc( )
  Patterns;
end proc:

ResetPatterns := proc( )
  Patterns := [ ]:
  TrainingSet := [ ]:
  Complements := [ ]:
end proc:

CreateComplement := proc( amt )
#amt is the proportion or number to use to put into the complement
  local tmp:

  if(0 < amt and amt ≤ 1 ) then
    tmp := floor(amt* nops(Patterns) ) :
  elif( 1 < amt and amt < nops(Patterns) ) then
    tmp := amt:
  end if:

  TrainingSet := {seq(i, i = 1 ..nops(Patterns))}:
  Complements := {seq(combinat[randperm](TrainingSet)[j], j = 1 ..tmp)}:
  TrainingSet := convert(TrainingSet minus Complements,'list');
  Complements := convert(Complements, 'list') :

  return 'TrainBy', TrainingSet,'ValidateWith', Complements;
end proc:

Train := proc(NumberOfReps :: posint)
   local InitValue, terr, Indices, i, n, esum, v;

   for n from 1 to NumberOfReps do
     Indices := combinat[randperm](TrainingSet) :
     esum := 0 :
     for i in TrainingSet do
       v := Patterns[i]:
       esum := esum + BackProp(v[1], v[2]) :
     end do:
   #if ROCanalysis(Patterns):-AUC() > .98 then
    #  break:
   #end if:
     cnt := cnt + 1 :
     errs := [errs[ ], esum]:
   end do:
   return errs;
 end proc:
```

```
Errors := proc( )
  local tmp:

  tmp := [errs[ ], LinearAlgebra:-MatrixNorm(whi - whi_old), LinearAlgebra:-
MatrixNorm(woh - woh_old)]:
  if(WeightsUpdated) then
    whi_old := Matrix(whi) :
    woh_old := Matrix(woh) :
    WeightsUpdated := false:
  end if:
  return errs;
end proc;

ShowTrainingResults := proc( )
  local v, i, err, cls;

  err := 0 :
  print(TrainingSet) :
  for i in TrainingSet do
    v := Patterns[i] :
    cls := Forward(v[1]) :
    print(i, cls, v[2]);
    err := err + LinearAlgebra:-VectorNorm(cls-v[2]) :
  end do:
  return err / nops(TrainingSet) :
end proc:

PredictComplements := proc( )
  local v, i, err, cls;

  err := 0 :
  #com_res := []:
  print(Complements) :
  for i in Complements do
    v := Patterns[i] :
    cls := Forward(v[1]) :
    print(i, cls, v[2]);
    err := err + LinearAlgebra:-VectorNorm(cls-v[2]) :
  end do:
  return err / nops(Patterns) :
end proc:
```

```
#incorporate the analysis of the Area under the Receiver Operating Characteristics
curve
ROCanalysis := proc(TrainingPatterns)
  local PosPatterns, NegPatterns, k, PosPreds, PosNum, i, NegPreds, NegNum, j,
_ROC_, BestThresh, thresh, TruePos, FalseNeg, PosPred, FalsePos, NegPred, tmp,
dist, _AUC_;

  PosPatterns := { }:
  NegPatterns := { }:
  for k from 1 to nops(TrainingPatterns) do
    if( op(1, convert(TrainingPatterns[k][2], list)) = 1 ) then
      PosPatterns := PosPatterns union {TrainingPatterns[k]}:
    else
      NegPatterns := NegPatterns union {TrainingPatterns[k]}:
    end if:
  end do: #print(nops(NegPatterns));

  if(nops(PosPatterns) = 0 or nops(NegPatterns) = 0) then
    error "ROC analysis is empty."
  end if:

  PosPreds := [ ]:
  PosNum := nops(PosPatterns) :
  PosPatterns := combinat[randperm](PosPatterns);
  for i from 1 to PosNum by 1 do
    PosPreds := [PosPreds[ ], op(1, convert(Classify(PosPatterns[i][1]),'list')) / (op(1,
convert(Classify(PosPatterns[i][1]),'list')) + op(2,
convert(Classify(PosPatterns[i][1]),'list')))]:
  end do:

