NOVELTY DETECTION FOR FUNCTION APPROXIMATION

By

ARJPOLSON PUKRITTAYAKAMEE

Bachelor of Engineering

Chulalongkorn University

Bangkok, Thailand

1997

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2001

NOVELTY DETECTION FOR FUNCTION

APPROXIMATION

Thesis Approved:

Martin T Hagan

Thesis Adviser

[illegible]

[illegible]

[illegible]

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Martin T. Hagan, for his moral support, guidance, patience, and friendship throughout my graduate studies. I also would like to thank my other committee member, Dr. R.G. Ramakumar and Dr. Keith A. Teague, for their useful suggestions.

I wish to sincerely thank again Dr. Martin T. Hagan for his financial support and his generosity to let me experience this project.

Finally, I would like to express my special appreciations to my parents and my sister for their loving support and strong encouragement at the difficult times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

One of the key tasks for neural networks is function approximation or plant identification. The ability of neural networks in these applications has been well documented. However, a major factor that limits the usage of neural networks is the difficulty to identify the reliability of the neural network outputs. The procedure to determine whether or not a neural network generates credible results is known as network validation or novelty detection. Our objective is to compare the performance of existing novelty detection methods as well as to find improvements to these techniques.

We will start our work by reviewing the general structure of neural networks for function approximation. A simple example will be used to show the ability of the neural network in performing this task. We will then demonstrate the main problem of function approximation using this example.

After the limitation of neural networks is shown, we will introduce existing novelty-detection algorithms. The simple example will again be utilized to exhibit the capability of each novelty detector.

We will propose a procedure to improve the performance of some algorithms, followed by a demonstration of the ability of the modified novelty detector via the example.

We will show through the simple example and our real world applications that the modified methods result in improvements in novelty detection.

Let us now outline the flow of this work. Chapter 2 will serve to review neural network background material, starting from basic concepts to the general structure. One of the main objectives of this chapter is to introduce common notation used in later chapters. We will finish this chapter by introducing the use of neural networks as function approximators.

In Chapter 3, we will introduce an existing algorithm known as the neural tree. We will explain how we can use this algorithm as a novelty detector. A simple example will be used to illustrate the capability of this method. We end this chapter by introducing a measure of algorithm performance.

The most widely-used method for novelty detection will be described in Chapter 4. The method is known as the Gaussian kernel estimator, in which the probability density function will be involved. A technique for improving the performance will be described. It will be followed by simulation results.

Chapter 5 will describe another algorithm, the autoassociative multilayer perceptron. The example will demonstrate the capability of this method. A technique for improving performance will be described, followed by an application to the simple example.

Then, in Chapter 6, the simplest method, known as the minimum distance algorithm, will be introduced. An example will be used to show the efficiency of this method. An improvement to this algorithm, which is called the minimum weighted distance, will be proposed, followed by an example. The mathematical framework that leads us to the idea of the minimum weighted distance algorithm will be discussed as well.

In Chapter 7 we explain the idea of principal components and outlier detection, and explain a new technique that combines the knowledge of outlier detection with the minimum distance algorithm. This is followed by an example.

Chapter 8 will be devoted to applying the techniques explained in Chapter 3, 4, 6, and 7 to real world data. We will provide short explanations of these methods within this chapter. A performance comparison of these algorithms on real world data will be summarized at the end of this chapter.

A summary of the main results and contributions of this thesis, followed by recommendations for future work will be contained in Chapter 9.

CHAPTER 2

LITERATURE REVIEW

Introduction

This chapter describes the fundamental concepts of the neural network and introduces the associated notation. It will start with an analysis of a single-input neuron, which is the smallest and the most basic component in the neural network, and this leads to more complicated architectures. The simplest architecture has a single layer of neurons. The more complex architectures have several layers. The multiple layer network has been widely used to perform pattern recognition. However, this chapter will focus on how the multiple layer network can be used for function approximation.

After we introduce the basic neural network concepts, the concept of novelty detection in neural networks will be explained. We will define what novelty detector is and will describe when it will be applied and how it relates to neural network function approximators.

Single-Input Neuron

A neuron is the smallest processing unit in the neural network. It has a scalar input p and a scalar output a. The input is multiplied by the scalar weight w and then added to the scalar bias b. Now, the output of the summer, which is $n = wp + b$, will be fed to the input of the transfer function f. The output of the neuron is described by the following equation.

$$a = f(n) = f(wp + b) \tag{1}$$

The weight and bias of a neuron can be any real value. They are adjustable parameters and are adapted by some learning rule.

The transfer function of the neuron can be a very simple linear function, or it can be a more complex nonlinear function such as the hard limit function (hardlim) or the logarithmic sigmoid function (logsig). The transfer function will be specifically chosen, depending on a particular problem that the neuron is going to solve.

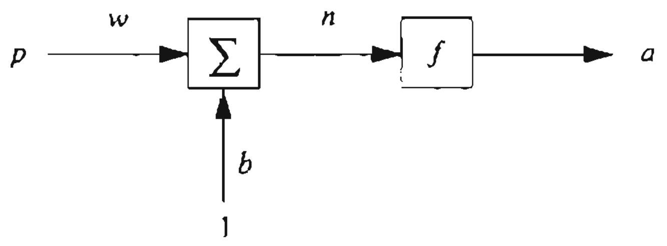

Figure 1 A Single Neuron

As seen in Equation (1), the weight w controls the slope of the neuron output and the bias b causes a translation in the neuron output. Next, we will give simple examples to demonstrate how the weight and bias affect the output of the neuron. The insights provided by these examples will be helpful when applying neural networks to the practical tasks that will be described later.

Let the transfer function f be the pure linear function $purelin$. The relation between the neuron input and output is

$$a = f(wp + b) = wp + b \qquad (2)$$

The following figure demonstrates this relationship for $w = 1$ and $b = 0$.

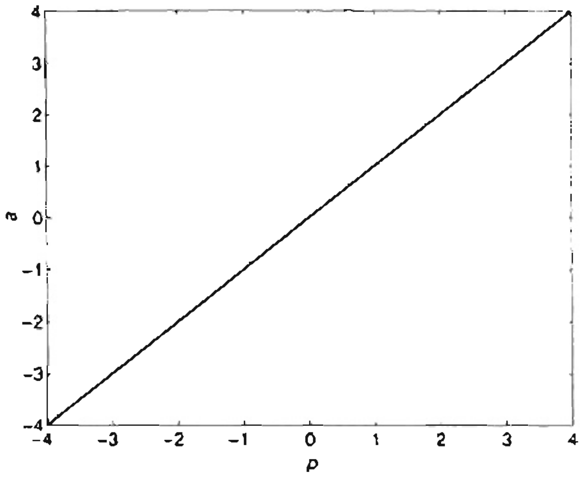

Figure 2 Pure Linear Transfer Function

The following figures show how changes to the weight and bias affect the response. Figure 3 illustrates the effect of changing the weight from 1 to 3. The effect of varying the bias is shown in Figure 4.

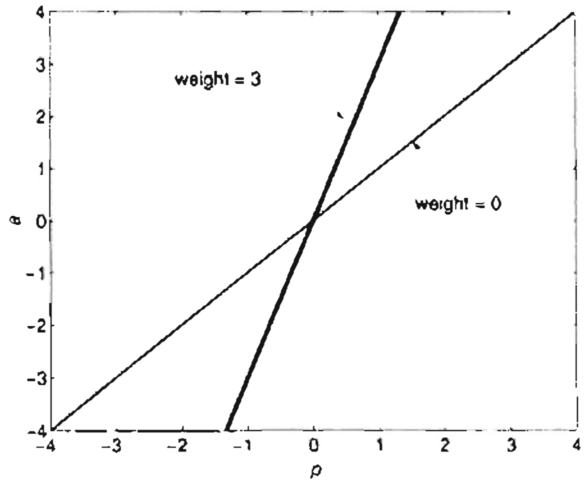

Figure 3 Effect of Weight

Note that when setting $b = 2$, the graph will shift to the left by $-\frac{b}{w}$. This is shown in Figure 4, where the graph shifts to the left by 2 for $b = 2$ and $w = 1$.

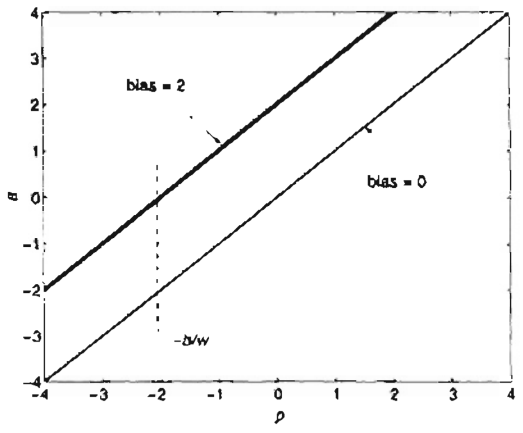

Figure 4 Effect of Bias

In the next example, we let the transfer function f be the hyperbolic tangent sigmoid. The hyperbolic tangent sigmoid function is a monotonically increasing function shown in Figure 5. This function has been extensively used for function approximation problems, as will be explained later in this chapter. The formula for this function is

$$f(n) = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}} \tag{3}$$

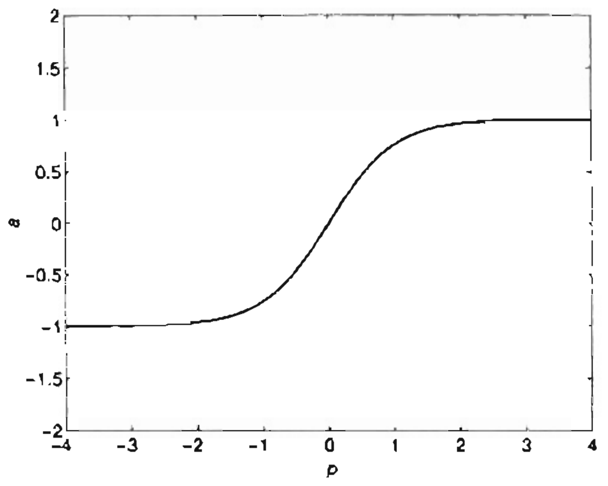

Figure 5 Hyperbolic Tangent Sigmoid Function

The function output is bounded to the range of [-1,1], regardless of how large the input is. The input-output relationship is almost linear in a small region near zero. The effect of weight and bias are demonstrated in the following figures.

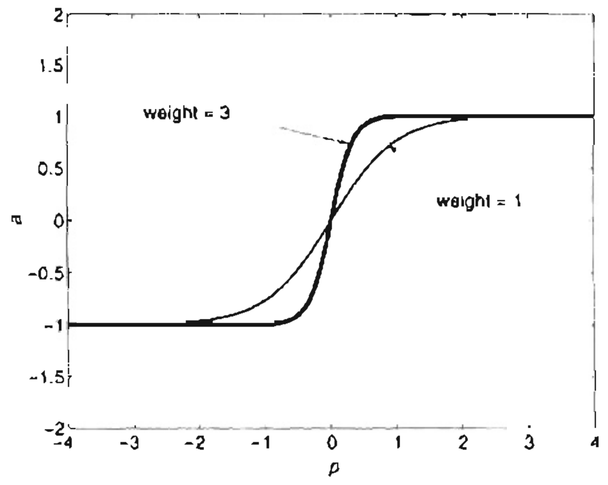

Figure 6 Weight Effect

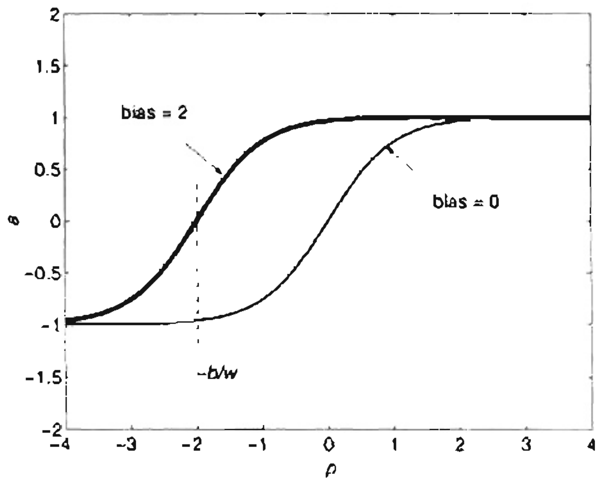

Figure 7 Bias Effect

As in the case of the pure linear function, increasing the weight value increases the slope of the graph. The bias shifts the center of the graph to the point $-\frac{b}{w}$.

Multiple-Input Neuron

Commonly, a neuron can have several inputs. R inputs connecting to a neuron are expressed as $\mathbf{p}^T = \begin{bmatrix} p_1 & p_2 & \cdots & p_R \end{bmatrix}$. A neuron with R inputs is shown in Figure 8.

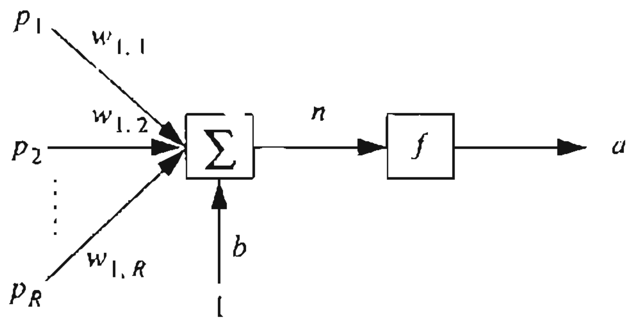

Figure 8 Multiple-Input Neuron

The net input for this neuron is $n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$. Notice that the number of weights equals the number of input elements R. The first weight subscript indicates the destination neuron whereas the second subscript indicates the input element. For example, $w_{1,5}$ means that this weight represents the connection to the first neuron from the fifth input p_5. We can write the net input n in matrix form:

$$n = \mathbf{Wp} + b \tag{4}$$

Then the neuron output a is written as:

$$a = f(n) = f(\mathbf{Wp} + b) \tag{5}$$

Network Architectures

A single multiple-input neuron may not be able to perform all tasks. Several neurons, operating in parallel, are called a "layer". A layer of neurons will be discussed next, followed by a discussion of multiple layers of neurons.

A Layer of Neurons

The architecture of a single layer of S neurons is shown in Figure 9. The architecture possesses R inputs, and S outputs. Note that the number of inputs R is not necessarily equal to the number of outputs S.

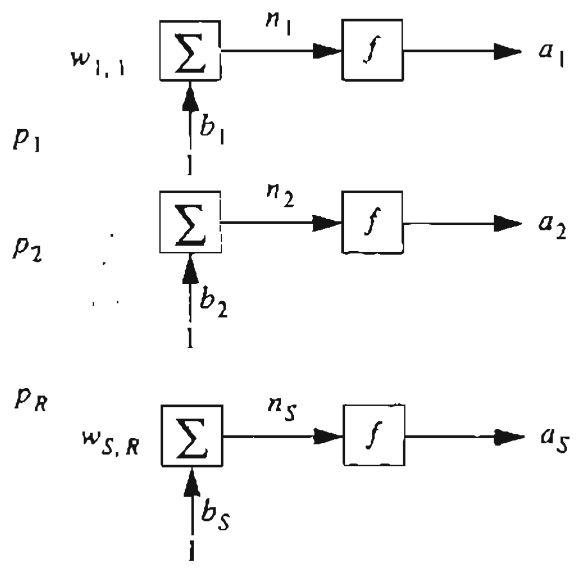

Figure 9 A Single-Layer Network

The input vector $\mathbf{p}$ is connected to each neuron through the weights, which will be defined as $\mathbf{W}$.

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \tag{6}$$

A layer of multiple-input neurons is sufficient to solve some problems, such as linear adaptive filtering, or simple pattern recognition tasks, etc. However, this structure is not adequate for approximating arbitrary functions. The more general multiple layer architecture will be introduced in the next section.

Multiple Layers of Neurons

This network consists of several layers, connected in series. Each layer has the same structure as noted earlier. Because this architecture has several layers, the superscript will be used to identify the layer number. Consequently, a weight can be written as $w^k{}_{i,j}$, where k denotes the layer to which the weight belongs. A three layer network is shown in Figure 10.

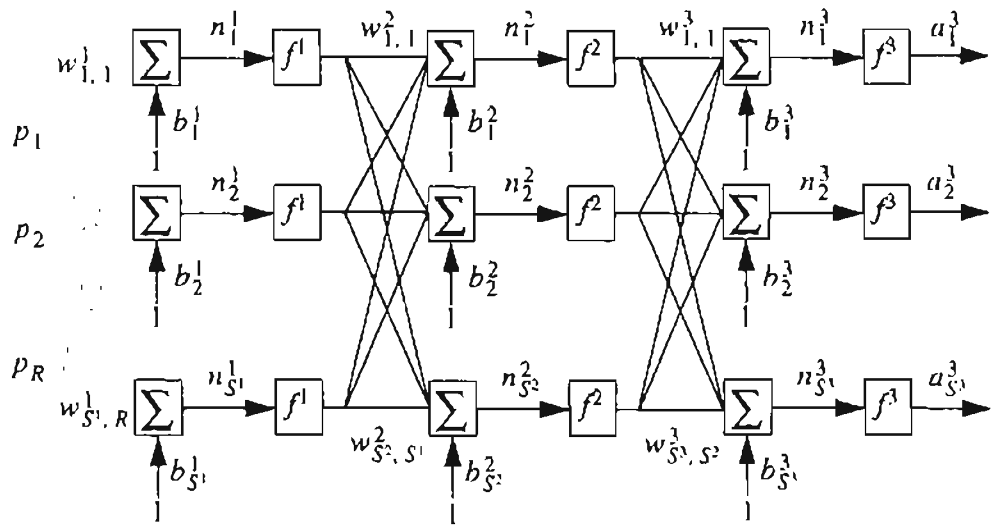

Figure 10 Three-Layer Network

The first layer of neurons has R input elements, S^1 neurons and S^1 outputs. The second layer has S^1 input elements, S^2 neurons, and S^2 outputs. Likewise, the third layer contains S^2 input elements, S^3 neurons and S^3 outputs. S^1, S^2 and S^3 are the number of neurons in the first, second and third layer, respectively. In the general case the number of layers is arbitrary.

The weight in the first layer is expressed by the notation of $\mathbf{W}^1$. This layer weight, $\mathbf{W}^1$, is the matrix with dimension S^1xR. Similarly, $\mathbf{W}^2$ and $\mathbf{W}^3$ are the weights in the second and third layer, respectively. $\mathbf{W}^2$ has dimension S^2xS^1, while the size of $\mathbf{W}^3$ is

$S^3 x S^2$. Generally, the weight matrix at layer k, $\mathbf{W}^k$, has dimension $S^k x S^{k-1}$. A network comprised of R input elements, S^1, S^2, and S^3 neurons will be referred to as an $R - S^1 - S^2 - S^3$ network.

The output of each layer is the input to the next layer, and is denoted $\mathbf{a}^k$. For instance, the output of the second layer is denoted $\mathbf{a}^2$. The last layer of the network is called an "output layer". The other layers, which are internally connected between the input vector and the output layer, are commonly called "hidden layers".

This structure of the network is powerful enough to estimate arbitrary functions, as will be shown below.

Function Approximation

Multilayer networks have been broadly used as function approximators. For example, in control systems neural networks are used to mimic plants in order to get proper feedback signals. They have been widely used to compensate for channel fading in telecommunication systems. Adaptive filtering is another application employing neural networks. The function approximation abilities of neural networks are discussed below.

The multilayer network has several layers of neurons with S^k neurons in the k^{th} layer. Normally, the number of layers of a network, N, is two or, at most, three. The number of neurons in the hidden layer is heuristically specified, and depends on how complex the function is. The number of neurons in the output layer depends on the number of outputs in the desired function. Though it seems that the more neurons in the hidden layers, the better a network can perform, it is possible that an overly complex network can overfit on a finite training set. Thus, the appropriate number of neurons is dependent on the individual problem.

Suppose that the network is a two-layer $1-2-1$ network, as shown in Figure 11. Let's assume that the hyperbolic tangent sigmoid is used in the first layer, and the second layer transfer function is linear.

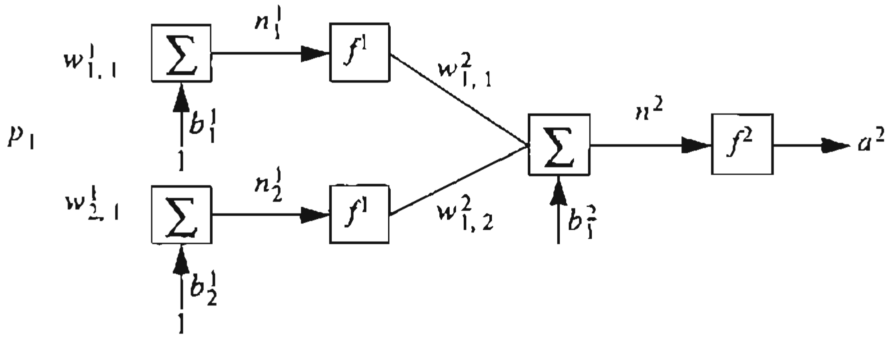

Figure 11 An Example of Network for Function Approximation

The input/output relation is shown in the following equation.

$$\begin{aligned}
a^2 &= f(p) \\
&= f^2(\mathbf{W}^2 f^1(\mathbf{W}^1 p + \mathbf{b}^1) + b^2) \\
&= f^2\left(\begin{bmatrix} w^2_{1,1} & w^2_{1,2} \end{bmatrix} f^1\left(\begin{bmatrix} w^1_{1,1} \\ w^1_{2,1} \end{bmatrix} p + \begin{bmatrix} b^1_1 \\ b^1_2 \end{bmatrix}\right) + b^2_1\right) \\
&= \begin{bmatrix} w^2_{1,1} & w^2_{1,2} \end{bmatrix} f^1\left(\begin{bmatrix} w^1_{1,1} \\ w^1_{2,1} \end{bmatrix} p + \begin{bmatrix} b^1_1 \\ b^1_2 \end{bmatrix}\right) + b^2_1
\end{aligned} \tag{7}$$

where $f^2(n) = purelin(n) = n$ and $f^1(n) = tansig(n) = \dfrac{e^n - e^{-n}}{e^n + e^{-n}}$.

We have trained this network to approximate the function $f(p) = p^4$ over the range $p \in [-1, 1]$. After training, the following weights and biases were obtained:

$$\mathbf{W}^1 = \begin{bmatrix} 3.03 \\ -3.03 \end{bmatrix},\ \mathbf{b}^1 = \begin{bmatrix} -3.2 \\ -3.2 \end{bmatrix},\ \mathbf{W}^2 = \begin{bmatrix} 1.22 & 1.22 \end{bmatrix} \text{ and } \mathbf{b}^2 = \begin{bmatrix} 2.42 \end{bmatrix} \tag{8}$$

Figure 12 shows the network response as well as the target values p^4. The network outputs are very close to the targets. This is just an example that a multiple-layer network can be used to approximate arbitrary functions.

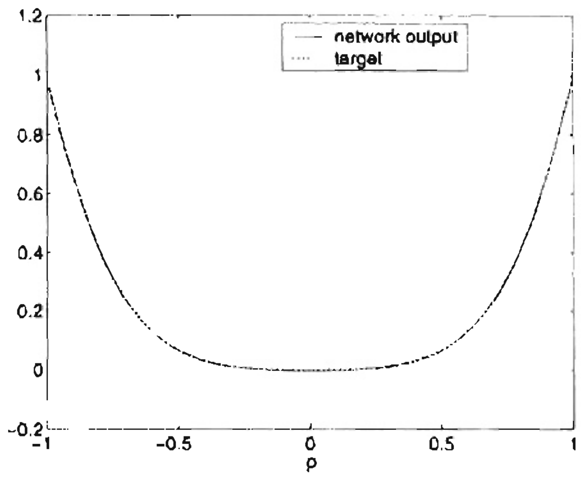

Figure 12 Network Outputs and Targets

There are many algorithms, such as the Levenberg-Marquardt algorithm, the Bayesian regularization algorithm, and the gradient descent algorithm, to train the weights and biases. Such algorithms are in general called "backpropagation algorithms", and can be found in many books and papers, such as [HaDeBe96], [HaMe94], [Fahl89], etc. They generally minimize errors between targets and network outputs. The mathematical derivations of these algorithms will not be included here since they are not the focus of this project. Note that the error minimization process in a neural network is also called "training".

Now that we have discussed the basic concepts, we will next introduce the concept of "novelty detection".

Novelty Detection in Neural Networks

Introduction

Recall from the last section that a network, given a sufficient number of neurons, can be trained to approximate arbitrary functions. However, the performance of the trained network will be dependent upon the data set that was provided during the training period. In other words, a training algorithm minimizes the errors between the targets and network outputs, for the training data set. When the network is subjected to data that were not in the training (usually called "testing" data), what will the outputs of the neural network look like? Will the network still approximate the function accurately? The following section will describe this problem.

Consider the previous example in which the network was trying to approximate the function $F(p) = p^4$. The inputs p fed into the network during training process had the range of [-1,1]. The outputs of the network eventually looked similar to the targets for this range of input. Now, suppose that new inputs that are between [-2,2] are applied to the network. The results of this test are shown in Figure 13.

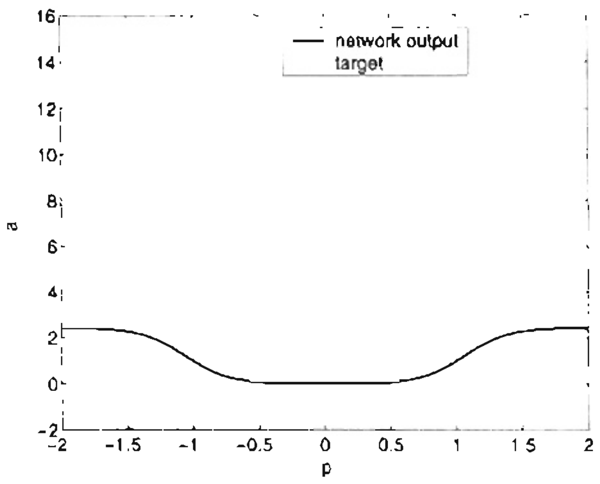

Figure 13 Network Outputs and Targets on Testing Data

In the region of input data between [-1,1], which is the same as that of training data, the network performs well, as expected. However, the network performs poorly outside this region, producing very large errors between targets and outputs.

Unfortunately, in real world applications, it is difficult to tell when an input vector falls outside the range of the inputs in the training set. Should we count on the network outputs? The next section will describe what we can do to alleviate such concerns.

What is Novelty Detection?

As shown in Figure 13, the network did not accurately approximate the function outside the training range. Therefore, we need to be able to identify when an input vector falls outside the range of the training data. This is called "Novelty Detection". In other words, novelty detection methods should have the capability to detect input data that are "abnormal". We expect that this data will generate large errors. If we can detect whether or not inputs are unusual, confidence in the network outputs would be stronger. Novel data (which may cause considerable errors) would be rejected, whereas standard data (giving desirable outputs) would be accepted.

Considering the example shown above, some might think that novel data could be easily distinguished by picking up data points out of the training data range (bounded between [-1,1] in this case). Then the rest, which occupied the interval [-2,-1] and [1,2], would be novel points. In practical applications, the dimension of the input to the network will be much larger than that of the input (one) shown in the example, making it much harder to detect these "unseen" data points. When the dimension of the input is large, the distinction between interpolation and extrapolation will be much more ambiguous. Therefore, differentiating the boundaries between normal and abnormal data is much more difficult, making it harder to decide whether a specific input should be accepted or rejected. In the next section, some algorithms for novelty detection will be introduced.

Algorithms for Novelty Detection

As discussed thus far, whether or not the output of the network for a particular test input should be relied on is dependent upon the difference between the test input and the inputs in the training data. For example, in the above example, if an input was in between the interval [-1,1], a corresponding output would be accepted, since the network was trained to perform well in this interval. In contrast, if an input was out of the range [-1,1], a corresponding output would be disregarded. In other words, inputs will be identified when they are not close to any training inputs. This concept leads to some existing algorithms, such as the neural tree [Mart98] and the minimum-distance computation. They are similar, in that they will flag any data as "abnormal" when the input vector is far from any training data.

The performance of the minimum-distance algorithm could be improved by applying some weighting factors, which will be explained in Chapter 6. The algorithm that uses the Gaussian kernel estimator model [Bish94], as elucidated in Chapter 4, works by means of computing the probability of the existence of an input vector in the training data near the test input vector.

Unlike the algorithms described above, we can also use autoassociative multilayer perceptron for novelty detection [FrGoPr96]. This method, which will be explained in Chapter 5, is used to recognize input vectors in the training data. An additional neural network is trained to memorize what the training inputs look like. Finally, the combination of principal component analysis and newly-defined minimum-distance computation will be discussed in Chapter 7.

Summary

In this chapter, the multilayer neural network was introduced. We described the ability of the multilayer network to operate as a very general function approximator. These multilayer function approximators are very good at interpolating between data points on which they were trained. However, they are not good at extrapolating outside the training set. The remainder of this thesis will present algorithms that can be used to detect when a network is performing an extrapolation.

CHAPTER 3

NEURAL TREE ALGORITHM

Introduction

The neural tree algorithm, originally proposed by [Mart98], is the combination of an unsupervised learning competitive network and a binary tree. The method takes advantage of fast learning, because it only deals with scalar information, unlike competitive networks that require matrix computation. Therefore, the algorithm generally uses less training time than competitive learning.

In this chapter, we will start by defining the notation used in this algorithm and by explaining how they relate to the data distribution. Then the process of learning the data distribution, i.e. training a tree, will be explained. After the tree is trained, the procedure for using a neural tree for novelty detection will be explained, and simple computer simulations will be shown. Finally, we will explain how to measure the effectiveness of a novelty-detection algorithm.

Neural Tree Algorithm

The neural tree hierarchically partitions a q-dimensional space into cells, separated by hyperplanes orthogonal to coordinate axes. As with any common searching tree, the neural tree contains nodes. The node that is at the top of the tree is called the root node, while the others are called child nodes, or leaf nodes. Each node stores a "weight" w_{ij}. The i^{th} hyperplane decision boundary is orthogonal to the j coordinate axis, and the position of the hyperplane is at w along axis j. Below the last level of children nodes are the "cells". A cell represents a certain region in the hyperspace that is hierarchically partitioned by the weights. Note that an N-cell tree has $N-1$ nodes that need to be trained. Figure 14 is an example of a 4-cell tree structure (having 3 nodes). The "circles" represent nodes, and "rectangles" represent partitioned cells. From Figure 14, the 2-dimensional hyperspace can be divided into four cells as illustrated in Figure 15.

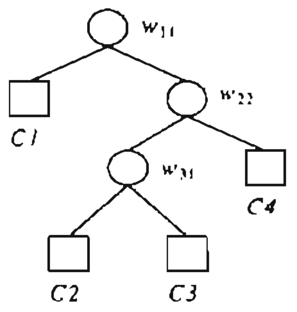

Figure 14 4-cell Tree

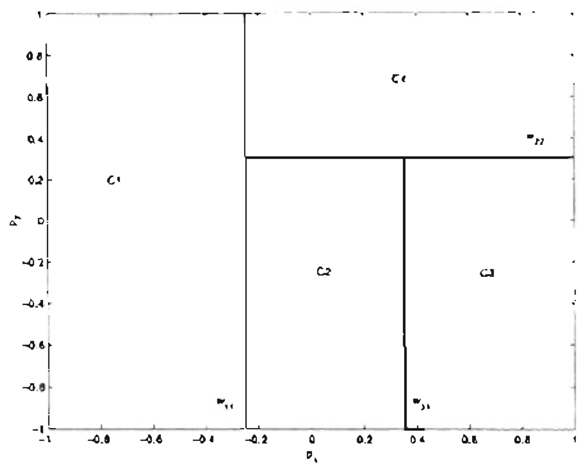

Figure 15 Partitioned Cells in Hyperspace

The cell $C1$ occupies the region less than weight w_{11} along coordinate axis 1, $C2$ is the region between w_{11} and w_{31} along axis 1 and less than w_{22} along axis 2. Similarly, $C3$ is restricted to the area greater than w_{31} along axis 1 and less than w_{22} along axis 2. $C4$ occupies the area greater than w_{11} along axis 1 and greater than w_{22} along axis 2. Therefore, the weights w_{ij} determine the boundaries of cells. In the 2-dimensional hyperspace, any data $\mathbf{p}_k = \begin{bmatrix} p_{k1} & p_{k2} \end{bmatrix}^T$ can be located in a certain cell by first comparing its j^{th} element where $j \in 1, 2$, p_{kj}, with the root node w_{1j}. If $p_{kj} > w_{1j}$, the data $\mathbf{p}_k$ will be sent to the right child node. Otherwise, it will be sent to the left child node. The scalar comparison will keep going until there is no node left to be compared. For example, from Figure 14 and Figure 15, assume that $p_{k1} < w_{11}$, the data $\mathbf{p}_k$ will be sent to the left child node. However, since there is no left child node under the root node, the scalar comparison is stopped and $\mathbf{p}_k$ is belong to cell $C1$.

The following section will describe the algorithm for adjusting the weights. Details and proofs can be found in [Mart98].

How the algorithm works

In order to train a tree, we need to initialize it first. The initialization process can be done in several ways. We can randomly select two values, j and w, for initializing a node. However, the initialized tree may be a very poor fit to the data distribution, which can make training difficult. Alternatively, all $N-1$ nodes of an N-cell tree can be initialized by using N random samples from the training data. This method will be sensitive to the selection of the N sampled data points. To reduce the sensitivity to the sampled data, a method called *norminal initialization* can be applied, which considers the distribution of the entire training data set before constructing a tree. This is explained in [MaRoGi85], and [RiGr91]. In this thesis, we will initialize the tree by sampling the training data set.

We first need to find an axis to which a decision hyperplane will be orthogonal, and then we need to compute the location of the decision hyperplane on the axis. To locate the axis, we compare the element of an incoming data vector and a previous vector that is located in the same partitioned cell. The element of the vector that shows the biggest difference is selected as the axis that will be orthogonal to the hyperplane. Then, the location of the hyperplane on the axis is found by computing the mean of the corresponding element of the two data points. The initialization process will continue until the desired number of cells is reached. Note that when the process is done, each partitioned cell will contain only one data point. An example of the initialization process is given below.

Assume that a four-cell tree is to be trained. Four sample vectors must be drawn from the training data, and suppose that they are

$$\mathbf{d}_1 = \begin{bmatrix} -0.50 \\ -0.80 \end{bmatrix}, \mathbf{d}_2 = \begin{bmatrix} -0.40 \\ -0.30 \end{bmatrix}, \mathbf{d}_3 = \begin{bmatrix} 0.80 \\ 0.70 \end{bmatrix}, \mathbf{d}_4 = \begin{bmatrix} 0.70 \\ 0.75 \end{bmatrix} \quad (9)$$

First, the samples $\mathbf{d}_1$ and $\mathbf{d}_2$ will be compared in order to place the root node. The difference between the samples is $|\mathbf{d}_1 - \mathbf{d}_2| = \begin{bmatrix} 0.1 & 0.5 \end{bmatrix}^T$. Therefore, we will place the hyperplane on the p_2 axis at the location $\frac{(-0.8) + (-0.3)}{2} = -0.55$, making the root node weight $w_{12} = -0.55$. Therefore, $\mathbf{d}_2$ is in the cell above the boundary $p_2 = -0.55$, and $\mathbf{d}_1$ is in the cell below the boundary.

Next, we will apply $\mathbf{d}_3$ to the initialized node by comparing the second element of $\mathbf{d}_3$, 0.7, with w_{12}. It turns out that $0.7 > -0.55$, and thus $\mathbf{d}_3$ is in the cell above the boundary $p_2 = -0.55$, which is the same region as $\mathbf{d}_2$. Therefore, we will compute the difference between $\mathbf{d}_2$ and $\mathbf{d}_3$, $|\mathbf{d}_2 - \mathbf{d}_3| = \begin{bmatrix} 1.2 & 1.0 \end{bmatrix}^T$. Since the first element is greater than the second, we will set up a hyperplane orthogonal to the p_1 axis at the location $\frac{(-0.4) + (0.8)}{2} = 0.2$, thereby making $w_{21} = 0.2$.

Now, $\mathbf{d}_1$ is below the boundary $p_2 = -0.55$, $\mathbf{d}_2$ is above $p_2 = -0.55$ and less than $p_1 = 0.2$, while $\mathbf{d}_3$ is above $p_2 = -0.55$ and greater than $p_1 = 0.2$. We will continue the initialization process by applying $\mathbf{d}_4$ to the initialized tree by finding the cell that

$\mathbf{d}_4$ falls into. Since $0.75 > -0.55$ and $0.70 > 0.2$, $\mathbf{d}_4$ is in the same region as $\mathbf{d}_3$. We will calculate the difference again, $|\mathbf{d}_3 - \mathbf{d}_4| = \begin{bmatrix} 0.10 & 0.05 \end{bmatrix}^T$. We therefore put the new hyperplane on the p_1 axis at the location $\frac{(0.8) + (0.7)}{2} = 0.75$, resulting in $w'_{31} = 0.75$. We now have the desired number of cells, and we therefore stop the initialization process. Figure 16 shows how the hyperspace is divided into 4 partitioned cells.

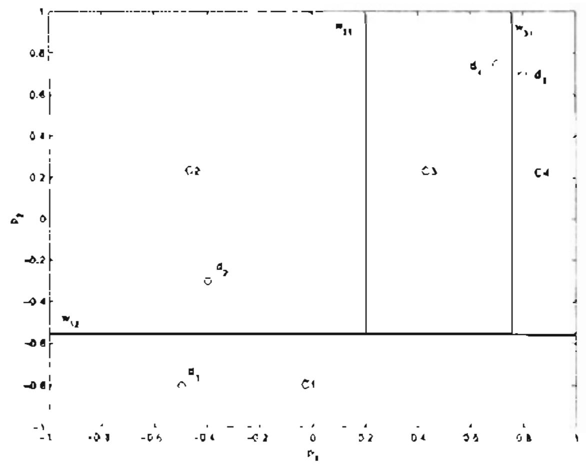

Figure 16 Initialized Divided Hyperspace

The corresponding tree structure that will represent the divided hyperspace is shown Figure 17.

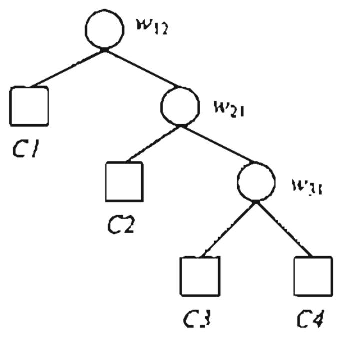

Figure 17 Tree Structure

After we have the initialized tree, the next step is to train those weights contained in the tree. Suppose that a vector $\mathbf{p}_k = \begin{bmatrix} p_{k1} & p_{k2} & \cdots & p_{kj} & \cdots & p_{kq} \end{bmatrix}^T$ flows to node i, which stores weight w_{ij}, at layer K. If $p_{kj} \le w_{ij}$, the vector $\mathbf{p}_k$ will be sent to the left child node directly under node i. Likewise, if $p_{kj} > w_{ij}$, the vector will be forwarded to the right child node directly under node i. Simultaneously, weight w_{ij} will be updated according to the following equation.

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)\left(\frac{1_{R(i)}(p_{kj})}{n_r(i)} - \frac{1_{L(i)}(p_{kj})}{n_l(i)}\right) \tag{10}$$

where $\eta(t)$ is the learning rate at time t with

$$1_{L(i)}(p_{kj}) = \begin{cases} 1 & ;\ p_{kj} \in L(i) \\ 0 & ;\ p_{kj} \notin L(i) \end{cases} \tag{11}$$

and

$$1_{R(i)}(p_{kj}) = \begin{cases} 1 & ;\ p_{kj} \in R(i) \\ 0 & ;\ p_{kj} \notin R(i) \end{cases} \tag{12}$$

where $L(i)$ and $R(i)$ are the unions of all partitioned cells belonging to the left or right subtree of node i, respectively. The terms $n_l(i)$ and $n_r(i)$ are the number of partitioned cells associated with the left and right subtrees under node i. Note that the learning rate $\eta(t)$ can be adjusted with time. Its value will be reduced during training so that the algorithm will converge.