  NegPreds := [ ]:
  NegNum := nops(NegPatterns) :
  NegPatterns := combinat[randperm](NegPatterns);
  for j from 1 to NegNum by 1 do
    NegPreds := [NegPreds[ ], op(1, convert(Classify(NegPatterns[j][1]),'list'))
 / (op(1, convert(Classify(NegPatterns[j][1]),'list')) + op(2,
convert(Classify(NegPatterns[j][1]),'list')))]:
  end do:


  _ROC_ := [ ]:
  BestThresh := [2, 0, [0, 0]]:
  for thresh from 0 to 1 by 0.001 do
    TruePos := 0:
    FalseNeg := 0:
```

```
  for PosPred in PosPreds do
    if(PosPred > thresh) then
      TruePos := TruePos + 1 :
      #FalseNeg := FalseNeg + 1:
    end if:
  end do:
  FalsePos := 0 :
  for NegPred in NegPreds do
    if(NegPred > thresh) then
      FalsePos := FalsePos + 1 :
    end if:
  end do:
  tmp := [evalf(FalsePos / NegNum), evalf(TruePos / PosNum) ]:
  dist := evalf(sqrt(tmp[1]^2 + (tmp[2]-1)^2)) :
    if( dist < BestThresh[1]) then
      BestThresh := [dist, thresh, tmp]:
    end if:
  _ROC_ := [ _ROC_[ ], tmp ]:

  _AUC_ := add( abs((_ROC_[k][1] - _ROC_[k+1][1])) * (_ROC_[k][2] + _ROC_[k
+1][2]), k=1..(nops(_ROC_)-1)) / 2;
 end do:
 return
   module( )
     local auc, best, roc, clst;
     export AUC, ROCplot, BestThreshold, ROC;

     best := BestThresh[2]:
     clst := BestThresh[3]:
     auc := _AUC_;
     roc := _ROC_;

     AUC := ( ) → auc;
     BestThreshold := ( ) → best;
     ROC := ( ) → roc;
     ROCplot := ( ) → plots[display](
       plots[listplot]( roc, color = blue ),
       plots[listplot]( roc, style = point, symbol = solidcircle, color = blue),
       plots[pointplot]( clst, style = point, symbol = solidcircle, color = green,
 symbolsize = 20),
       plots[textplot]( [clst[1], clst[2], cat("(", sprintf("%0.4f", clst[1]), ",",
sprintf("%0.4f", clst[2]), ")")], font = [TIMES, ROMAN, 14], align = [BELOW,
RIGHT]),
         axes = boxed, labels = ["FPR", "TPR"], labeldirections = [horizontal, vertical],
```

```
            font = [TIMES, ROMAN, 14], view = [0..1, 0..1], title
    = cat("ROC Curve: \n Threshold = ", sprintf("%0.3f", best), ",    AUC = ",
     sprintf("%0.4f", auc))
          );
       end module;
    end proc:

  end module:

end proc:

 >#Import the test data set for evaluation
EvalPatterns := ExcelTools:-
    Import(
    "/home/daniel/Documents/ROCanalysisData_Daoping/ticeval2000_Normalized.
    xls", "Sheet2", "A1:CG4000") :
EvalClassifications := ExcelTools:-
    Import(
    "/home/daniel/Documents/ROCanalysisData_Daoping/ticeval2000_Normalized.
    xls", "Sheet2", "CH1:CH4000") :
EvalSet := [seq([convert(EvalPatterns[i], Vector), Vector([EvalClassifications[i], 1
    -EvalClassifications[i]])], i = 1..4000)]:
#Import 348 positive training patterns
PosPatterns := ExcelTools:-
    Import(
    "/home/daniel/Documents/ROCanalysisData_Daoping/ticdata2000_normalized_c\
    luster.xls", "Sheet1", "A1:CG348") :
PosPatterns := {seq(convert(PosPatterns[i], Vector), i = 1..348)}:
#PosSet := {seq([PosPatterns[i],Vector([1,0])],i=1..348)}:
#Import 5474 negative training patterns
NegPatterns := ExcelTools:-
    Import(
    "/home/daniel/Documents/ROCanalysisData_Daoping/ticdata2000_normalized_c\
    luster.xls", "Sheet1", "A349:CG5822") :
NegPatterns := {seq(convert(NegPatterns[i], Vector), i = 1..5474)}:
#NegSet := {seq([NegPatterns[i],Vector([0,1])],i=1..5474)}:

 >
    #Draw a random sample of size m from the positive subset of 348 for training and
    the remaining for validation
ran_samp_pos := proc(data, size)
  local ran_ind, ran_, i, j, k, l, m, Ran:
  global PosTrainSet, PosValidPatterns, PosValidSet:
  # generate by randomly assigning index
  ran_ := { }:
  Ran := combinat[randperm]({seq(i, i = 1..nops(data))}):
  ran_ind := {seq(Ran[j], j = 1..size)}:
  for k in ran_ind do
```

```
ran_samp_pos(PosPatterns, 70);

    #Draw a random sample of size n from the negative subset of 5474 for training and
    the remaining for validation
ran_samp_neg := proc(data, size)
  local ran_ind, ran_, i, j, k, l, m, Ran:
  global NegTrainSet, NegValidPatterns, NegValidSet:
  # generate by randomly assigning index
  ran_ := { }:
  Ran := combinat[randperm]({seq(i, i = 1..nops(data))}):
  ran_ind := {seq(Ran[j], j = 1..size)}:
  for k in ran_ind do
    ran_ := ran_ union {data[k]}: #NegTrainPatterns
  end do: print("ran_ind", ran_ind, "ran_", ran_[1], whattype(%));
  NegTrainSet := {seq([ran_[l], Vector([0, 1])], l = 1..size)}:
  NegValidPatterns := data minus ran_:
  NegValidSet := {seq([NegValidPatterns[m], Vector([0, 1])], m = 1
    ..nops(NegValidPatterns))}:
  return;
end proc:
ran_samp_neg(NegPatterns, 70);
#Combine negatives with positives to obtain the training set and the validation set
TrainingSet := PosTrainSet union NegTrainSet:
ValidationSet := PosValidSet union NegValidSet:
#create a neural network
TestNet := MakeANN(85, 2);
for TS in TrainingSet do
  TestNet:-AddPattern(TS[1], TS[2])
end do:
TestNet:-Get('alpha');
TestNet:-Get(fplot);
TestNet:-Initialize(0.05);
TestNet:-SetLearningRate(0.0125);
TestNet:-Train(100):
plots[display](
  plots[listplot](%),
  plot(0, 0..nops(%)),
  labels = ["Iterations", "Error"], labeldirections = [horizontal, vertical],
  font = [TIMES, ROMAN, 12]
);
TestNet:-ROCanalysis(TrainingSet):-ROCplot( );

 > #Prediction outputs on the validation set
ValidRes := [ ]:
for vs in ValidationSet do
  tmp := TestNet:-Classify(vs[1]):
```

```
  ValidRes := [ ValidRes[ ], [tmp / (tmp[1] + tmp[2]), vs[2]] ]:
  #print( ValidRes[-1] );
end do:
#Classification performance on the validation set
TP := 0:
FN := 0:
TN := 0:
FP := 0:
Threshold := .931:
valid_res := Vector(nops(ValidRes)):
for i from 1 to nops(ValidRes) do
  if (ValidRes[i][1][1] > Threshold and ValidRes[i][2][1] = 1) then
    TP := TP+1:
    valid_res[i] := 1:
  elif (ValidRes[i][1][1] > Threshold and ValidRes[i][2][1] = 0) then
    FP := FP+1:
    valid_res[i] := 1:
  elif (ValidRes[i][1][1] < Threshold and ValidRes[i][2][1] = 1) then
    FN := FN+1:
    valid_res[i] := 0:
  elif (ValidRes[i][1][1] < Threshold and ValidRes[i][2][1] = 0) then
    TN := TN+1:
    valid_res[i] := 0:
  end if:
end do: print("TP", TP, "FN", FN, "TPR", evalf(TP/ (TP+FN)), "TN", TN, "FP", FP, "FPR",
    evalf(FP/ (FP+TN))):
TestNet:-ROCanalysis(ValidationSet):-ROCplot( );