The algorithm is a top-to-bottom learning method; training the root node first and then down to the lowest level of children nodes. We will apply the next input vector to the root node and the method will be repeated. Notice that as long as data are applied to the tree, the boundaries of the cells will gradually move in accordance with the weight update in Equation (10). Therefore, the location of the hyperplanes of the partitioned cells at this moment are still in transition. The tree will learn the data distribution until the final input comes in. After training is complete, all the nodes in the tree will contain fixed weight values. We can refer to a specific cell in the final tree by using the following notation:

$$C \in [(a_1, b_1), (a_2, b_2), \ldots, (a_q, b_q)] \tag{13}$$

The meaning is that the cell C occupies the region from a_1 to b_1 in the first coordinate axis, from a_2 to b_2 in the second axis and so on. For example, recall Figure 15, cell $C1$ represents the region $[(-\infty, w_{11}), (-\infty, \infty)]$. The quantization range of the first coordinate axis is defined as $|a_1 - b_1|$, and is likewise to any other dimension. Notice that the quantization range is now unmeasurable. The following section will describe how to make it measurable and how the neural tree algorithm relates to novelty detection. The convergence of the algorithm was proven in [Mart98].

Application to Novelty Detection

After training is complete, the neural tree contains fixed boundaries, which partition the hyperspace into different cells. We would like to use the trained tree to detect future inputs that are unlike the inputs used for training (i.e., we are looking for novel inputs). There are two approaches to identify these novel data. The first approach is to compute probability density of each cell. If the probability density of a cell is low, it indicates that data within that cell is less likely to occur. Therefore, any data falling in a cell having low probability density will be more likely to be rejected as novel data. For the second approach, data will be identified as novel when it is outside of a cell. We will explain these two approaches in the following paragraphs.

For the first method (identifying novel data by estimating the probability density), Martinez [Mart98] suggested that the probability of a vector falling into a cell is equal to $\frac{1}{N}$, where N is the number of cells. Therefore, in order to obtain the probability density, we need to divide the probability by the cell area in the two dimensional space (or volume in high dimensional spaces). That means that if the volume of a cell is large, the probability density of a vector falling into the cell will be low, thereby making the data in the cell prone to being rejecting as novel. As we mentioned at the end of the last section, some cells occupied infinite area (in the two dimensional data). We need to limit the occupied region of a cell. Martinez suggested using the maximum and minimum value of the training data. That means that the unmeasurable value will be replaced by -1 or 1 in the normalized hyperspace. For example, in Figure 15, after training the tree, cell $C1$ will occupy the region

$[(-1, w_{11}), (-1, 1)]$. Then, the area or volume of cell $C1$ that was infinite is now computable. Therefore, the probability density of cell $C1$ will then be $\left(\frac{1}{4}\right)/(|-1-w_{11}| \times |-1 - 1|)$. As we explained earlier, the density of a cell will be low if the volume of the cell is high. And, any data falling in the cell will be more likely to be rejected as novel data since the chance of such data occurring is small.

For the second approach, novel data will be identified when they are out of the cell boundaries. In this case, the maximum or minimum values of training data in a cell may be used to limit the cell size in every dimension. Note that we may use the other values such as the maximum or minimum plus some margins to limit cell size. A further study of how to choose an appropriate margin may be required. However, we will use zero margin in this thesis. By applying this technique, every cell size will be fixed and finite. After we perform this procedure, the tree can be used for novelty detection. After an input vector follows the tree structure and is located at a cell, the algorithm determines whether the input vector is more than a certain distance beyond the cell boundary. Abnormalities are identified whenever input vectors are outside of their cells regardless of which boundaries they break.

We will show the simulation results of these two cases for novelty detection in the following example.

Simulation of a simple example

We will begin this section by demonstrating the capability of a neural network for function approximation. Then, novelty detection using the neural tree algorithm with the second approach we discussed in the last section will be applied. We will also show the simulation results using the first approach (density estimation) and will introduce the problem of utilizing this approach for novelty detection.

The following is a two dimensional example used to demonstrate novelty detection employing the neural tree algorithm. This example will be used to demonstrate the other algorithms as well.

In the example, a two-layer feed forward neural network, with 40 neurons in the hidden layer and one neuron at the output layer, was trained to approximate the following function.

$$\begin{aligned} t &= F(\mathbf{p}) \quad ;\forall \mathbf{p} \\ &= F\left(\begin{bmatrix} p_1 & p_2 \end{bmatrix}^T\right) \\ &= \frac{\sin(10\sqrt{p_1^2 + p_2^2})}{10\sqrt{p_1^2 + p_2^2}} \end{aligned} \tag{14}$$

Figure 18 is a graph of the function $F(\mathbf{p})$.

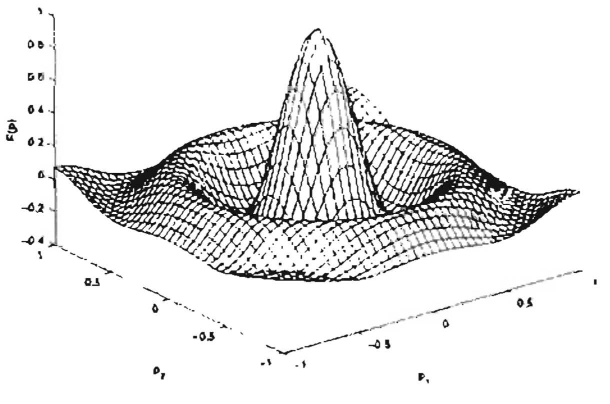

Figure 18 Function $F(\mathbf{p})$

Figure 19 shows where the training data is located.

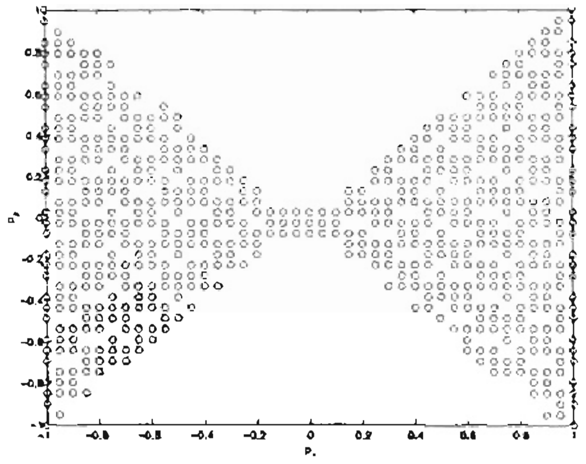

Figure 19 Training data

Figure 20 shows the error on the 638 training points from Figure 19. We can see that the network provides an accurate approximation to the function for all training points.

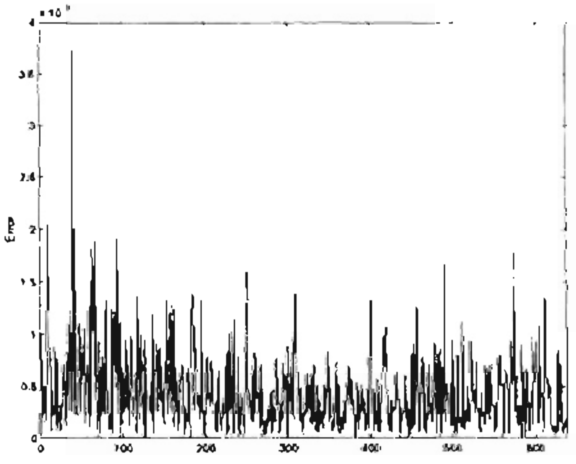

Figure 20 Error between target and network output after training

Figure 21 shows the error between the output of the function and the output of the function approximator. We can see that the errors outside the training data are larger than the errors within the training data region.

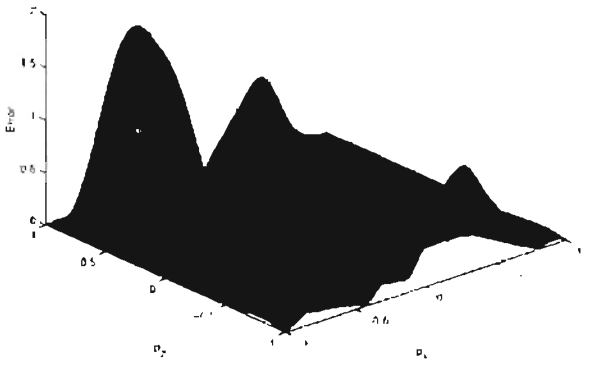

Figure 21 Error between target and network output in the normalized hyperspace

Assume that we are going to test the trained network using 437 new data points. The testing and training points are shown in Figure 22.

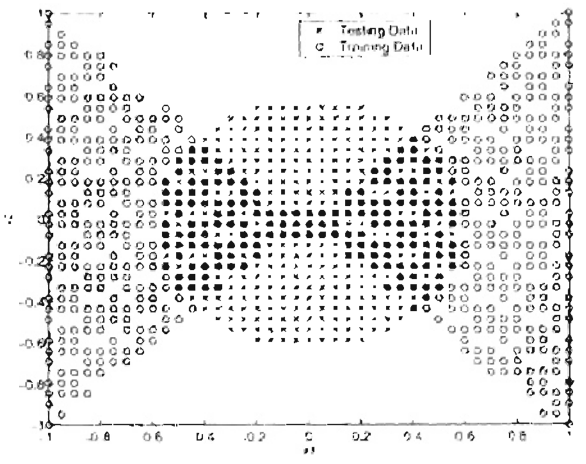

Figure 22 Testing and Training Data

Let's apply the 437 test inputs, $\mathbf{p}'$, to the trained network. Figure 23 shows the errors on the testing data.

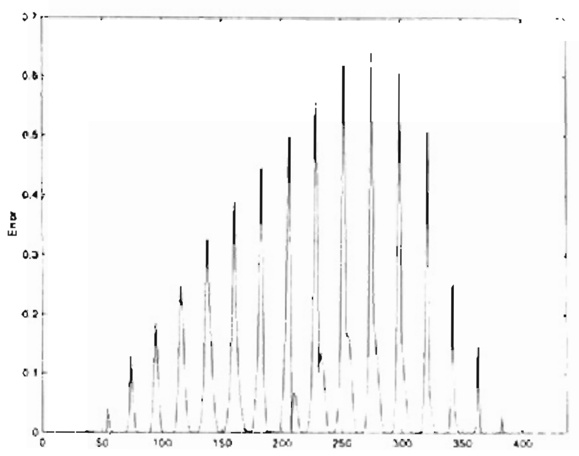

Figure 23 Error of the testing data

The large errors that occur in Figure 23 are for inputs outside the range of the training data. We would like to use a neural tree to detect inputs outside the range of the training data. This would enable us to determine the reliability of the multilayer network output. In this example we use the tree described in Figure 24 and Figure 25.

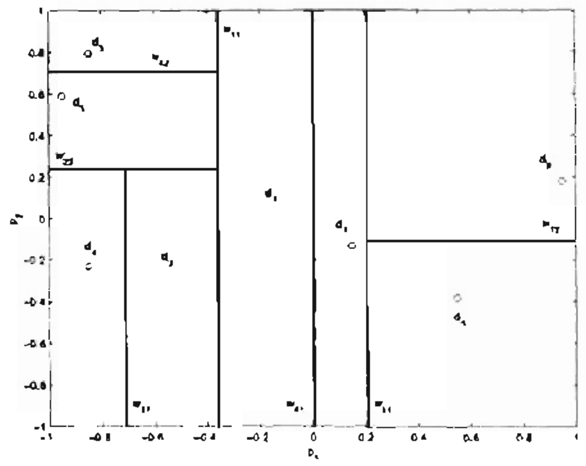
Figure 24 Initialized Cells

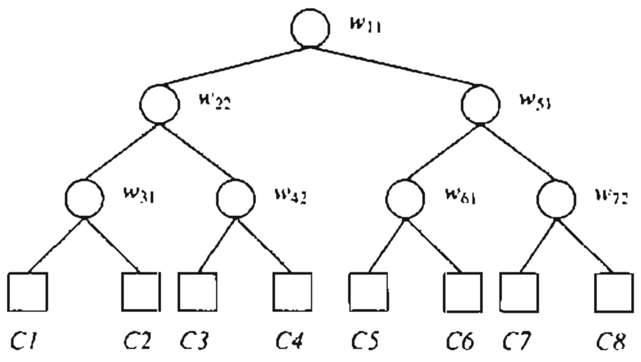

Figure 25 An 8-Cell Tree

After the tree structure was created, the training data were used to train the tree. Training took about 0.55 seconds on a 300 MHz PC, with learning rate $\eta(t) = \frac{0.3}{t}$, where t is the number of data points that have been applied to the tree so far. The learning rate is decreased during training to insure convergence. Now, after training the final weights are shown in Equation (15).

$$\begin{aligned} w_{11} &= -0.3379_{11} \\ w_{22} &= 0.1010_{22} \\ w_{31} &= -0.7153_{31} \\ w_{42} &= 0.5492_{42} \\ w_{51} &= 0.1398_{51} \\ w_{61} &= -0.1000_{61} \\ w_{72} &= -0.0182_{72} \end{aligned} \tag{15}$$

Figure 26 shows the final partitioned cells.

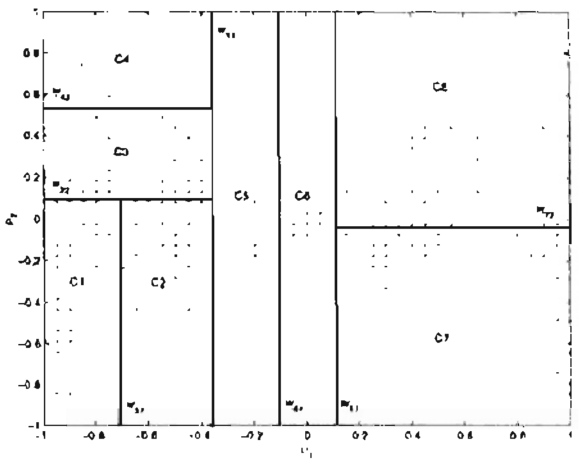

Figure 26 Partitioned Cells After Training

Every cell in the trained tree covers an infinite area. For example, cell $C1$ covers the $C1 \in [(-\infty, -0.7153), (-\infty, 0.1010)]$. Therefore, we need to limit the cell size. In this example, the maximum or minimum values of the training-data points falling in a cell will be used to limit the cell sizes. This approach gives cell $C1$ a finite area, which is denoted by $C1 \in [(-1, -0.7153), (-1, 0.1010)]$; the minimum value of data within cell $C1$ in the first dimension is -1. This is also the smallest value for the second coordinate of training data in cell $C1$.

Novelty detection can be implemented after every cell has bounded area. The algorithm defines abnormal data as data outside of any cell. Figure 27 illustrates novelty detection by plotting training data, testing data, and identified abnormalities within the testing data.

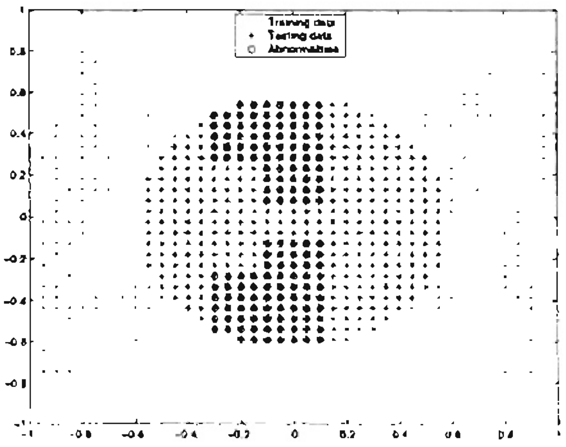

Figure 27 Abnormalities outside the cells

The next step is to test whether data flagged as novel is correlated with large errors in the multilayer network output. (Recall that the purpose of novelty detection in this thesis is to identify inputs for which the trained multilayer network is unreliable.) Before doing so, we will introduce our indicators to measure the performance of novelty detectors.

When performing novelty detection to reject or to accept data, we can make two types of *errors*. In the first case, we reject data, even though they create small errors. In the second case, we accept data even though they generate large errors. The following table illustrates these ideas.

Table 1 Novelty detection decision vs. approximation error

Decision\Error	Small Error	Large Error
Accept	Correctly-classified data	Misclassified data
Reject	Misclassified data	Correctly-classified data

From the above table, one might ask how we decide what is a small error or what is a large error. Throughout this thesis, we consider the error as unacceptable when its value is greater than 0.15 in the normalized hyperspace, in which targets are bounded between $[-1,1]$. The threshold used to accept or reject data will vary from algorithm to algorithm. For each algorithm we will indicate the threshold we use.

An indicator we will use to measure the performance of the algorithm is the percentage of misclassified data points. Clearly, the larger the percentage of misclassified points, the worse the algorithm. However, keep in mind that the percentage of misclassifications depends on the definition of large error. For some applications, the error for abnormalities is required to be very small to guarantee the reliability of the network output. For example, if the large error (abnormality) is defined as the error greater than 0.01, the percentages of misclassifications we will show throughout this thesis will be changed as well (since we defined the large error as greater than 0.15). Note that from now on we will use the term *type I error* to represent small-error data that is rejected by novelty detection. On the other hand, *type II error* will represent large-error data that is accepted by novelty detection.

From the above example, the percentage of misclassified points for the neural tree algorithm was 25.17%. Figure 28 illustrates the error of the testing data and the data marked as abnormal. (The abnormal data points are flagged with an x at the bottom of the figure.)

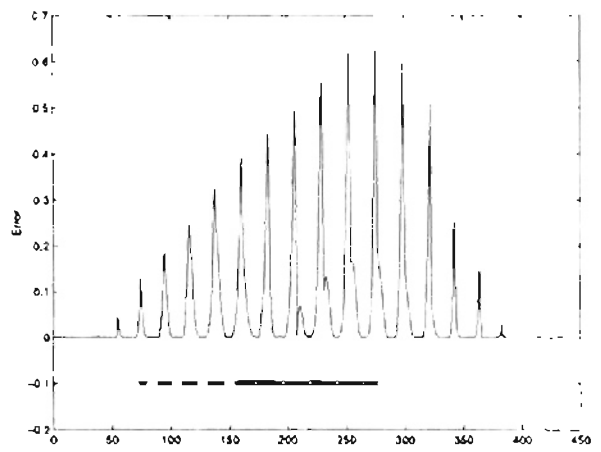

Figure 28 Error and Abnormalities of the 8-cell tree

We can see in Figure 28 that some large-error points were identified. This is because some testing data outside the training data region were undetected (shown in Figure 27). This is due to the fact that we do not have enough cells (we had only 8). Therefore, increasing the number of cells will be a way to detect such data. By increasing the number of cells to 200, the time used to initialize and train the 200-cell tree was 4.51 seconds. The time utilized to identify abnormalities of the 437 testing points was 1.36 seconds. Figure 29 demonstrates data that the 200-cell tree decided to mark as abnormal.

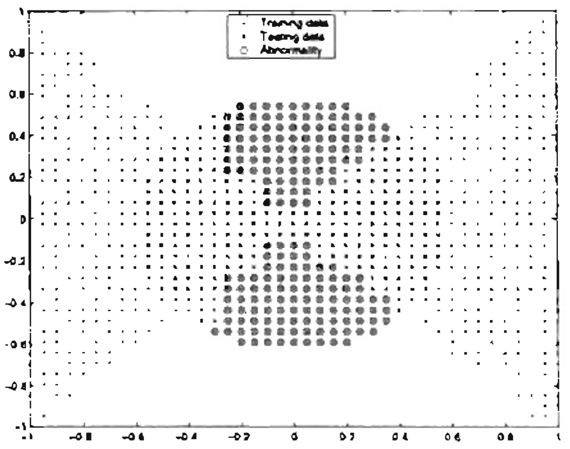

Figure 29 Abnormalities from the 200-cell tree

We can see that almost all of testing data outside training data region were identified as abnormalities.

Figure 30 shows the error from the function approximator on the testing data and indicates data marked as abnormal.

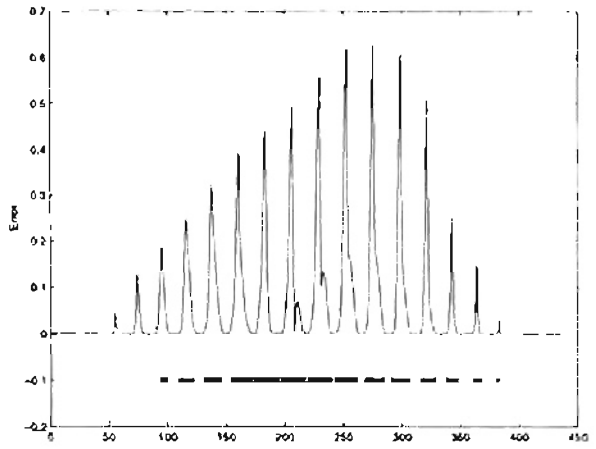

Figure 30 Error and abnormalities of the 200-cell tree

As seen in Figure 30, most of testing data were identified as abnormal. Although most of the testing data that fell outside our training data were detected in Figure 29, the number of misclassification was 46.91%. All of these misclassified points were classified as novel, although the errors were small (type I error). Although we can see that the percentage of misclassifications for the 200-cell tree was larger than for the 8-cell tree, we should realize that this percentage is from only one data set. The percentage we show here will not apply to every data set.

From this chapter through Chapter 7, we will use only one simulated data set to demonstrate how the novelty detection algorithms work. However, in Chapter 8, we will apply real world data to various novelty detection algorithms. That chapter will provide more thorough tests of the algorithms.

From the results we have so far, as we increase the number of cells in the neural tree algorithm, the more abnormalities will be identified. However, any data point that is close to the training data but is outside the cell boundaries will be discarded as abnormal. This will increase the percentage of type I misclassification error.

Thus far, we flag novel data when they break cell boundaries. We will now use the neural tree algorithm to estimate the density for novelty detection.

After training the 200-cell tree, infinite cells will be limited by using the maximum and minimum value of input data, which is 1 and -1 in this case. The area of each cell will be calculated, and the probability density over cell Ci will be computed as $\left(\frac{1}{200}\right)/A_i$, where A_i is the area of cell Ci. Figure 31 illustrates data having low and high density. The red points represent data having density less than 0.2, while the blue points represent data having density greater than or equal to 0.2

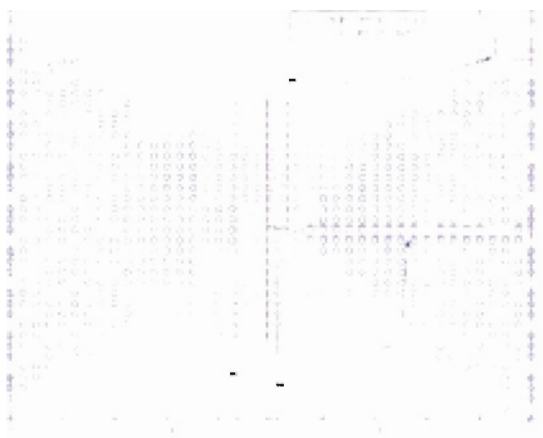

Figure 31. Estimated density using neural tree algorithm

We can see that some training data on the middle of the figure have low density values. This is because the cells (such as cell $C1$, $C3$, $C4$, $C106$ and $C189$ in the figure) occupy very large areas compared with other cells. Therefore, if we use this approach for novelty detection, some training data in large cells may be misclassified as novel. This phenomenon will increase the percentage of type I error.

Although it seems that the neural tree algorithm tends to have very large type I error, we found that it can also have a significant percentage of type II error in some circumstances. Consider the data set in Figure 32 (assume these are training data inputs for a function approximator).

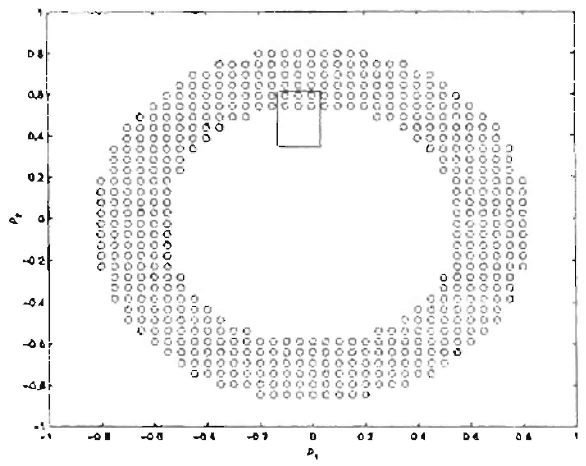
Figure 32 Donut Shape

If we flag novel data when they fall outside cell boundaries, we can see that there is no way to detect data inside the inner circle. This is because the data in this region are always in the cell boundaries (a cell is shown in Figure 32), therefore type II errors may be increased. If we use the density estimation approach, some data inside the inner circle may be detected and some may be not. The final results will depend on how the cells are arranged.

The neural tree algorithm has a high percentage of misclassifications, but its major advantage is its speed.

Summary

In this chapter, we introduced the fundamental ideas of the neural tree algorithm. We defined the notation and terminology commonly used with this algorithm (e.g. tree, node, cell, etc.) We explained how a tree divides the hyperspace, and how a tree is trained to learn a data distribution. We then explained that, for novelty detection, we had to first limit the size of cells. Abnormalities were indicated as data falling into low-density cells (density estimation approach) or as data located outside of the cells (cell boundary approach). The ability of this algorithm was then demonstrated using a simple example. We showed through the simulation results that increasing the number of cells increases the ability of the neural tree to identify data outside the region of training data. We showed why the density estimation approach tends to increase type I misclassification.

After we demonstrated the novelty-detection process, we introduced the performance measure – the percentage of misclassified data points. We suggested that a further study on how to appropriately limit cells for novelty detection should be necessary.

CHAPTER 4

THE GAUSSIAN KERNEL ESTIMATOR

Introduction

The Gaussian kernel estimator for novelty detection was first adopted by [Bish94], and was thoroughly described in [Bish95]. This method is based upon the estimation of the probability density function, as described in many statistics books. Most real-world applications and research involving novelty detection employ this method, as in [Bish94], [NaCoRiToTa97], [TaNaToCo99], or [HiAu00].

Within this chapter, we will first give the reason why we are interested in using the density function estimate for novelty detection. Then we will explain three well-known methods for density estimation, which include histogram, naive estimator, and kernel estimator. A specific kernel function, the univariate Gaussian estimator, will be introduced, followed by the generalized model for the multivariate case. Next, the procedure for adopting this algorithm to novelty detection will be described. The algorithm will be demonstrated through simulation example. After that, we will propose an idea to improve the performance of the algorithm by incorporating the network output with the network input. The improved algorithm will be illustrated with computer simulations. Finally, we will analyze a problem with the proposed algorithm.

Estimated density for novelty detection

[Bish94] developed the novelty-detection method employing the Gaussian kernel estimator by the error equation for training a function approximator:

$$sse = \sum_{i=1}^{N} \iint \{a(\mathbf{p}_i, \mathbf{W}) - t_i\}^2 f(\mathbf{p}_i, t_i) d\mathbf{p} dt \tag{16}$$

where $a(\mathbf{p}_i, \mathbf{W})$ is the output of a function approximator corresponding to the i^{th} training-data input, $\mathbf{p}_i$, through the network having weights $\mathbf{W}$, t_i is the target and $f(\mathbf{p}_i, t_i)$ is the joint probability density function of the input and target. By applying Bayes' rule, we will replace the joint density with a product of $f(t_i|\mathbf{p}_i)$ and $f(\mathbf{p}_i)$. When rearranging Equation (16), we obtain

$$sse = \sum_{i=1}^{N} \int \{a(\mathbf{p}_i, \mathbf{W}) - E[t_i|\mathbf{p}_i]\}^2 f(\mathbf{p}_i) d\mathbf{p} + \sum_{i=1}^{N} \int \{E[t_i^2|\mathbf{p}_i] - E[t_i|\mathbf{p}_i]^2\} f(\mathbf{p}_i) d\mathbf{p} \tag{17}$$

where E is the expectation operation.

From Equation (17), we can see that the error equation is weighted by $f(\mathbf{p}_i)$, which represents the density function of the input data $\mathbf{p}$. After training the function approximator (minimizing the sse over a finite data set), we expect that the approximation is accurate in regions that the density $f(\mathbf{p}_i)$ is high. On the other hand, there is a small contribution to the error minimization (Equation (17)) from the regions where $f(\mathbf{p}_i)$ is low. Consequently, the approximation should not be precise in these regions, thus resulting in large error from the function approximator. Therefore, the density function $f(\mathbf{p}_i)$ can be an indicator to predict when the approximation is not accurate. In other words, we may use the density function as novelty detector for function approximation. Unfortunately, the density function $f(\mathbf{p}_i)$ is unknown, and therefore we have to estimate it. The next section is dedicated to reviewing the density estimation, from the histogram to the Gaussian kernel estimator.

Background

The probability density function gives a description of the distribution of a random variable p. Probabilities associated with p can be found by

$$P(c_1 < p < c_2) = \int_{c_1}^{c_2} f(\lambda)d\lambda \quad ;\forall c_1 < c_2 \tag{18}$$

Now, suppose that there is a set of observed data sampled from an unknown probability density function. The method of predicting the unknown density function from the observed data points is called "density estimation". Three density estimation methods will be discussed here, starting with the well-known histogram, followed by the naive estimator, and finally leading to the kernel estimator.

Though most applications in the real world deal with multiple variables, we will begin with the univariate case because of its simplicity. Before starting the discussion, let's define some notation that will be used throughout this chapter. Assume that a random variable p consists of N real observations $p_1, p_2, \ldots, p_N$ sampled from a data set whose underlying density is unknown and to be estimated. Then, the estimated density function of these observations will be denoted as $\hat{f}$.

Histogram

The oldest and most extensively used method for estimating an unknown density function is the histogram. It is mainly comprised of a series of boxes with heights indicating how many data are contained in a certain region. To create a histogram, we begin by defining a set of bins starting from the point λ_0. Each bin is defined as the interval $[\lambda_0 + mb, \lambda_0 + (m+1)b)$, where m is an integer representing the bin number. Every bin has width b. Then the histogram is defined as

$$\hat{f}(\lambda) = \frac{1}{Nb}(\text{numbers of } p_i \text{ in same bin as } \lambda) \tag{19}$$

The estimated density is constant over each bin. The bin width b is sometimes called the smoothing parameter. Note that in a more general form, the bin width can be adjustable from bin to bin as well.

The accuracy of the histogram depends upon both the starting value and the bin width. These values have to be chosen by experience and the choice may cause undesirable effects, such as misinterpretation of the density estimate. Furthermore, though it is an excellent tool to represent the approximate density for a single random variable, it does not work well in high dimensional spaces. Even in two or three dimensions, it is extremely difficult to create understandable figures. Moreover, the discontinuities between adjacent boxes are not desirable. The next method we will discuss known as the naive estimator is designed to overcome some of the problems of the histogram.

Naive estimator

The naive estimator is an alternative to the histogram. It eliminates some undesirable effects of histogram, especially the choice of origin value, as will be seen. From Equation (18), the density function can be rewritten as

$$f(\lambda) = \lim_{b \to 0} \frac{1}{2b} P(\lambda - b < p < \lambda + b) \tag{20}$$

The right hand side can be estimated by computing the fraction of observed data falling within the interval $(\lambda - b, \lambda + b)$, and can be written as

$$P(\lambda - b < p < \lambda + b) \approx \frac{\text{numbers of } p_i \text{ falling in interval}(\lambda - b, \lambda + b)}{N} \tag{21}$$

Therefore, for a small enough value of b, Equation (20) can be reformulated and used as an estimate

$$\hat{f}(\lambda) = \frac{\text{numbers of } p_i \text{ falling in interval}(\lambda - b, \lambda + b)}{2Nb} \tag{22}$$

The above equation is a simple form of the naive estimator. The general form is

$$\hat{f}(\lambda) = \frac{1}{Nb} \sum_{i=1}^{N} w\left(\frac{\lambda - p_i}{b}\right) \tag{23}$$

where $w(\lambda)$ is called the weight function and can be defined as

$$w(\lambda) = \begin{cases} \frac{1}{2} & ; |\lambda| < 1 \\ 0 & ; \text{otherwise} \end{cases} \tag{24}$$

In Equation (23), we are constructing a box of width $2b$ over the range $(\lambda - b, \lambda + b)$ and height $\frac{1}{2Nb}$ on each observed data point. After adding them together, the density estimate is eventually obtained. Therefore, the naive estimator uses observations to be the reference for each box, meaning that the center of each box is an individual observation. In the case of the histogram, the origin value has to be initialized before making a box. That means that the naive estimator is a modified version of histogram in which each sampled data point creates its own histogram centered on itself.

The naive estimator eliminates the problem of setting up the origin value. Nevertheless, the trouble of selecting the smoothing parameter and the discontinuity of the curve still remain. The discontinuity of the curve is the most undesirable feature, because the derivative of the estimated density is infinite at every point $p_i \pm b$, and is zero elsewhere.

Before going on to another section, let's consider how the estimate is affected by changes in the smoothing parameter. When the parameter b is reduced, Equation (22) implies that the interval $2b$ is decreased whereas the ordinate of each box $\frac{1}{2Nb}$ is increased, making the estimated density more jagged. The graphical presentation will look more noisy when the parameter is smaller. On the other hand, when the parameter is increased, the discontinuous characteristics will be reduced but some underlying information will be lost because of too much overlap of the blocks. Figure 33 demonstrates the effect of varying the smoothing parameter on the naive estimator.

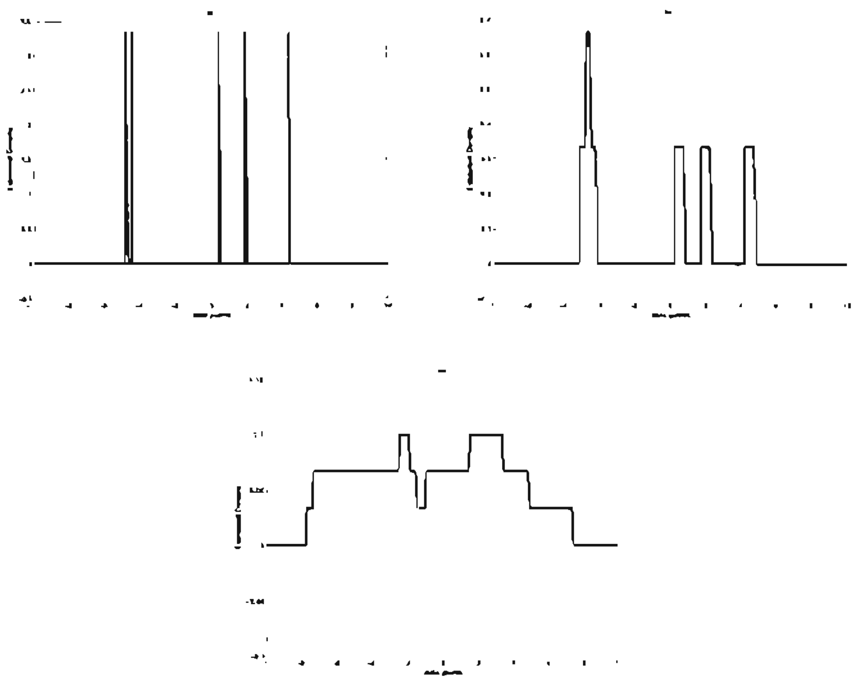

Figure 33 Effect of the smoothing parameter on the naive estimator (a) $b = 0.03$ (b) $b = 0.3$ (c) $b = 3$

As with the above explanation, the estimated density will be smeared if the smoothing parameter is too small, and it will be obscured if the smoothing parameter is too large. Therefore, the choice of the smoothing parameter is one of the most critical steps in using the naive estimator.

As noted above, the undesirable characteristic of the naive estimator is its discontinuity. To overcome this problem, we will modify the weight function using a kernel function, as explained in the next section.

Kernel estimator

In order to obtain a continuous curve, we can modify the weight function. We will replace the weight function $w(\lambda)$ with the kernel function $K(\lambda)$, which satisfies the condition

$$\int_{-\infty}^{\infty} K(\lambda)d\lambda = 1 \tag{25}$$

By substituting this kernel function into Equation (23), it becomes

$$\hat{f}(\lambda) = \frac{1}{Nb}\sum_{i=1}^{N} K\left(\frac{\lambda - p_i}{b}\right) \tag{26}$$

The kernel function is normally chosen to be symmetric and non-negative everywhere. Some of examples of kernel functions are the biweight, triangular, and normal density function (the Gaussian function). By selecting a continuous kernel function, the density function estimate will also be continuous.

As with the naive estimator, the kernel estimator can be thought of as adding together all of the kernel curves centered at the observations. In other words, rather than placing a box of width $2b$ having a midpoint at the observations with height $\frac{1}{2Nb}$, a kernel curve with a specific interval and height constrained by its own properties is located on each observation. The density estimate at a point λ is obtained by summing all of the individual kernel functions. Like the naive estimator, the smoothing parameter controls how smooth the curve is. If the parameter is too large, the kernel functions overlap too much and are smeared together. If the parameter is too small, the approximate density is just as misleading, and consists of a spike at each observation.

Therefore, though the estimate is continuous, the problem of how to select the value of the smoothing parameter still remains. However, the kernel estimator has been one of the most popular methods employed for density estimation thus far. In the next section, a certain kernel function called the Gaussian function will be deployed in order to perform to novelty detection.

The Gaussian kernel estimator

In this section, one of the most well-known functions in the statistics, mathematics and engineering fields – the Gaussian function – will be employed as the kernel function. We will begin with the univariate case, followed by the generalized model in which any number of dimensions can be used.

One-dimensional data

We generally choose a kernel function that is differentiable, symmetric and non-negative. The Gaussian function with zero mean and variance of one

$$g(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}x^2\right) \tag{27}$$

satisfies all of the above properties. Now, by substituting Equation (27) into the kernel function in Equation (26), it becomes

$$\begin{aligned}\hat{f}(\lambda) &= \frac{1}{Nb}\sum_{i=1}^{N}\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{\lambda-p_i}{b}\right)^2\right)\\ &= \frac{1}{\sqrt{2\pi}Nb}\sum_{i=1}^{N}\exp\left(-\frac{1}{2}\frac{(\lambda-p_i)^2}{b^2}\right)\end{aligned} \tag{28}$$

The above equation is the Gaussian kernel estimator for univariate density functions.

Notice that the smoothing parameter controlling the width of the Gaussian curve is the standard deviation in the normal density function, while an observation p_i can be thought of as the mean of the kernel curve. When the standard deviation of a normal density function increases, the curve expands and the points far away from its mean have comparable values to points close to the mean. In contrast, when the standard deviation decreases, the shape becomes more like a spike, and the density at a small distance remote from the mean is close to zero. Figure 34 shows what the estimated densities look like when the smoothing parameter is varied. The estimated density computed from Equation (28) is exhibited by the bold solid line, while individual kernels for the five observations are indicated by the thin lines. When the smoothing parameter is too small, several peaks pop up and could be misleading. When the value is too large, it can smooth over much of the detail in the true density.

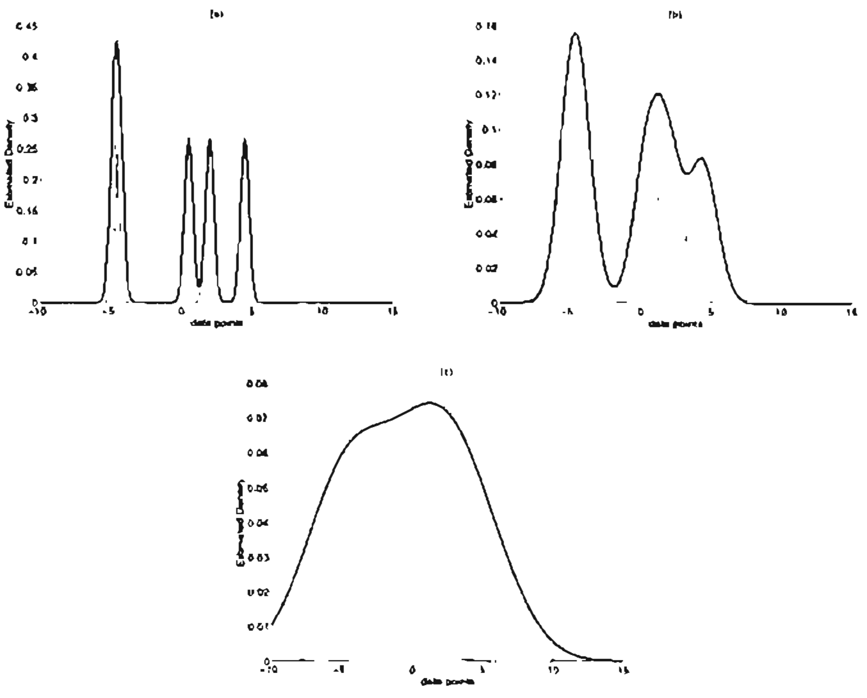

Figure 34 The Gaussian kernel estimator with (a) $b = 0.3$ (b) $b = 1$ (c) $b = 3$

In the next section, the multivariate case will be discussed.

The generalized model

Most applications involve high dimensional spaces. The higher the dimension, the more difficult it will be to make accurate estimates of the density function and to represent it graphically. However, let's modify our existing notation in order to represent high dimensional data.