 > #Prediction outputs on the test set
EvalRes := [ ]:
for es in EvalSet do
  tmp := TestNet:-Classify(es[1]):
  EvalRes := [ EvalRes[ ], [tmp / (tmp[1] + tmp[2]), es[2]] ]:
  #print( EvalRes[-1] );
end do:
#Classification performance on the test set
Threshold := .931:
TP := 0:
FN := 0:
TN := 0:
FP := 0:
eval_res := Vector(nops(EvalRes)):
for i from 1 to nops(EvalRes) do
  if (EvalRes[i][1][1] > Threshold and EvalRes[i][2][1] = 1) then
```

```
    TP := TP + 1 :
    eval_res[i] := 1 :
  elif (EvalRes[i][1][1] > Threshold and EvalRes[i][2][1] = 0) then
    FP := FP + 1 :
    eval_res[i] := 1 :
  elif (EvalRes[i][1][1] < Threshold and EvalRes[i][2][1] = 1) then
    FN := FN + 1 :
    eval_res[i] := 0 :
  elif (EvalRes[i][1][1] < Threshold and EvalRes[i][2][1] = 0) then
    TN := TN + 1 :
    eval_res[i] := 0 :
  end if:
end do: print("TP", TP, "FN", FN, "TPR", evalf(TP/(TP+FN)), "TN", TN, "FP", FP, "FPR",
    evalf(FP/(FP+TN))) :
#Export evaluation results
#ExcelTools:-Export(eval_res,"eval_res.xls","eval_res","A1");

 > #Average performance over trials
EvalRes := ExcelTools:-
    Import("/home/daniel/Documents/ROCanalysisData_Daoping/eval_res.xls",
     "eval_res", "C1:C4000") :
Target := ExcelTools:-
    Import(
    "/home/daniel/Documents/ROCanalysisData_Daoping/ticeval2000_Normalized.
    xls", "Sheet2", "CH1:CH4000") :
TP := 0 :
FP := 0 :
FN := 0 :
TN := 0 :
threshold := 150.5 :
for i from 1 to 4000 do
  if op(convert(EvalRes[i], list)) > threshold and op(convert(Target[i], list)) = 1 then
    TP := TP + 1 :
  elif op(convert(EvalRes[i], list)) > threshold and op(convert(Target[i], list)) = 0 then
    FP := FP + 1 :
  elif op(convert(EvalRes[i], list)) < threshold and op(convert(Target[i], list)) = 1 then
    FN := FN + 1 :
  elif op(convert(EvalRes[i], list)) < threshold and op(convert(Target[i], list)) = 0 then
    TN := TN + 1 :
  end if:
end do: print("TP", TP, "FN", FN, "TPR", evalf(TP/(TP+FN)), "TN", TN, "FP", FP, "FPR",
    evalf(FP/(FP+TN)), "k", evalf(FP*(TP+FN)/(FP+TN)+FN)) :

 >
    #Estimate 95% confidence interval for the AUC with lower bound given by minimum
    of "cil" and upper bound given by maximum of "ciu"
cil := [ ]:
ciu := [ ]:
```