Suppose the random vectors $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N$ are sampled from a population with unknown distribution. Also, assume that the size of each vector is $q \times 1$. Then, using the same concept as the univariate case, the kernel estimator in Equation (26) can be rewritten as

$$\hat{f}(\lambda) = \frac{1}{Nb^q} \sum_{i=1}^{N} K\left(\frac{\lambda - \mathbf{p}_i}{b}\right) \tag{29}$$

where the kernel function has to abide by the condition

$$\int_{R^q} K(\lambda) d\lambda = 1 \tag{30}$$

Similarly, the Gaussian kernel function will be modified to fit into high dimensional spaces using the following formula.

$$g(\mathbf{x}) = \frac{1}{(2\pi)^{q/2}} \exp\left(-\frac{1}{2}\mathbf{x}^T\mathbf{x}\right) \quad ; \forall \mathbf{x} \tag{31}$$

By substituting Equation (31) into the kernel function in Equation (29), it becomes

$$\begin{aligned}\hat{f}(\lambda) &= \frac{1}{Nb^q}\sum_{i=1}^{N}\frac{1}{(2\pi)^{q/2}}\exp\left(-\frac{1}{2}\left(\frac{\lambda-\mathbf{p}_i}{b}\right)^T\left(\frac{\lambda-\mathbf{p}_i}{b}\right)\right)\\ &= \frac{1}{Nb^q(2\pi)^{q/2}}\sum_{i=1}^{N}\exp\left(-\frac{1}{2b^2}(\lambda-\mathbf{p}_i)^T(\lambda-\mathbf{p}_i)\right) \qquad (32)\\ &= \frac{(2\pi)^{-q/2}}{Nb^q}\sum_{i=1}^{N}\exp\left(-\frac{(\lambda-\mathbf{p}_i)^T(\lambda-\mathbf{p}_i)}{2b^2}\right)\end{aligned}$$

Now, the above equation is a generalization of the Gaussian kernel estimator. The smoothing parameter b is heuristically chosen, depending on the problem that is going to be solved. It has to be neither too large nor too small to obtain a desirable result. In the more general form, the smoothing parameter could be a matrix, like the covariance matrix, and the density estimate can be written as

$$\hat{f}(\lambda) = \frac{(2\pi)^{-q/2}}{N|\Sigma|^{1/2}}\sum_{i=1}^{N}\exp\left(-\frac{(\lambda-\mathbf{p}_i)^T\Sigma^{-1}(\lambda-\mathbf{p}_i)}{2}\right) \qquad (33)$$

where Σ is the smoothing-parameter matrix, which can be thought of as the covariance matrix, and has the size $q \times q$. The notation $|\Sigma|$ represents the determinant of the matrix. Notice that (33) is the general form of (32) such that $\Sigma = b^2\mathbf{I}_q$, and $\mathbf{I}_q$ is the identity matrix with size $q \times q$.

Note that the use of $\Sigma = b^2\mathbf{I}_q$ implies that the width of the Gaussian kernel placed on each observation is equal in all directions, and that the data in each dimension is uncorrelated with each other. Although a smoothing parameter matrix should be used to efficiently estimate density functions for data that is not evenly distributed, a constant value will be used in this thesis for novelty detection.

In the next section, we will explain how this density function estimator can be used for novelty detection.

Application to novelty detection

We will use the estimated density function for novelty detection. The main idea is the following. The estimated density function describes the distribution of the training data set. If a new input vector is similar to vectors in the training set, we would expect that the estimated density function will be relatively large at that point. If a new input vector is unlike any vector in the training set, then the estimated density function should be small at that point. Therefore, those inputs that have a small value for the estimated density will be considered novel inputs.

In the next section we will revisit the example problem described in Chapter 3. We will use it to test the Gaussian kernel estimate of the density function, Equation (33), with covariance matrix $\boldsymbol{\Sigma}_p = b^2\mathbf{I}_q$:

$$\hat{f}(\boldsymbol{\lambda}) = \frac{(2\pi)^{-q/2}}{Nb^q}\sum_{i=1}^{N}\exp\left(-\frac{(\boldsymbol{\lambda}-\mathbf{p}_i)^T(\boldsymbol{\lambda}-\mathbf{p}_i)}{2h^2}\right) \tag{34}$$

Note that $\mathbf{p}_i$ is a training vector, corresponding to an observation $\mathbf{p}_i$ in Equation (33).

Simulation of the simple example #1

The following is the example illustrating the capability of the Gaussian kernel estimation novelty detector. The regions for the testing and training data were shown in il. The input vectors are two dimensional, and the number of training data is 638, thus giving $q = 2$ and $N = 638$. Figure 35 shows the estimated density for various values of the smoothing parameter.

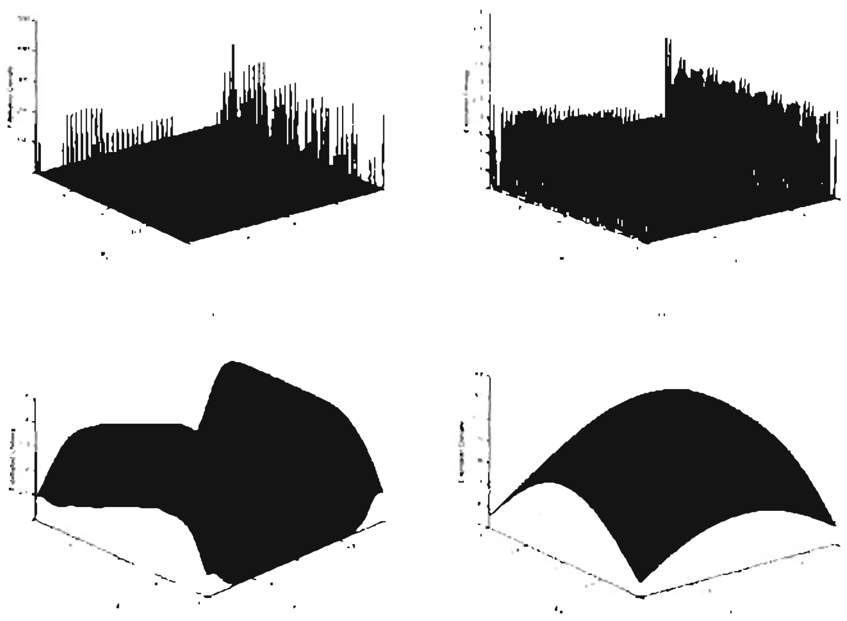

Figure 35 Estimated Density with (a) $b = 0.001$ (b) $b = 0.01$ (c) $b = 0.1$ (d) $b = 1$

Recall from the previous chapter that we want to use a novelty detector in combination with a multilayer network that has been trained for function approximation. If the novelty detector flags data as being different than data in the training set, then we expect that the multilayer network may perform poorly on that data. The novelty detector is a warning system for the multilayer network. In this context, we will test the Gaussian kernel estimator on the function approximation problem described in the previous chapter.

Figure 36 plots the estimated density values versus the error of the multilayer network on the 437-test points of the function approximation problem. (See Figure 22 for the location of the test points.) Figure 36 does indicate that the lower the estimated density is, the more likely the error of the multilayer network will be large.

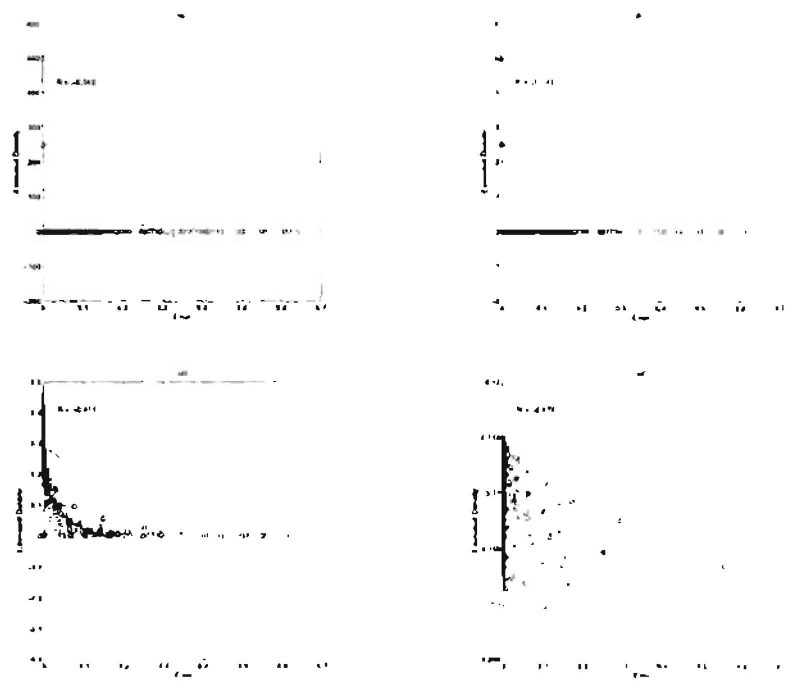
Figure 36 Estimated Density and The Errors (a) $b = 0.001$ (b) $b = 0.01$ (c) $b = 0.1$ and (d) $b = 1$

From Figure 36, we can see that there is a straight line and a scalar value R shown on each graph. The straight line represents a linear regression between the two variables – the network error and the estimated density in this case. The regression line will be used to determine the density function value used to flag abnormal data. We will describe how to determine this threshold value from the regression line later.

The R value represents the correlation coefficient between the network error and the estimated density $(-1 \leq R \leq 1)$. If R value is positive $(R > 0)$, it means that the error tends to be high when the estimated density is high. On the other hand, if $R < 0$, it indicates that the error tends to be high when the estimated density is low. In addition, the greater the magnitude of the correlation coefficient ($|R| \rightarrow 1$), the more correlation between the variables. Therefore, when the R value is close to zero, it means that there is almost no correlation between the two variables.

We can see in Figure 36 that the R values were negative, thus implying that when the estimated density was low, the error tended to be high.

In order to decide which smoothing parameter values should be used for novelty detection, Bishop [Bish94] suggested the method of choosing the b value by averaging the distance to the ten nearest neighbors over the entire training data. By using this method, we found that the b value for our example is 0.0836.

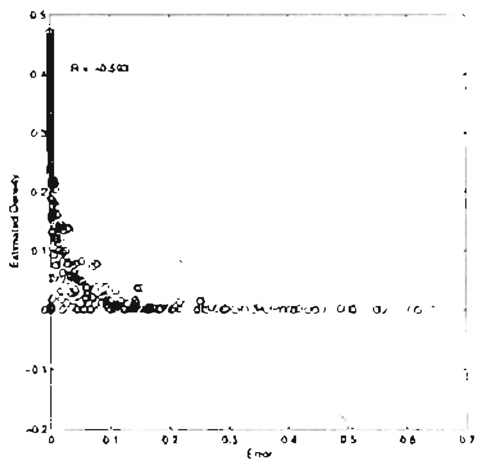

Figure 37 Estimated density and approximation error: $b = 0.0836$

To use the density estimate for novelty detection, we must choose a threshold below which the data will be flagged as novel. Based on Figure 36 (c), we found that the regression line that represents the network error and the estimated density with $b = 0.0836$ is

$$density = -0.967 \times error + 0.294 \tag{35}$$

We will use Equation (35) to find the estimated density value (based on this data set) that produces an approximation error of 0.15. By substituting 0.15 for the error term in Equation (35), the corresponding estimated density is 0.149. That indicates that, based on this testing data set, the estimated density for data points generating an error of 0.15 for the function approximator is on average equal to 0.149. We will use this value of the estimated density to be the threshold to reject novel data. In other words, any data generating an estimated density less than 0.149 will be discarded as novel.

Figure 38 shows the approximation error of the neural network. The points that are flagged with an **x** have estimated density values less than 0.149.

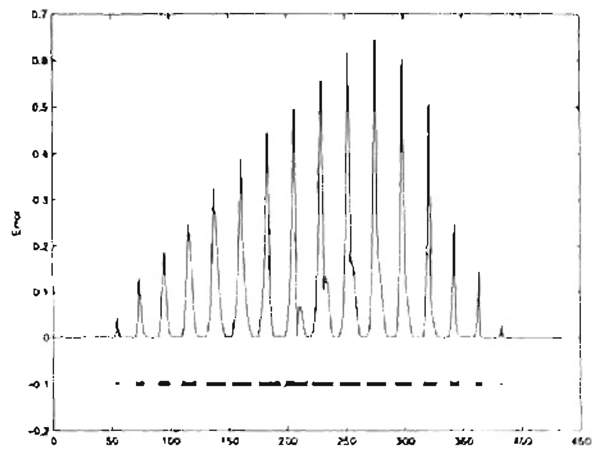

Figure 38 Novelty Detection: Density of input

As seen in Figure 36 (c) and Figure 38, the algorithm has a certain capability to pin-point which data should be discarded. The percentage of misclassified points when employing this algorithm was 28.38%. All of the misclassifications were from small-error points that were flagged as novel (type I error).

Note from Figure 37 that if we use density lower than 0.149 to reject novel data, the percentage of misclassifications will be reduced. This is because the threshold we chose (based on the regression line) discarded many data points with small errors. We found that in this example the threshold that minimizes the percentage of misclassifications is around 0.005. For this threshold, the percentage of misclassifications is 7.32%. Around 5.49% out of the 7.32% are type I misclassifications. Though there is a large difference between the percentages of misclassifications for the two different thresholds in this example, when we apply the algorithm to our real world data in Chapter 8, the difference is no larger than 2%.

Although the purpose of using the testing data is to set the threshold, it should be noted that it is very difficult to have a general value for the threshold that will minimize the percentage of misclassifications for all data sets. What we can generally say about setting the threshold is that the *higher* the threshold, the more likely we will experience type II error. On the other hand, the *lower* the threshold, the more likely we will face type I misclassification. Therefore, the appropriate threshold value will depend on our application.

Even though the percentage of misclassifications in this case is much less than that for the neural tree, we found that this algorithm sometimes can reject more training data than any other method. This is due to the fact that the estimated density of some training data can be lower than that of some testing data. Such training data are located in regions far away from the majority, thereby reducing the effect of the kernel curves from adjacent training points. Figure 39 shows the estimated density of a one-dimensional data set with the training data marked as **x**.

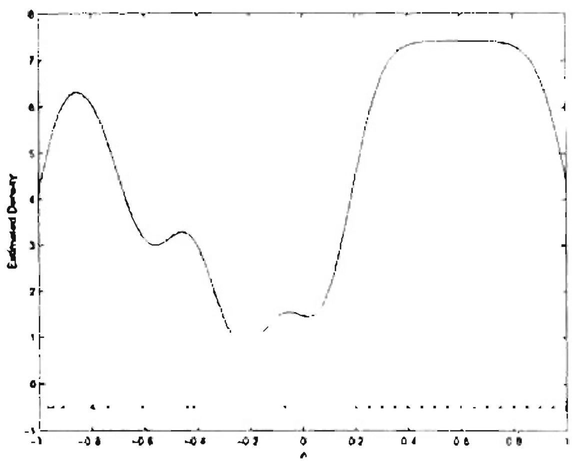

Figure 39 Estimated density of a data set

From the figure, we can see that some training data points, for example at $p = -0.06$, have very low density. Its value is even lower than some testing data points, such as at $p = -0.9$ or $p = -0.5$. That means that these low-density training points are more prone to being discarded as abnormalities than some testing points, whose errors for the function approximator may be larger. We can see in Chapter 8 that this algorithm rejects more training data as novel than any other novelty detector. A way to reduce this problem is to reduce the value of the smoothing parameter.

As we explained earlier, the smaller the smoothing parameter, the smaller the region the kernel curve will cover. Thus, if we choose to have a small smoothing parameter, the estimated density of the training data will be higher than testing points; however, the density of the interpolations will be very low (see Figure 35 (a)). That indicates that we will discard these small-error points (interpolations) as abnormalities, thus increasing the percentage of misclassifications. Therefore, no matter how large the smoothing parameter, this method tends to reject small-error points (either training data or interpolation points) as novel data.

A way to improve the performance: Joint Density

Thus far, we have computed the estimated density of the input to the function approximator. Recall from Equation (16) that the probability density function that we actually use to minimize the sum-square error is the joint density between the input and the target of the network. Because the target is assumed to be unknown, it may be difficult to find the joint density. However, in this section will propose a procedure for computing the joint density between network input and output. Then we will use the estimated joint density to develop an improved novelty detection procedure.

One advantage of using the joint density is that if a network could not minimize the errors very well on some of the training data, the joint density between the network inputs and outputs for those training data could be rejected based on comparing with the joint density between the network inputs and targets. Also, there might be a regions close to training-data inputs where the network did not minimize the error. This would cause the network outputs from that region to be unreliable (computing the density of the network output would be a good way to indicate such phenomena).

We can represent the joint density of two jointly Gaussian random vectors $\mathbf{x}$ and $\mathbf{y}$ by creating an augmented vector $\mathbf{z} = [\mathbf{x}\ \mathbf{y}]^T$. The new random vector $\mathbf{z}$ will also follow the Gaussian distribution. Thus, for any training data, we will create the new composite vector $\Gamma_i = \begin{bmatrix} \mathbf{p}_i & t_i \end{bmatrix}^T$. For testing data, we need to propagate the input $\mathbf{p}_j^t$ through the function approximator to get the network output a_j^t. Then we will augment the network input and output to create the composite testing data $\beta_j = \begin{bmatrix} \mathbf{p}_j^t & a_j^t \end{bmatrix}^T$. Then Equation (33) can be rewritten as

$$\hat{f}(\beta) = \frac{(2\pi)^{-q/2}}{N|\Sigma_\Gamma|^{1/2}} \sum_{i=1}^{N} \exp\left(-\frac{(\beta - \Gamma_i)^T \Sigma_\Gamma^{-1} (\beta - \Gamma_i)}{2}\right) \tag{36}$$

where Σ_Γ is the smoothing parameter matrix. Note that the structure of Σ_Γ is

$$\Sigma_\Gamma = \begin{bmatrix} \Sigma_p & \Sigma_{pt} \\ \Sigma_{pt}^T & \Sigma_t \end{bmatrix} \tag{37}$$

where Σ_t is the smoothing parameter for the target, and Σ_{pt} is the co-smoothing parameter between the network input and target.

In the next section, the estimation of the joint density will be illustrated by using the simple example we have used in previous sections.

Simulation of the simple example #2

In this example we will compute the joint density of training data, and compare with that of testing data. We will use Equation (36) for estimating the joint density.

We assume that all elements of the augmented training vectors Γ are uncorrelated. Therefore, Σ_Γ will be a diagonal matrix. By using the same criterion as we did in example

#1 (set the b value equal to the average distance to the ten-nearest neighbors in the training data), the b value in this case then is 0.1164. In other words, Σ_Γ in this example is equal to

$$\Sigma_\Gamma = \begin{bmatrix} 0.1164^2 & 0 & 0 \\ 0 & 0.1164^2 & 0 \\ 0 & 0 & 0.1164^2 \end{bmatrix} \tag{38}$$

After we used Equation (36) for computing the estimated joint density, we obtained the estimate shown in Figure 40.

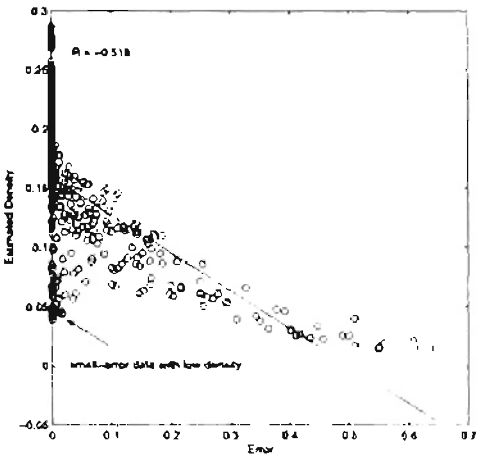

Figure 40 Estimated density and approximation error: $b = 0.1164$

We can see from Figure 40 that large-error data clearly have low density. This relationship is clearer here than in example #1. The regression line shown in the figure is

$$density = -0.318 \times error + 0.159 \tag{39}$$

By substituting 0.15 for the error term, the corresponding density is equal to 0.1113. We will use this value as the threshold. Figure 41 demonstrates the network error, and abnormal data are flagged with an $\mathbf{x}$.

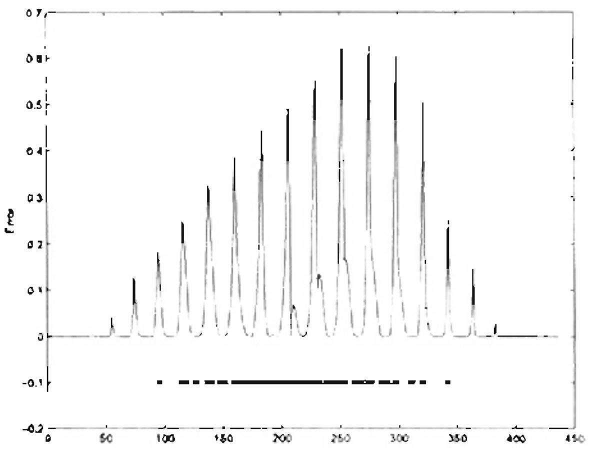

Figure 41 Novelty detection: Density of input and output

The percentage of misclassifications in this case was 23.34%, which is a little bit less than the result in example #1. All of the misclassifications in this case were from type I errors.

We found that the threshold that produces the fewest misclassifications is 0.04. For this threshold, the percentage of misclassifications is 8.46%. Around 8% of the 8.46% are type II errors. This is less than the percentage of misclassifications we obtained for the threshold based on the regression line. However, in chapter 8, where we apply this technique to real world data, we found that the difference will not be this large.

From the results we obtained using the threshold of 0.04, the total misclassification is a little larger in example #1 (when we chose the threshold with fewest misclassifications). This is due to the fact that there are some small-error data with low density marked in Figure 40. Such data points are shown in the two-dimensional plot in Figure 41.

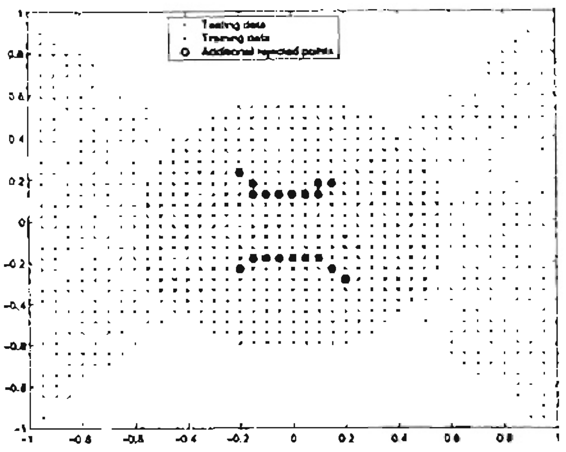

Figure 42 Small-error and low-density points

From Figure 42, we can see that these data occupied a region where the density of training inputs is low, compared with the other regions containing training data. However, the real problem is that the density of the targets for this data is very low, resulting in low joint density around this region. Therefore, any data around this region will naturally have low density, compared to the density in the other regions. This phenomenon makes the data in this region (though their errors are small) more prone to being discarded as novel data than data in other regions. This is the reason why some small-error points can have low density, and this is the main problem with this algorithm.

In this section, we proposed a technique to improve the performance of the Gaussian kernel estimator for novelty detection. We expected that any data generating unreliable output but falling in the regions whose density of input are somewhat high (from the overlapping of kernels) should have low density. Unfortunately, we found that data points occurring in the regions where the density of inputs and targets is low could have low joint density as well. These data points were therefore detected as novel data.

Summary

We began this chapter by deriving the error equation for training a function approximator. We concluded that the error of the function approximator depends on the joint density between target and input data. However, since the target is unknown, we factorized the joint density using Bayes' rule, and used only the density of the input data. Because the density function of the input data is unknown, we have to estimate it.

We introduced three methods for density estimation, and concluded that the Gaussian kernel estimator is the most desirable method. However, there is a smoothing parameter required by the estimator. We demonstrated that if the parameter is not set correctly the algorithm will perform poorly.

When using the Gaussian kernel estimator, we found that the lower the estimated density, the more likely we were to find large errors in the function approximator. We then proposed a way to improve the performance of the novelty detector by estimating the joint density between network input and output for testing data, and comparing it with the density between network input and target for training data. The simulation results showed that the joint density estimate had an improved capability of identifying abnormalities, and reduced the percentage of misclassified points.

CHAPTER 5

AUTOASSOCIATIVE MULTILAYER PERCEPTRON

Introduction

One of the most common uses of neural networks is to solve pattern recognition problems. In Chapter 2, the general structure of multilayer neural networks was introduced. Frosini and Gori proposed a new approach in [FrGo96] for using multilayer networks to recognize their data and to perform novelty detection. Their application was the detection of bogus banknotes.

In this chapter, we will use multilayer networks as novelty detectors, to recognize data unlike training data. We will start this chapter by briefly recapitulating the neural network structure. The definition of a new novelty detection will be given, followed by a description of how we can use such a neural network to identify abnormalities. A simple simulated example will be used for demonstration. After that, we will propose a technique to improve the performance of this method. Finally, the simulation outcomes with this new technique will be shown.

Autoassociative Multilayer Perceptron

Note that multilayer perceptron is another name for multilayer neural network, which was previously described in Chapter 2. The word *autoassociative* is used because the targets of these neural networks are the same as the inputs to the networks.

The major difference between the autoassociative multilayer perceptron and the multilayer network function approximator is the target output. For the function approximator network, the target output is the output of the function we wish to approximate. For the autoassociative network the target output is the same as the network input.

Note that the objective of training a network is to minimize the sum-square errors sse, or mean-square error mse, between network outputs and targets. Since the output of the autoassociative multilayer perceptron is not in scalar, mse can be written in the form of vectors as:

$$\begin{aligned} mse &= \frac{1}{N}\sum_{i=1}^{N} \mathbf{e}_i^T\mathbf{e}_i \\ &= \frac{1}{N}\sum_{i=1}^{N} (\mathbf{t}_i - \mathbf{a}_i)^T(\mathbf{t}_i - \mathbf{a}_i) \\ &= \frac{1}{N}\sum_{i=1}^{N} (\mathbf{p}_i - \mathbf{a}_i)^T(\mathbf{p}_i - \mathbf{a}_i) \end{aligned} \tag{40}$$

where N indicates the number of training data vectors, $\mathbf{p}_i$ is the i^{th} input vector, and $\mathbf{a}_i$ is the corresponding network output.

From Equation (40), we can see that the quantity written as $(\mathbf{p}_i - \mathbf{a}_i)^T(\mathbf{p}_i - \mathbf{a}_i)$ is the squared error of the i^{th} observation. The error itself can be calculated through the 2-norm operation.

$$\begin{aligned} norm(\mathbf{e}_i) &= \|\mathbf{e}_i\| \\ &= ((\mathbf{e}_i)^T\mathbf{e}_i)^{1/2} \\ &= ((\mathbf{p}_i - \mathbf{a}_i)^T(\mathbf{p}_i - \mathbf{a}_i))^{1/2} \end{aligned} \tag{41}$$

We will call the quantity, $norm(\mathbf{e}_i)$, in Equation (41) "autoassociative error".

In this section, we defined the structure of the neural network used for novelty detection. The major difference between the network for novelty detection and that for function approximation is the number of neurons in the output layer. We will describe in the next section how we can bring this new neural network to be our novelty detector for function approximation. The simulation results will be also provided.

Application to novelty detection

Recall that we want to use a novelty detector in combination with a multilayer network that has been trained for function approximation. If the novelty detector flags data as being different from data in the training set, we then expect that the multilayer network may perform poorly on that data. That means that the novelty detector is a warning system for the multilayer network. In this chapter, we want to test the autoassociative multilayer perceptron on the function approximation problem.

We will create a neural network with the structure described in the previous section (autoassociative multilayer perceptron). We will use this network for novelty detection. Since two kinds of multilayer neural networks are in use, we will call the neural network utilized for novelty detection as the novelty detection (ND) network. The main idea for using the ND network to be a novelty detector for the function approximator is provided in the following.

Although the ND network seems to perform a linear function (the output is the same as its input), Hwang and Cho showed in [HwCh99] that the nonlinear transfer function, e.g. the hyperbolic tangent sigmoid, in the hidden layers is necessary for novelty detection. They concluded that the nonlinear transfer functions create output-constrained hyperplanes on which all output vectors $\mathbf{a}_i$ are projected. Therefore, when we minimize the square error (train the ND network), the hyperplanes will be moved toward the vicinity of the training vectors. Consequently, the squared error of the data points within the constrained hyperplanes will be small, while that of data points far away from such hyperplanes will not be minimized and this will result in large squared error (and autoassociative error). This is the reason why the ND network can "recognize" training data.

We would like to train the ND network with the data we used to train the function approximator so that the autoassociative error for these data is small. Therefore, if a new testing vector is similar to a training vector, the autoassociative error of this vector will be small. We then expect that the error of the function approximator would be small as well (since the testing input is similar to a training input). For a new vector much different from training data, the autoassociative error of this point will be large since it is not in the constrained hyperplanes, and the error from the function approximator of this data should also be large. Therefore, those inputs that have a large autoassociative error will be considered novel points. Figure 43 demonstrates the diagram for the ND network for the function approximator.

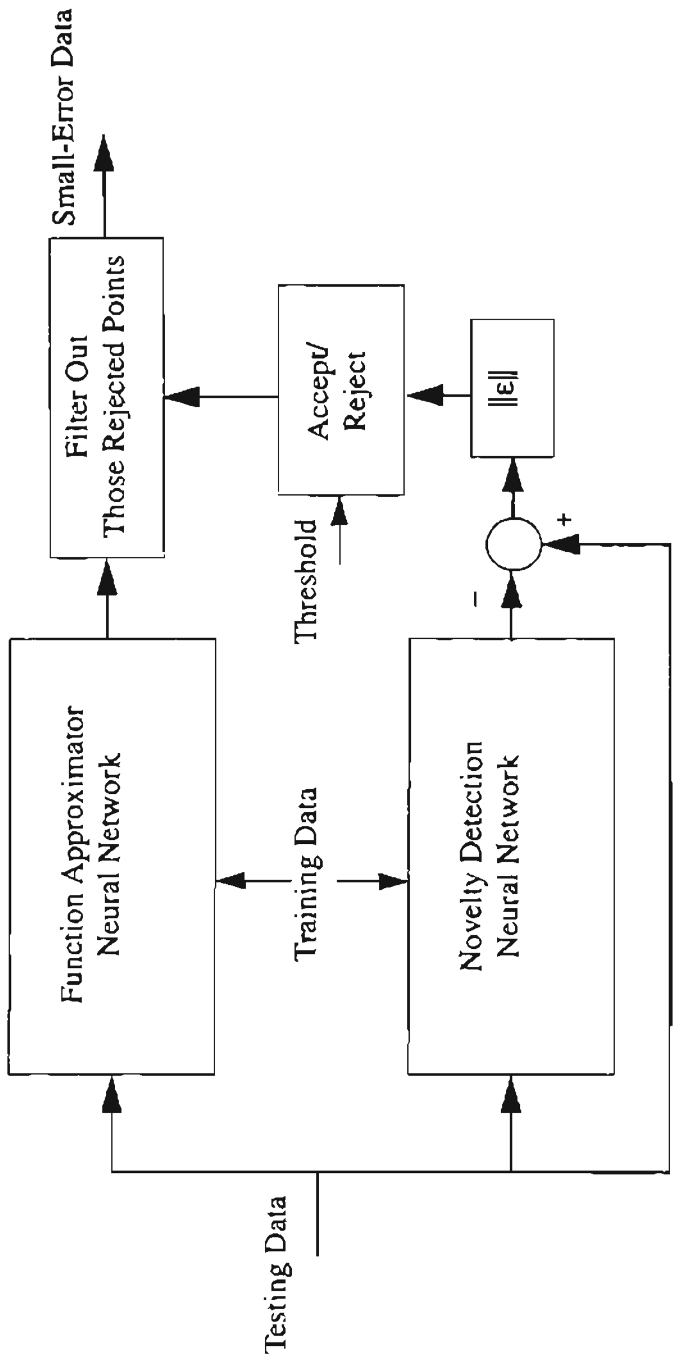
Small-Error Data
Filter Out Those Rejected Points
Accept/ Reject
Threshold
||ε||
+
-
Function Approximator Neural Network
Training Data
Novelty Detection Neural Network
Testing Data

Figure 43 Diagram of an autoassociative novelty detector

In the next section, we will revisit the example problem described in the previous chapters. We will use it to show the ability of this method.

Simulations of the simple example #1

The ND network was designed with a three-layer architecture to recognize the training data shown in Figure 19. We used 20 neurons in the first layer and 10 neurons in the second layer. The number of neurons in the output layer must be two, since the input vectors $\mathbf{p}$ have two elements. This made the architecture of the ND network $2 - 20 - 10 - 2$. We then used the Levenberg-Marquardt algorithm [HaDe96] to train the ND network (note that we also tried the Bayesian regularization algorithm and the results were not much different). In this example, the mean-square error was driven down to the order of 10^{-11}. Figure 44 illustrates the errors of the ND network for the training data.

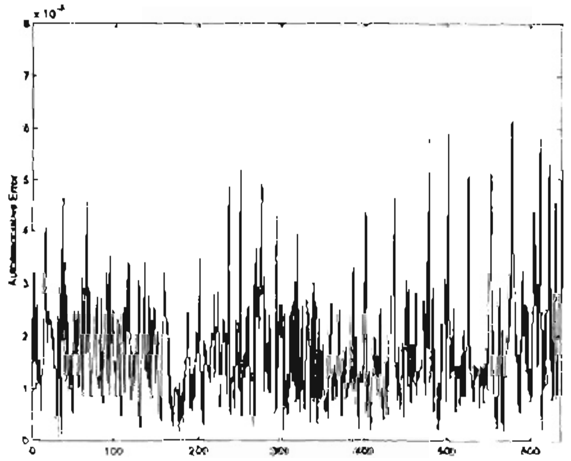

Figure 44 Autoassociative error of training data

The ND network generates output vectors that are very similar to the training inputs. The maximum autoassociative error for the training data was 6.13×10^{-5}. After the training process of the ND network is complete, the next step is to apply testing data $\mathbf{p}'$ to the ND network and to compute the autoassociative errors – the error between the testing data

p' and the corresponding ND-network outputs. Now, we want to see whether the error from the function approximator correlates with the autoassociative error. Figure 45 shows the correlation between the autoassociative errors from the ND network and the errors from the function approximator on the test data points shown in Figure 22.

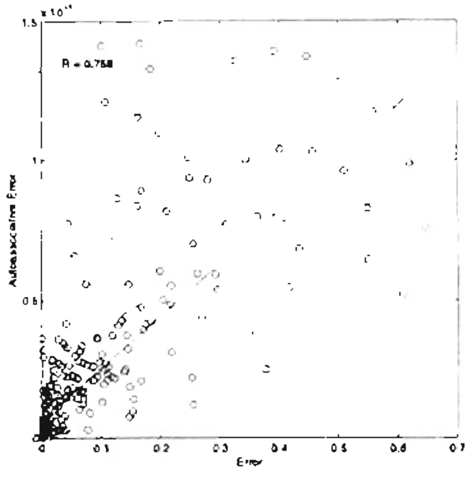

Figure 45 Autoassociative Error and Approximation Errors

As seen in Figure 45, the correlation (indicated by the R value) between the autoassociative errors and the errors from the function approximator was somewhat high, implying that there exists some correlation between these two variables.

Once again, to set up the threshold to discard abnormalities, we consider the regression line shown in Figure 45. The equation for the line in Figure 45 is

$$autoassociative\ error = 0.00189 \times error + 6.66 \times 10^{-5} \tag{42}$$

From Equation (42), the autoassociative error that corresponds to a network error of 0.15 is 3.501×10^{-4}. We will use this value as the threshold to reject novel data. Figure 46 demonstrates the network errors, and novel data are flagged with an **x**.

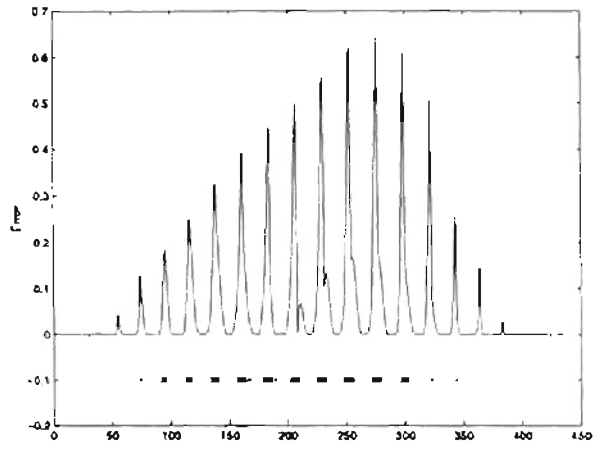

Figure 46 Error and abnormalities

As with previous methods, there are two types of misclassifications. In type I misclassification, the autoassociative error is greater than 3.501×10^{-4} (data flagged as novel), but the approximation error is less than 0.15. In type II misclassifications, the autoassociative error is less than 3.501×10^{-4}, but the approximation error is greater than 0.15. For this test, the total misclassification rate was 6.63%. Most of these misclassifications (4.81%) was of type I.

If we choose a threshold of 4.0×10^{-4}, the percentage of misclassifications is 5.72%. Around 3.41% of this 5.72% is from type I error. We can see that type I misclassifications are reduced when we increase the threshold. It should be noted that it is very difficult to have a threshold minimizing the percentage of misclassifications for every data set. However, what we can say in general about setting the threshold is that the *higher* the autoassociative error we choose for the threshold, the more likely we will reduce type I misclassifications (type II error increases). On the other hand, the *lower* the autoassociative error we select for the threshold, the more likely we will reduce type II misclassification (type I error increases). It will depend on the application what type of misclassifications we can tolerate. For this thesis, we will use the threshold based upon the regression line.

From the simulation results, we can conclude that the autoassociative errors from the ND network are correlated with the network errors. As we can see in Figure 45, some data points generating large approximation errors have small autoassociative errors, while some creating acceptable approximation errors make large autoassociative errors. These data points will increase the percentage of misclassifications. Therefore, in the next section, we will propose a technique to reduce the percentage of misclassifications.

A technique to improve efficiency

In the previous chapter, we described our technique to improve the efficiency of the Gaussian kernel estimator. We will use a similar procedure in this chapter. For our training data, we will incorporate the target with the input to create new training vectors $\Gamma_i = \begin{bmatrix}\mathbf{p}_i & t_i\end{bmatrix}^T$, and we will incorporate the output of the function approximator to make new testing data $\Gamma'_j = \begin{bmatrix}\mathbf{p}'_j & a'_j\end{bmatrix}^T$. That means that we will train an ND network with the new augmented training data Γ. Furthermore, in order to get new testing data, we need to propagate our testing input $\mathbf{p}'_j$ through the function approximator to get the network output a'_j, and then stack these two variables. Finally, these new testing data will be applied to be the inputs of the ND network. The advantage of using this method is described in the following.

If we have testing data $\mathbf{p}_j'$ close to the training data $\mathbf{p}$, but the function approximator turns out an eccentric output, the ND network will generate large autoassociative error. For example, assume that the function approximator creates a large error for one of its training points, $\mathbf{p}_i$ ($|t_i - a_i|$ is big). We also assume that the original ND network can recognize our training data $\mathbf{p}_i$ (output of the ND network is very similar to the input $\mathbf{p}_i$). The autoassociative error from the original ND network is thus small, since $\mathbf{p}_i$ is in the constrained hyperplane (since it is a training vector for the ND network). However, when we use the new ND network trained with the augmented training vectors Γ, the autoassociative error from the ND network for $\begin{bmatrix}\mathbf{p}_i & a_i\end{bmatrix}^T$ should be large, since $\begin{bmatrix}\mathbf{p}_i & a_i\end{bmatrix}^T$ is not close to our training data and thus is not in the constrained hyperplane (rather, $\Gamma_i = \begin{bmatrix}\mathbf{p}_i & t_i\end{bmatrix}^T$ is in the constrained hyperplane). Therefore, we would be able to reject this point as abnormal for the function approximator.

In the next section, we will use the new training data set Γ to train another ND network, and will apply the new testing data Γ' to the ND network. We will also show the correlation between the autoassociative error and the network error and will discuss whether or not it is improved over the original method.

Simulation of the simple example #2

In this example, the new data set is now three-dimensional. The number of neurons in the hidden layers remain the same as in the previous example. Therefore, the ND network has the structure $3 - 20 - 10 - 3$. We will again use the Levenberg-Marquardt algorithm for training the ND network to recognize the data set Γ. The mean-square error was driven down to about the same level as in the previous case, and the maximum autoassociative error from the training data was 6.14×10^{-5}.