```
m := 586:
n := 586:
k0 := 449:
kl := floor(k0 - sqrt((m + n) / 2) * solve(int(exp(-x^2 / 2) / sqrt(2*Pi), x = t..infinity)
    = (1-.95) / 2, t)) + 1:
ku := floor(k0 + sqrt((m + n) / 2) * solve(int(exp(-x^2 / 2) / sqrt(2*Pi), x = t..infinity)
    = (1-.95) / 2, t)):
for k from kl to ku do
  l := evalf( 1 - k/ (m + n) - (m - n)^2 * (m + n + 1) / 4*m*n* ( k/ (m + n)
    - sum(binomial(m + n, x), x = 0..k-1) / sum(binomial(m + n + 1, x), x = 0..k) ) - (1
    / sqrt(1-.95)) * sqrt( (m + n + 1) * (m + n) * (m + n-1) * (3 * ((m - n)^2 + m + n)
    + 2) * ((m + n-2) * sum(binomial(m + n-3, x), x = 0..k-4) / sum(binomial(m + n
    + 1, x), x = 0..k) - (2*m-n + 3*k-10) * sum(binomial(m + n-2, x), x = 0..k-3)
    / sum(binomial(m + n + 1, x), x = 0..k)) / (72*m^2*n^2) + (m + n + 1) * (m + n)
    * (3 * ((m - n)^2 + m + n) + 2) * (m^2-n*m + 3*k*m-5*m + 2*k^2-n*k + 12-9
    * k) * sum(binomial(m + n-1, x), x = 0..k-2) / sum(binomial(m + n + 1, x), x = 0..k)
    / (48*m^2*n^2) - (m + n + 1)^2 * (m-n)^4 * (sum(binomial(m + n, x), x = 0..k
    -1) / sum(binomial(m + n + 1, x), x = 0..k))^2 / (16*m^2*n^2) - (m + n + 1)
    * ( (3 * ((m - n)^2 + m + n) + 2) * k^3 + 3 * (m - 1) * (3 * ((m - n)^2 + m + n)
    + 2) * k^2 + ((-3*n^2 + 3*m*n - 3*m + 8) * (3 * ((m - n)^2 + m + n) + 2) - 6
    * (6*m*n + m + n)) * k + (-3*m^2 + 7 * (m + n) + 3*m*n) * (3 * ((m - n)^2
    + m + n) + 2) - 2 * (6*m*n + m + n) ) * sum(binomial(m + n, x), x = 0..k-1)
    / sum(binomial(m + n + 1, x), x = 0..k) / (72*m^2*n^2) + k* ( (m + n + 1) * (3
    * ((m - n)^2 + m + n) + 2) * k^2 + ((-3*n^2 + 3*m*n + 3*m + 1) * (3 * ((m
    - n)^2 + m + n) + 2) - 12 * (3*m*n + m + n) - 8) * k + (-3*m^2 + 7*m + 10*n
    + 3*n*m + 10) * (3 * ((m - n)^2 + m + n) + 2) - 4 * (3*m*n + m + n + 1) )
    / (144*m^2*n^2) ) ):
  cil := [cil[ ], l]:
  u := evalf( 1 - k/ (m + n) - (m - n)^2 * (m + n + 1) / 4*m*n* ( k/ (m + n)
    - sum(binomial(m + n, x), x = 0..k-1) / sum(binomial(m + n + 1, x), x = 0..k) ) + (1
    / sqrt(1-.95)) * sqrt( (m + n + 1) * (m + n) * (m + n-1) * (3 * ((m - n)^2 + m + n)
    + 2) * ((m + n-2) * sum(binomial(m + n-3, x), x = 0..k-4) / sum(binomial(m + n
    + 1, x), x = 0..k) - (2*m-n + 3*k-10) * sum(binomial(m + n-2, x), x = 0..k-3)
    / sum(binomial(m + n + 1, x), x = 0..k)) / (72*m^2*n^2) + (m + n + 1) * (m + n)
    * (3 * ((m - n)^2 + m + n) + 2) * (m^2-n*m + 3*k*m-5*m + 2*k^2-n*k + 12-9
    * k) * sum(binomial(m + n-1, x), x = 0..k-2) / sum(binomial(m + n + 1, x), x = 0..k)
    / (48*m^2*n^2) - (m + n + 1)^2 * (m-n)^4 * (sum(binomial(m + n, x), x = 0..k
    -1) / sum(binomial(m + n + 1, x), x = 0..k))^2 / (16*m^2*n^2) - (m + n + 1)
    * ( (3 * ((m - n)^2 + m + n) + 2) * k^3 + 3 * (m - 1) * (3 * ((m - n)^2 + m + n)
    + 2) * k^2 + ((-3*n^2 + 3*m*n - 3*m + 8) * (3 * ((m - n)^2 + m + n) + 2) - 6
    * (6*m*n + m + n)) * k + (-3*m^2 + 7 * (m + n) + 3*m*n) * (3 * ((m - n)^2
    + m + n) + 2) - 2 * (6*m*n + m + n) ) * sum(binomial(m + n, x), x = 0..k-1)
    / sum(binomial(m + n + 1, x), x = 0..k) / (72*m^2*n^2) + k* ( (m + n + 1) * (3
    * ((m - n)^2 + m + n) + 2) * k^2 + ((-3*n^2 + 3*m*n + 3*m + 1) * (3 * ((m
    - n)^2 + m + n) + 2) - 12 * (3*m*n + m + n) - 8) * k + (-3*m^2 + 7*m + 10*n
    + 3*n*m + 10) * (3 * ((m - n)^2 + m + n) + 2) - 4 * (3*m*n + m + n + 1) )
    / (144*m^2*n^2) ) ):
  ciu := [ciu[ ], u]:
```