After the ND network was trained, we applied the testing data set Γ' to the ND network. Note that we propagate testing inputs $\mathbf{p}'$ (shown in Figure 22) through the function approximator to obtain the network outputs a', and then stack them to obtain the inputs to the ND network. Figure 47 demonstrates the correlation between the errors of the function approximator and the autoassociative error from this ND network.

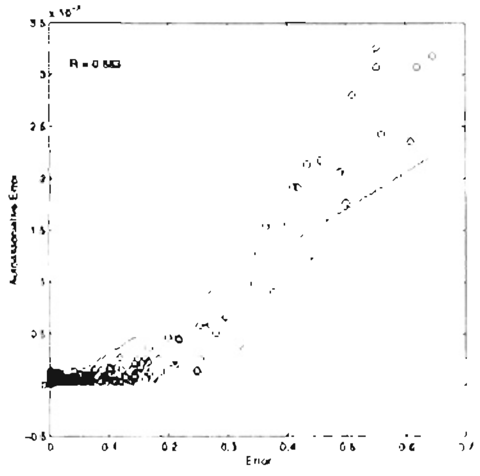

Figure 47 Autoassociative Error versus Approximation Error

The correlation coefficient value in this case was higher than the previous case (0.883 vs. 0.756). Also, the data points generating large errors are more distinguishable from data points creating small errors (compare Figure 45 and Figure 47).

Again, we consider the regression line in order to set up the threshold for rejecting novel data. From Figure 47, the regression line is

$$autoassociative\ error = 0.0035 \times error - 4.06 \times 10^{-5} \qquad (43)$$

From Equation (43), the autoassociative error corresponding to the network error of 0.15 is 4.844×10^{-4}. We will use this value as the threshold for our novelty detector. In Figure 48 we plot the approximation error on the testing set and indicate with an **x** all points that have autoassociative error greater than 4.844×10^{-4}.

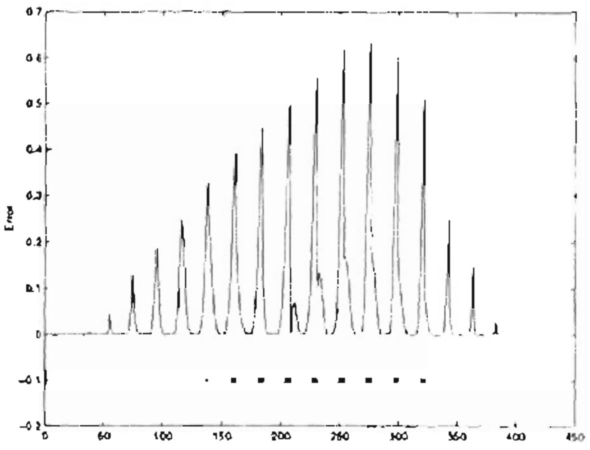

Figure 48 Novelty Detection

In this case, the total percentage of misclassified points was reduced to 5.72% (compared with 6.63% with the previous method). All of the misclassifications in this case were from type II.

Note that some other threshold may reduce the percentage of misclassifications. For example, if we choose the threshold to be 2×10^{-4}, the percentage of misclassifications is

3.66%. Around 2.97% of the 3.66% were type II error. There is no general threshold to minimize the percentage of misclassifications for every data set. However, when we reduce the threshold from 4.844×10^{-4} to 2×10^{-4}, type II error decreases and type I error increases. Therefore, the *higher* the autoassociative error threshold, the higher the type II errors (and less type I errors) will decrease. On the other hand, the *lower* the autoassociative error threshold, the more likely type I error will be increased (and type II errors decreased).

As demonstrated in this experiment, the technique of augmenting the input vectors with the target output does reduce the percentage of misclassifications. The percentage error for this method is lower than any other method we tested, and the method does not require us to set parameters (as in the smoothing parameter of the Gaussian kernel estimator). However, the training time for this method can be very long.

Summary

In this chapter, the multilayer neural network architecture was briefly reviewed. We slightly changed the multilayer structure to fit the novelty detection problem. The modified structure has the number of neurons of the output layer equal to the input dimension and the target output of the novelty detection (ND) network was the same as the network input. The error of the ND network was called the autoassociative error.

We used the autoassociative error to decide whether or not we would discard data as too novel for accurate function approximation. The simulation results showed that when the autoassociative error from the ND network was large, the more likely we would find large errors for the function approximator.

Finally, we introduced a new technique to improve the performance of this algorithm by making use of network outputs and targets (as in the previous chapter). The simulation outcomes demonstrated a better performance in terms of fewer misclassifications.

CHAPTER 6

MINIMUM DISTANCE ALGORITHM

Introduction

In this chapter, a new novelty-detection method will be introduced. The fundamental idea behind this method is based on the 2-norm between training vectors and testing vectors. We will first describe how the new novelty detection method can be employed, followed by simulation results. After that, we will propose a technique to improve the efficiency of this algorithm based on a derivation of the error equation. The simulation results of the modified version will be illustrated. Finally, a procedure to speed up the computation will be described.

Minimum Distance Computation

This algorithm is based on the idea that any two vectors that are close together would contain similar properties (information), therefore the function approximator should turn out comparable outcomes. The measurement of distance between any two vectors will be computed by the 2-norm of the difference between the two vectors. This 2-norm is written as:

$$\|\mathbf{a}_i - \mathbf{b}_j\| = [(\mathbf{a}_i - \mathbf{b}_j)^T(\mathbf{a}_i - \mathbf{b}_j)]^{1/2} \tag{44}$$

where $\mathbf{a}_i$ and $\mathbf{b}_j$ are the two vectors that we will measure the distance between.

As discussed in previous chapters, the objective of this thesis is to develop procedures to detect when a new input to a multilayer network function approximator is unlike inputs contained in the training set for the approximator. When we have a new input to the multilayer network, we will want to know how close that input is to inputs contained in the training set. For that purpose we will measure the distance from the new input to every input in the training set. The minimum of these distances will be used to measure the similarity of the new input to training inputs.

We will explain how the concept of minimum distance can be applied to novelty detection in the following section. We will also provide computer simulations of this technique.

Application to novelty detection

Recall that we need a novelty detector since we would like to flag data on which the multilayer function approximation network will perform poorly (generate large error). In this chapter we will use the minimum distance algorithm as a novelty detector. The main idea of this method is the following. Because errors on the training data are made small during the training process, any new data that is similar to some training data (has a small minimum distance) should also produce a small approximation error. On the other hand, new data with a large minimum distance to the training set can be expected to produce a large error for the function approximator. Therefore, we will flag any data for which the minimum distance is high.

Equation (45) expresses the distances from a testing vector $\mathbf{p}_j'$ to all of the N training data inputs $\mathbf{p}_i$

$$\mathbf{d} = \begin{bmatrix} \left\|\mathbf{p}_1 - \mathbf{p}_j^t\right\| \\ \left\|\mathbf{p}_2 - \mathbf{p}_j^t\right\| \\ \vdots \\ \left\|\mathbf{p}_N - \mathbf{p}_j^t\right\| \end{bmatrix} \qquad (45)$$

The minimum distance is the minimum element of the above vector

$$d_m = min(\mathbf{d}) \qquad (46)$$

As we explained earlier, we expect that the bigger the value of d_m, the more likely there will be a large error for the function approximator. Note that if the testing vector is one of the training vectors, the minimum distance d_m will be equal to zero, and we will accept such data since the error of any training vectors should be small.

In the next section, we will revisit the example we have used in the previous chapters. We will use it to test the minimum distance method, Equation (45) and Equation (46), for novelty detection.

Simulation of the simple example #1

The following simulations will illustrate the results of novelty detection employing the minimum distance algorithm. We use Equation (45) to find the distances of each testing vector to the 638 training vectors (shown in Figure 19), and then Equation (46) is utilized to find the minimum distance. We repeat this method for all of the 437 testing vectors (shown in Figure 22). Figure 49 shows the minimum distance values of all 437 testing vectors versus the errors obtained from the function approximator.

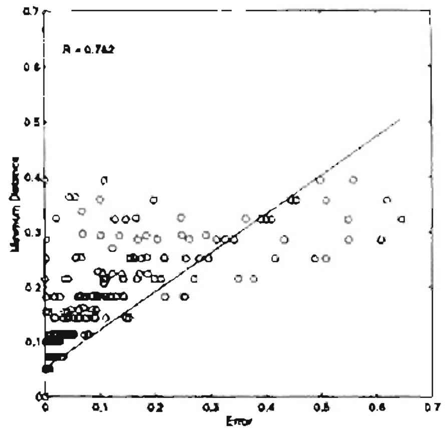

Figure 49 Minimum Distance versus Error

From the figure, we can see that as the minimum distance value increases, the more likely we will find large errors. Also, the fairly high correlation coefficient (R value) implies that there exists a correlation between the minimum distance and the error from the function approximator.

To create a threshold to reject novel data, the regression line shown in Figure 49 will be used. We found that the regression line in this case was

$$d_m = 0.702 \times error + 0.0518 \tag{47}$$

After substituting 0.15 in the error term, the corresponding minimum distance equals 0.1571. (We will be using 0.15 as an arbitrary point to represent large approximation error.) That indicates that, on the average for this data set, data points generating an approximation error of 0.15 are a distance of 0.1571 from their closest training vectors. We will use this value to be the threshold for the novelty detector. In other words, any data generating a minimum distance larger than 0.1571 will be considered as novel data. Figure 50 demonstrates the graph showing the network errors for this testing data, and novel data are flagged with an **x**.

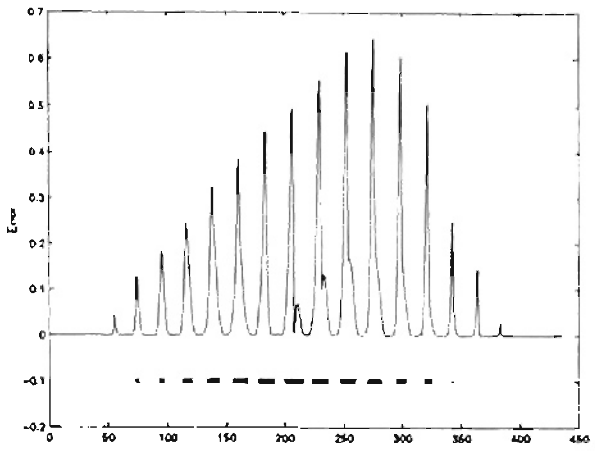

Figure 50 Error and abnormalities

There are two types of misclassifications. In type I misclassifications, the minimum distance is greater than 0.1571 (data flagged as novel), but the approximation error is less than 0.15. In the type II misclassification, the distance is less than 0.1571, but the approximation error is greater than 0.15. For this test case, the total misclassification rate was 12.82%, and most of this misclassification (12.58%) was of type I.

We found that the threshold that produces the fewest misclassifications in this example is 0.25. The error rate is 6.87% and around 3.89% (out of 6.87%) are type I errors. Although the purpose of using the testing data is to set up the threshold to reject novel data, it should be noted that it is very difficult to have a general value for the threshold that will minimize the percentage of misclassifications for all data sets. What we can generally say about setting the threshold is that the *lower* the threshold, the more likely we will experience type I errors. On the other hand, the *higher* the threshold, the more likely we will have type II errors. Therefore, the threshold of 0.25 tends to give us more type II errors than the threshold of 0.1571. That means that if another test set is quite similar to this test set, the threshold of 0.25 would be better than 0.1571. In contrast, if another test set is different from this test set (i.e. it has data with large errors and small minimum distances), the threshold of 0.1571 should outperform. However, in our case, we will use the threshold based on the regression line.

From the simulation results in this example, we can see that there is a relationship between the minimum distance and the error from the function approximator. However, as shown in Figure 49, there exist some points creating large approximation errors but having minimum distances that are similar to data points that have small approximation errors. In the next section, we will explain a technique to improve the performance of this algorithm.

A Technique to Improve Performance: Minimum Weighted Distance

In the previous chapter, we proposed that instead of just using network inputs, the network outputs a and targets t could be also used for novelty detection. In this chapter, we will modify the minimum distance algorithm in order to include network outputs. First, we will start this section by deriving the error equation for the function approximator, using the Taylor's series expansion.

For an input vector $\mathbf{p}_j^l$ to the function approximator, the error between the corresponding network output a_j^l and the target t_j^l can be written as:

$$\begin{aligned} e_j^l &= t_j^l - a_j^l \\ &= F(\mathbf{p}_j^l) - a_j^l \end{aligned} \tag{48}$$

where F is the function we want to approximate.

Now we will use the Taylor's series to expand the function F about the training input vector that is closest to the vector $\mathbf{p}_j^t$. Therefore, Equation (48) can be rewritten as follows:

$$\begin{aligned} e_j^t &= F(\mathbf{p}_0) + \left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^T\right|_{\mathbf{p}=\mathbf{p}_0} (\mathbf{p}_j^t - \mathbf{p}_0) + \ldots - a_j^t \\ &= (F(\mathbf{p}_0) - a_j^t) + \left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^T\right|_{\mathbf{p}=\mathbf{p}_0} (\mathbf{p}_j^t - \mathbf{p}_0) + \ldots \\ &= (t_0 - a_j^t) + \left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^T\right|_{\mathbf{p}=\mathbf{p}_0} (\mathbf{p}_j^t - \mathbf{p}_0) + \ldots \end{aligned} \tag{49}$$

where $\mathbf{p}_0$ is a training vector, and $\left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^T\right|_{\mathbf{p}=\mathbf{p}_0}$ is the first derivative of the function with respect to the input and is evaluated at the training vector.

Recall that by using Equation (45) and Equation (46) to find the minimum distance, we are able to indicate which training vector is closest to the testing vector. Let us assume that $\mathbf{p}_m$ is the training vector closest to $\mathbf{p}_j^t$. In other words, $\mathbf{p}_m$ can be found by finding $\mathbf{p}_m = \underset{\mathbf{p}}{\arg min}(\mathbf{d})$. Thus, the last expression in Equation (49) can be written as:

$$e_j^t = (t_m - a_j^t) + \left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^T\right|_{\mathbf{p}=\mathbf{p}_m} (\mathbf{p}_j^t - \mathbf{p}_m) + \ldots \tag{50}$$

Now, if we assume that the higher-order terms in Equation (50) can be ignored, then we can rewrite it as:

$$e_j^t \simeq (t_m - a_j^t) + \left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^T\right|_{\mathbf{p}=\mathbf{p}_m} (\mathbf{p}_j^t - \mathbf{p}_m) \tag{51}$$

We can see from Equation (51) that the error from the function approximator e_j^t does not depend only on the distance between testing vector $\mathbf{p}_j^t$ and the closest training vector $\mathbf{p}_m$, but also on the gradient evaluated at the training vector $\left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^T\right|_{\mathbf{p} = \mathbf{p}_m}$ and on the distance between the target t_m and the network output of the testing vector a_j^t. Therefore, if the underlying function F at the training vector $\mathbf{p}_m$ is fairly steep (the derivative at the point is very large), the error of the function approximator could be large even though the minimum distance between the testing vector and the training vector is small. Similarly, if the distance between target t_m and a_j^t is large even though the distance between these input vectors is small, the error can be large.

Unfortunately, the underlying function F in general is unknown, therefore it is impossible to calculate its gradient (which is a vector of size $q \times 1$) at any training data point. We first considered calculating the derivative of the function approximator $\hat{F}$; however, we found that this did not improve the novelty detection. We will explain in the next section how to calculate the estimated derivative. We will also explain why we cannot use it for novelty detection, and how it can be used for other work.

Though Equation (51) is not directly useful for error estimation, it gives us an idea that the error depends not only on the distance between the two input vectors. Therefore, we will modify our previous distances $\mathbf{d}$ (Equation (45)) to make them depend on another computable term – the distance between target and network output. Equation (52) shows our new measurements:

$$\mathbf{d}^{\alpha} = \begin{bmatrix} \|\mathbf{p}_j' - \mathbf{p}_1\| + \alpha\|t_1 - a_j'\| \\ \|\mathbf{p}_j' - \mathbf{p}_2\| + \alpha\|t_2 - a_j'\| \\ \vdots \\ \|\mathbf{p}_j' - \mathbf{p}_N\| + \alpha\|t_N - a_j'\| \end{bmatrix} \tag{52}$$

where α is greater than or equal to zero ($\alpha \geq 0$). We will call the elements of $\mathbf{d}^{\alpha}$ weighted distances, because they include the distance between the targets and network outputs. As before, we will obtain the minimum value of the weighted distances by computing

$$d_m^{\alpha} = min(\mathbf{d}^{\alpha}) \tag{53}$$

We will call d_m^{α} the minimum weighted distance. The effect of α will be discussed further.

As shown in Equation (52), if we set $\alpha = 0$, the minimum weighted distance will be equal to the minimum distance. Increasing the weighting factor will strengthen the effect of the difference between targets and network outputs and will neutralize the distance of the input data. For example, assume that $\mathbf{p}_k$ is a *training* data point that generates large error, thereby making the value $\|t_k - a_k\|$ high. When we substitute $\mathbf{p}_j'$ with $\mathbf{p}_k$ in Equation (45), the term $\|\mathbf{p}_j' - \mathbf{p}_k\|$ will be zero, thus causing zero minimum distance ($d_m = 0$). However, the weighted minimum distance will not equal zero, because $\alpha\|t_k - a_k\|$ will not be zero. Choosing an appropriate value of α will be important for successful novelty detection. We will discuss this further in the next section.

We will demonstrate the ability of this method for novelty detection using our previous example. The effect of α on the performance of the novelty detector will be demonstrated through computer simulations.

Simulation of the simple example #2

We will illustrate the use of the minimum weighted distance computation for novelty detection through a simple example. We will show the effect of varying α on the correlation coefficient R between the error of the function approximator and the minimum weighted distance.

First, we need to propagate a testing vector $\mathbf{p}_j^t$ through the function approximator to get the network output a_j^t. Equation (52) and Equation (53) are then used to find the minimum weighted distance at a specific value of α for the 437 testing vectors. After that, we compute the correlation coefficient (R) between the two variables – the approximation error and the minimum weighted distance – for the 437 testing data. Figure 51 demonstrates the relationship between the weighting factor and the correlation coefficient.

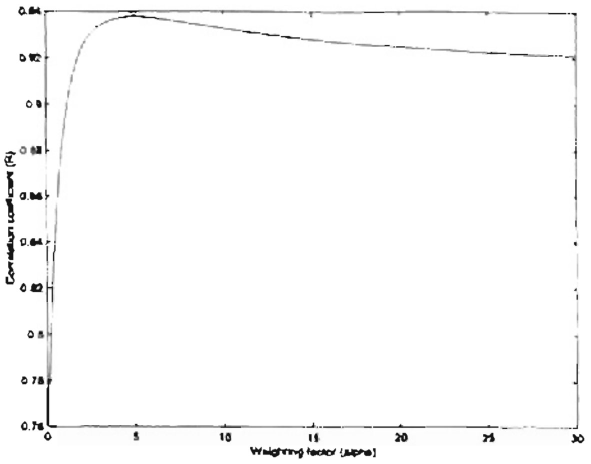

Figure 51 Effect of the weighting factor to R

We can see from Figure 51 that when the value of α is increased, the correlation coefficient R is raised as well. This is because the effect of distance between target and network input is added to the distance between the input vectors. And as we expected, when the value of α is too large (more than 5.1 in this case), the R value begins to drop. We expect that the curve shown in Figure 51 may change if we use a different testing set. However, for our simulations, we will choose the weighting factor that maximizes the correlation coefficient in this test set ($\alpha = 5.1$).

After choosing the value of $\alpha = 5.1$, we plotted the error from the function approximator and the minimum weighted distance in Figure 52.

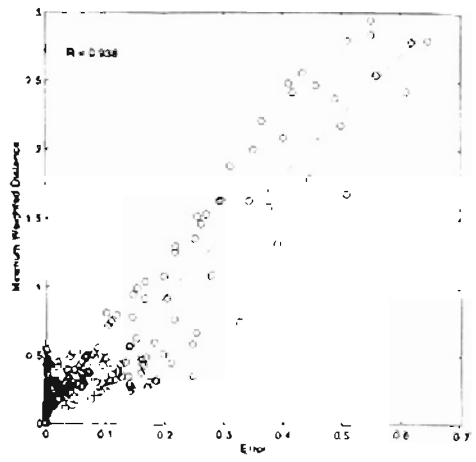

Figure 52 Approximation Error and minimum weighted distance

As shown in Figure 52, the correlation coefficient is very high, and most data points follow the regression line. Therefore, the regression line in this case may be able to represent these two variables more *precisely* than the regression line shown in Figure 49.