```
> # k-means clustering for multiple dimentional data
with(Student[NumericalAnalysis]) : with(MTM) :
cluster := proc(data, size)
  local n_point, n_cluster, ran_ind, ran_centroid, rem_data, newdata, data_group,
    min_dis, dist, Min_Dis, Dist, group_index, Group_Index, a, b, c, d, e, f, g, h, i, j, k, l,
    m, n, p, q, r, s, t, u, v, w, x, y, R, centroid, centroid_, count, data_regroup, val, dis,
    total_dis, max_dis, centroid_rec, centroid_rec_ :
  n_point := nops(data) :
  n_cluster := size :
  # generate the centroids by randomly assigning index
  ran_centroid := [ ] :
  R := combinat[randperm]({seq(a, a = 1 ..n_point)}) :
  ran_ind := {seq(R[b], b = 1 ..n_cluster)} :
  #ran_num := RandomTools[Generate](integer(range=0..n_point)): #rand(0..n_point):
  for c in ran_ind do
    ran_centroid := [ran_centroid[ ], data[c]] :
  end do: print("ran_ind", ran_ind, "ran_centroid", ran_centroid[1], whattype(%));
  newdata := convert(data,'set') minus convert(ran_centroid,'set') :
  n_point := n_point-n_cluster:

  # group points around a centroid of nearest distance
  #with(Student[NumericalAnalysis]):
  #min_dis := Vector(n_point):
  #dist := Vector(n_cluster):
  data_group := Array(1 ..n_cluster) :

  for v from 1 to n_cluster do
    data_group[v] := {ran_centroid[v]} :
  end do:

  for d in seq(e, e = 1 ..n_point) do
    min_dis := arccos(DotProduct(newdata[d], ran_centroid[1])
    / (VectorNorm(newdata[d], 2) * VectorNorm(ran_centroid[1], 2))) :
    #Distance(newdata[d],ran_centroid[1],2):
    group_index := 1 :
    for f in seq(g, g = 1 ..n_cluster) do
      dist := arccos(DotProduct(newdata[d], ran_centroid[f])
      / (VectorNorm(newdata[d], 2) * VectorNorm(ran_centroid[f], 2))) :
      #Distance(newdata[d], ran_centroid[f],2):
      if (dist < min_dis) then
        min_dis := dist:
        group_index := f:
      end if:
    end do:
    data_group[group_index] := data_group[group_index] union {newdata[d]} :
```

```
end do: print("min_dis", min_dis, "data_group[3]", whattype(data_group[3]),
  "nops(data_group[])", nops(data_group[1]), nops(data_group[2]),
  nops(data_group[3]), nops(data_group[n_cluster-2]), nops(data_group[n_cluster
  -1]), nops(data_group[n_cluster]), "group_index", group_index) :

#calculate the centroids
centroid := [ ]:
centroid_ := Vector(n_cluster) :
for g in seq(h, h = 1..n_cluster) do
  centroid_[g] := convert(evalm(add(data_group[g][i], i = 1..nops(data_group[g]))
  / nops(data_group[g])),'Vector') :
  centroid := [centroid[ ], centroid_[g]]:
end do: print("centroid_[1]", centroid_[1], whattype(%)) :


 #regroup points around centroids and recalculate centroids until no change would
 be made
val := 1 :
count := 0 :
while (val = 1) do
  #Min_Dis := Vector(n_point):
  #Dist := Vector(n_cluster):
  data_regroup := Array(1..n_cluster) :

  for w from 1 to n_cluster do
    data_regroup[w] := {centroid[w]}:
    end do:

  for j in seq(k, k = 1..n_point) do
    Min_Dis := arccos(DotProduct(data[j], centroid[1]) / (VectorNorm(data[j], 2)
  * VectorNorm(centroid[1], 2))) :#Distance(data[j], centroid[1],2):
    Group_Index := 1 :
    for l in seq(m, m = 1..n_cluster) do
      Dist := arccos(DotProduct(data[j], centroid[l]) / (VectorNorm(data[j], 2)
  * VectorNorm(centroid[l], 2))) :# Distance(data[j], centroid[l],2):
      if (Dist < Min_Dis) then
        Min_Dis := Dist :
        Group_Index := l:
      end if:
    end do:
    data_regroup[Group_Index] := data_regroup[Group_Index] union {data[j]}:
  end do: print("data_regroup[1]", whattype(data_regroup[1]),
 "nops(data_regroup[])", nops(data_regroup[1]), nops(data_regroup[2]),
 nops(data_regroup[3]), nops(data_regroup[n_cluster-2]),
 nops(data_regroup[n_cluster-1]), nops(data_regroup[n_cluster]), "Group_Index",
 Group_Index) :
```

```
    centroid_rec := Array(1..n_cluster):
    #centroid_rec_ := Vector(n_cluster):
    for n in seq(p, p = 1..n_cluster) do
      centroid_rec[n] := convert(evalm(add(data_regroup[n][q], q = 1
    ..nops(data_regroup[n])) / nops(data_regroup[n])),'Vector'):
      #centroid_rec := [centroid_rec[],centroid_rec_[n]]:
    end do:

    dis := Vector(n_cluster):
    total_dis := 0:
    max_dis := arccos(DotProduct(centroid_rec[1], centroid[1])
    / (VectorNorm(centroid_rec[1], 2) * VectorNorm(centroid[1], 2))):
   #Distance(centroid_rec[1],centroid[1],2):
    for x from 1 to n_cluster do
      dis[x] := arccos(DotProduct(centroid_rec[x], centroid[x])
    / (VectorNorm(centroid_rec[x], 2) * VectorNorm(centroid[x], 2))):
   #Distance(centroid_rec[x], centroid[x],2):
      total_dis := total_dis + dis[x]:
      if dis[x] ≥ max_dis then
        max_dis := dis[x]:
      end if:
    end do:

    if (max_dis < 0.01) then
      val := 0:
    else
      val := 1:
      centroid := centroid_rec:
    end if:
    count := count + 1:
  end do: print("total_dis", total_dis, "max_dis", max_dis, "count", count, "val", val,
   "centroid", whattype(centroid)):
  for y from 1 to n_cluster do
    centroid_rec[y] := convert(centroid_rec[y],'list'):
  end do: print("centroid_rec[1]", centroid_rec[1], whattype(%)):
  centroid_rec := rtable(1..n_cluster, 1..85, convert(centroid_rec,'list')):
  ExcelTools:-Export(centroid_rec, "clustering.xls", "centroid", "A1");
end proc:
cluster(PosPatterns, 174);
#the end of maple code
```

VITA

DAOPING YU

Education:	B.S. Information Management & Information Systems, South China Normal University, Guangzhou, China 2006 M.S. Mathematics, East Tennessee State University, Johnson City, Tennessee 2010
Professional Experience:	Software Engineer, Pujaa, Inc. in Beijing & Qingdao, Beijing & Qingdao, China, 2006–2008 Teaching Assistant, Tutor, East Tennessee State University, Johnson City, Tennessee, 2008–2010