From the result shown in Figure 52, the regression line was

$$d_m^{5.1} = 4.37 \times error + 0.0643 \tag{54}$$

For the error of 0.15, the minimum weighted distance is equal to 0.7198. That means that, on the average for this data set, the error of 0.15 corresponds to the minimum weighted distance of 0.7198. Therefore, in order to reject any data generating errors greater than 0.15, we will set up the threshold so that any data generating minimum weighted distance (with $\alpha = 5.1$) larger than 0.7198 will be considered as novel. Figure 53 plots the network errors for the testing data, and novel data are identified with an **x**.

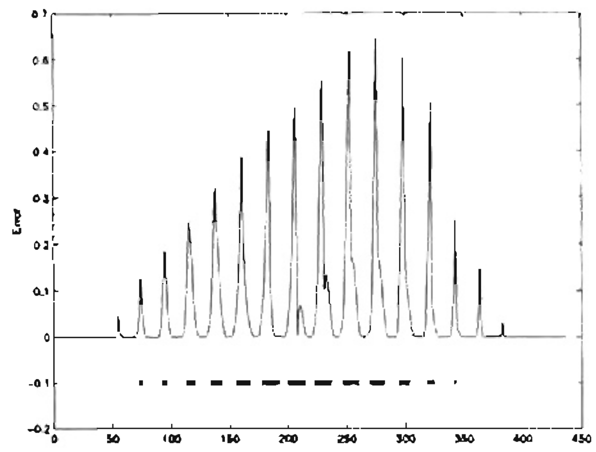

Figure 53 Novelty detection

There were 5.03% of misclassified points using this method. We found that 3.66% out of 5.03% misclassifications were type I errors. The rest of the misclassified data points (1.37%) were type II.

If we choose the threshold to be 0.9 rather than 0.7198, we will get only 4.34% misclassified data. Most of the misclassifications (around 4.11% out of 4.34%) are type II error. As we can see, though the total percentage of misclassifications is reduced by a certain amount (4.34% versus 5.03%), the percentage of type II is fairly increased (4.11% versus 1.37%). It will depend on the application what types of misclassifications one can tolerate. However, in our case, we will stick with the threshold obtained from the regression line.

As we can see from the simulation results, the percentage of misclassifications was reduced after we included the distance between targets and network outputs to the distance measure. The execution time in this case was slightly longer than computing minimum distance.

Note that although this algorithm worked somewhat well for this data set, we found that the weighting factor we used in the example was not always best for other data sets. Therefore, if we computed the percentage of misclassifications on another data set, its value would not be this low. We will introduce another novelty detection method in the next chapter – minimum distance with outlier detection. We found that a parameter in that method worked well for a variety of data sets.

Recall that we introduced the minimum weighted distance because the error from the function approximator depends on three terms, and the derivative term in Equation (51) is unknown. Furthermore, we briefly mentioned that the estimated derivative is not helpful for novelty detection. In the next section, we will explain in detail how to compute the estimated derivative of the function approximator, the reason why the obtained value is not useful for novelty detection, and what is the advantage of finding the estimated derivative.

Calculating the estimated derivative

Recall from Equation (48) that we began with the error equation and expanded it using the Taylor's series. We then searched for the training vector *closest* to the testing input, $\mathbf{p}_m = \underset{\mathbf{p}}{\arg min}(\mathbf{d})$, so that the higher-order terms of the series could be ignored, resulting in Equation (51). Now, since having the multilayer network to approximate the original function F, we came up with an idea to estimate the derivative of a function F from the function approximator $\hat{F}$. Then the estimated derivative will be substituted into Equation (51) to obtain the estimated error. The method to derive the estimated derivative follows.

Apply the chain rule to the k-layer network $\hat{F}$ (function approximator):

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{p}}\hat{F}(\mathbf{p}) &= \frac{\partial}{\partial \mathbf{p}}a^k(\mathbf{n}^k) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}n^k(\mathbf{a}^{k-1})\right)^T \frac{\partial}{\partial n^k}a^k(n^k) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}n^k(\mathbf{a}^{k-1})\right)^T \dot{f}^k(n^k) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}\mathbf{a}^{k-1}(\mathbf{n}^{k-1})\right)^T \frac{\partial}{\partial \mathbf{a}^{k-1}}n^k(\mathbf{a}^{k-1})\dot{f}^k(n^k) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}\mathbf{a}^{k-1}(\mathbf{n}^{k-1})\right)^T \mathbf{W}^{k^T}\dot{f}^k(n^k) \qquad\qquad (55) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}\mathbf{n}^{k-1}(\mathbf{a}^{k-2})\right)^T \frac{\partial}{\partial \mathbf{n}^{k-1}}\mathbf{a}^{k-1}(\mathbf{n}^{k-1})\mathbf{W}^{k^T}\dot{f}^k(n^k) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}\mathbf{n}^{k-1}(\mathbf{a}^{k-2})\right)^T \dot{\mathbf{F}}^{k-1}(\mathbf{n}^{k-1})\mathbf{W}^{k^T}\dot{f}^k(n^k) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}\mathbf{a}^{k-2}(\mathbf{n}^{k-2})\right)^T \frac{\partial}{\partial \mathbf{a}^{k-2}}\mathbf{n}^{k-1}(\mathbf{a}^{k-2})\dot{\mathbf{F}}^{k-1}(\mathbf{n}^{k-1})\mathbf{W}^{k^T}\dot{f}^k(n^k) \\
&= \left(\frac{\partial}{\partial \mathbf{p}}\mathbf{a}^{k-2}(\mathbf{n}^{k-2})\right)^T \mathbf{W}^{k-1^T}\dot{\mathbf{F}}^{k-1}(\mathbf{n}^{k-1})\mathbf{W}^{k^T}\dot{f}^k(n^k)
\end{aligned}$$

where $\dot{\mathbf{F}}^l(\mathbf{n}^l)$ is the derivative of the transfer function f at layer l:

$$\dot{\mathbf{F}}^l(\mathbf{n}^l) = \begin{bmatrix} \frac{\partial}{\partial n_1^l}f^l(n_1^l) & 0 & \dots & 0 \\ 0 & \frac{\partial}{\partial n_2^l}f^l(n_2^l) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \frac{\partial}{\partial n_{S^l}^l}f^l(n_{S^l}^l) \end{bmatrix} \qquad (56)$$

For example, if we use the hyperbolic tangent sigmoid function at layer l, Equation (56) becomes

$$\dot{\mathbf{F}}^{l}(\mathbf{n}^{l}) = \begin{bmatrix} a_1^l(1-a_1^l) & 0 & \dots & 0 \\ 0 & a_2^l(1-a_2^l) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & a_{S^l}^l(1-a_{S^l}^l) \end{bmatrix} \tag{57}$$

If we continue to apply the chain rule to the first term in Equation (55), we will eventually obtain the derivative of the function approximator (the k-layer network):

$$\frac{\partial}{\partial \mathbf{p}}\hat{F}(\mathbf{p}) = \mathbf{W}^{1^T}\dot{\mathbf{F}}^{1}(\mathbf{n}^{1})\mathbf{W}^{2^T}\dot{\mathbf{F}}^{2}(\mathbf{n}^{2})\dots\mathbf{W}^{k-1^T}\dot{\mathbf{F}}^{k-1}(\mathbf{n}^{k-1})\mathbf{W}^{k^T}\dot{f}^{k}(n^{k}) \tag{58}$$

We can use the above equation to estimate the derivative of the function F at any training data point, $\left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^{T}\right|_{\mathbf{p}=\mathbf{p}_m}$, by using the following equation:

$$\begin{aligned} \left.\frac{\partial F(\mathbf{p})}{\partial \mathbf{p}}^{T}\right|_{\mathbf{p}=\mathbf{p}_m} &\cong \left.\frac{\partial \hat{F}(\mathbf{p})}{\partial \mathbf{p}}^{T}\right|_{\mathbf{p}=\mathbf{p}_m} \\ &= [\mathbf{W}^{1^T}\dot{\mathbf{F}}^{1}(\mathbf{n}^{1})\mathbf{W}^{2^T}\dot{\mathbf{F}}^{2}(\mathbf{n}^{2})\dots\mathbf{W}^{k^T}\dot{f}^{k}(n^{k})]^{T}\Big|_{\mathbf{p}=\mathbf{p}_m} \\ &= \left[\mathbf{W}^{1^T}\dot{\mathbf{F}}^{1}(\mathbf{n}^{1})\Big|_{\mathbf{n}_m^1}\mathbf{W}^{2^T}\dot{\mathbf{F}}^{2}(\mathbf{n}^{2})\Big|_{\mathbf{n}_m^2}\dots\mathbf{W}^{k^T}\dot{f}^{k}(n^{k})\Big|_{n_m^k}\right]^{T} \end{aligned} \tag{59}$$

where $\mathbf{n}_m^1 = \mathbf{W}^1\mathbf{p}_m + \mathbf{b}^1$ and $\mathbf{n}_m^l = \mathbf{W}^l\mathbf{F}^{l-1}(\mathbf{n}_m^{l-1}) + \mathbf{b}^{l-1}$. Note again that $\mathbf{p}_m$ is obtained by computing $\mathbf{p}_m = \underset{\mathbf{p}}{\arg min}(\mathbf{d})$.

By substituting Equation (59) into Equation (51), we expected that the error could be estimated. Consequently, we would be able to reject the input vector $\mathbf{p}_l'$ as novel if the estimated error is large. Unfortunately, we found that this estimated derivative (shown in

Equation (59)) will not accurately approximate the error for extrapolations. For example, refer to Figure 12 and Figure 13. Though the function approximator could accurately predict the values between $p = -1$ and $p = 1$, the estimated derivative at training points $p = -1$ and $p = 1$ were incorrect. (Note that the estimated derivatives were very accurate for all other training points.) Therefore, when using Equation (51) and Equation (59) to estimate the error for extrapolations (for inputs greater than 1 or less than -1), we will obtain incorrect estimated errors. This is due to the fact that the derivative we calculated is for the function $\hat{F}$, not for the underlying function F. We found that Equation (51) and Equation (59) are very good tools to *estimate* the error for *interpolations*, but this is not the objective of this thesis.

For real world applications in which the underlying function we want to estimate is unknown, we cannot check whether or not we have the precise estimated derivative at the training data points. Therefore, we finally conclude that the estimated derivative is not generally useful for novelty detection.

How to speed up distance computation

We discussed earlier that the execution time of a novelty detector can sometimes be long. In this section, we will propose a technique to reduce the execution time for the minimum distance algorithm by using the Kohonen rule. The following section will briefly describe what the Kohonen rule is and how we can use this for novelty detection. The details of the Kohonen rule can also be found in [HaDe96].

The Kohonen rule

The Kohonen rule is an algorithm that can be applied to a certain type of neural network. This algorithm allows unsupervised learning, meaning that there are no targets for corresponding inputs. The rule gives the network the capability to learn associations between data that are similar. After learning, the network will be able to perform some tasks such as pattern recognition [HaDe96]. The following equation expresses the Kohonen rule:

$$ {}_i\mathbf{w}(t) = {}_i\mathbf{w}(t-1) + \eta(\mathbf{p}(t) - {}_i\mathbf{w}(t-1)) \qquad (60) $$

where ${}_i\mathbf{w}$ is the "i^{th} cluster center", $\mathbf{p}$ is the input vector, t is the iteration number, and η is the learning rate. Only the cluster center that is closest to the input vector is updated.

We will first initialize our cluster centers by randomly selecting them from our input vectors, or by setting them all to zero. When an input vector $\mathbf{p}(t)$ is closest to the i^{th} cluster center, the learning, Equation (60), occurs by moving the i^{th} cluster center toward the input vector. Figure 54 demonstrates how a cluster center learns on an input vector. We can see that the cluster center moves along a line between the old cluster center and the input vector.

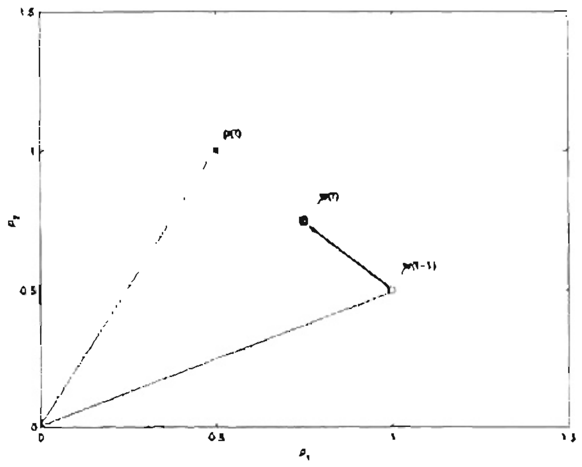

Figure 54 Graphical Representation of the Kohonen Rule

Normally, the number of cluster centers is much less than the number of data points, so that each cluster center will represent nearby data. Figure 55 shows data and four cluster centers obtained after using Equation (60) for 500 iterations. We can see from this figure that each cluster center is located near the middle of each data group. The explanation of how we can use this algorithm to speed up our minimum distance computation follows.

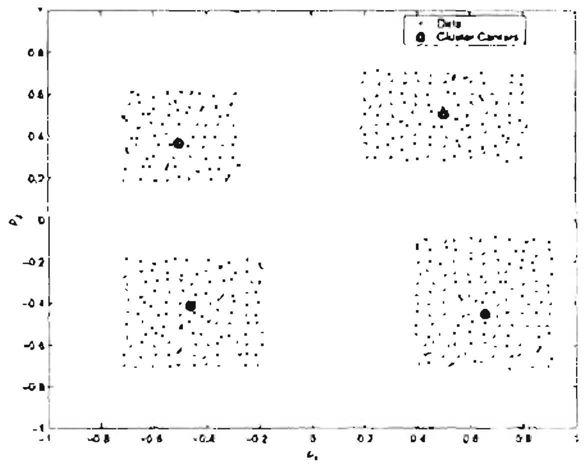

Figure 55 Data and cluster centers

How to use the Kohonen rule to speed up distance computation

We first assume that data points shown in Figure 55 are the two-dimensional N training input vectors $\mathbf{p}$ for a function approximator. Assume that we have a testing vector $\mathbf{p}_j^t = \begin{bmatrix} -0.8 & 0.9 \end{bmatrix}^T$ whose the minimum distance needs to be computed. We have to calculate the distances between the testing vector and all of the training vectors. We can see that some training vectors are clearly far away from the testing vector, but we still have to compute all distances. Rather than calculating distances to every training data point, we first compute distances between the testing vector and the cluster centers, and then select the minimum one. Assume cluster center ${}_j\mathbf{w}$ is closest to the testing vector. We can now quickly obtain the minimum distance d_m from the testing vector to the entire training set by finding the minimum distance from the vector to only the training data *within* the ${}_j\mathbf{w}$ cluster center. For example, for the testing vector $\mathbf{p}_j^t = \begin{bmatrix} -0.8 & 0.9 \end{bmatrix}^T$, we will obtain the minimum distance by calculating distances from $\mathbf{p}_j^t$ to the training data located in the fourth quadrant. From data shown in Figure 55, if the number of training data within each cluster center (over each quadrant in this case) is about the same, the execution time will be four times faster. Note that the decreased computation time is proportional to the number of cluster centers.

In order to use this method, we need to choose the number of cluster centers. If the number of cluster centers is small, there will be a large number of vectors in each cluster, and the computation of minimum distance may take too much time. If the number of cluster centers is large, it may take too much time to compute the minimum distance to the cluster centers. Also, the larger the number of cluster centers, the longer it may take to train the cluster centers. A compromise must be made to choose the optimal number of clusters.

Summary

We introduced a new method for novelty detection that calculates the minimum distance from a testing vector to the vector in the training set. The minimum distance can be used to reject novel points. This method was acceptable because the computation time was not too high and the percentage of misclassified points was small. We then proposed a technique to reduce the percentage of misclassified data by incorporating network outputs in the distance calculation.

We concluded from the analysis in this chapter that the estimated derivative was useful for error estimation when interpolating but was not appropriate for novelty detection. Finally, we proposed a way to speed up the computation time by applying the Kohonen rule to the training data set before calculating the minimum distance.

CHAPTER 7

MINIMUM DISTANCE AND OUTLIER DETECTION

Introduction

The intention of this chapter is to improve the efficiency of the minimum distance algorithm in terms of decreasing the percentage of misclassified data. In the previous chapter, we proposed the minimum weighted distance to improve the efficiency; however, the weighting factor with the highest correlation coefficient in general is unknown and varies from data set to data set. We found that the outlier detection using principal components is another approach that can reduce the percentage of misclassifications when we use it with the minimum distance algorithm. In addition, the parameter in the outlier detection method works well for many data sets.

This chapter will begin with the definition and derivation of principal components. We will then discuss outliers and will explain how to distinguish them by employing principal components. We then will describe how the outlier-detection method can be used for novelty detection through the minimum distance algorithm. Finally, a computer simulation employing a modified version of the minimum distance algorithm with outlier detection will be illustrated.

Principal component analysis

Principal component analysis was initially presented by [Pear01], and progressively developed by Hotelling, whose papers can be found in [Hotel33], and [Hotel36]. It is a well-known multivariate technique whose objective is to reduce the dimension of a data set, while preserving as much information and variation as possible. Each data vector is transformed to a principal component vector, in which the elements are uncorrelated, and ordered such that the first *few* principal components (PC) retain most of the variation in the original data. The following subsection will describe how to transform data to principal components.

Definition and derivation

First let's define some notation that will be used in this chapter. Assume that $\mathbf{x}$ is a $q \times 1$ random vector with computable covariance matrix. The first step in principal component analysis is to create a linear function of $\mathbf{x}$, which is denoted $\upsilon_1^T\mathbf{x}$ and is called the first PC, that has the maximum variance. Note that υ is a unit vector with q elements:

$$\upsilon_1^T = \begin{bmatrix}\upsilon_{11} \ \upsilon_{12} \ \dots \ \upsilon_{1q}\end{bmatrix} \tag{61}$$

The next step is to find another linear transformation $\upsilon_2^T\mathbf{x}$ that is uncorrelated with the first PC $\upsilon_1^T\mathbf{x}$ and that maximizes the variance. It is called the second PC. The process continues so that, at the k^{th} stage of transformation, a linear function of $\mathbf{x}$ having maximum variance is uncorrelated with the previous $k-1$ linear functions. In other words, the k^{th} PC, $\upsilon_k^T\mathbf{x}$, is uncorrelated with the previously-transformed variables $\upsilon_1^T\mathbf{x}, \upsilon_2^T\mathbf{x}, \ldots, \upsilon_{k-1}^T\mathbf{x}$. The vector υ_k^T has the form

$$\upsilon_k^T = \begin{bmatrix}\upsilon_{k1} & \upsilon_{k2} & \cdots & \upsilon_{kq}\end{bmatrix} \tag{62}$$

Assume that the transformation is recalculated until the r^{th} PC is found. Now, one might question how large r should go. The largest number that r can be is q, the dimension of the vector $\mathbf{x}$. However, it is hoped that r will be much less than q. It is hoped that $\mathbf{x}$ can be closely approximated by a small number of principal components.

We next want to show how to find the constant vectors υ_k^T to create the k^{th} PC. Suppose that a random vector $\mathbf{x}$ has covariance matrix Σ. Assume that the covariance matrix is written as

$$\Sigma = E[(\mathbf{x}-\mu_{\mathbf{x}})(\mathbf{x}-\mu_{\mathbf{x}})^T] = \begin{bmatrix}\varepsilon_{11} & \varepsilon_{12} & \cdots & \varepsilon_{1q}\\ \varepsilon_{21} & \varepsilon_{22} & \cdots & \varepsilon_{2q}\\ \vdots & \vdots & & \vdots\\ \varepsilon_{q1} & \varepsilon_{q2} & \cdots & \varepsilon_{qq}\end{bmatrix} \tag{63}$$

where ε_{ij} indicates the covariance between the i^{th} and j^{th} elements of $\mathbf{x}$ and can be calculated as,

$$\begin{aligned}\varepsilon_{ij} &= cov(x_i, x_j)\\ &= E[(x_i-\mu_{x_i})(x_j-\mu_{x_j})]\end{aligned} \tag{64}$$

where E is the expectation and $\mu_{\mathbf{x}} = E[\mathbf{x}] = \begin{bmatrix}\mu_{x_1} & \mu_{x_2} & \cdots & \mu_{x_q}\end{bmatrix}^T$.

Now, for the first PC, we want to find the a linear function of the random vector that has maximum variance, meaning that the transformation has to maximize the variance of $\upsilon_1^T\mathbf{x}$:

$$\begin{aligned} var(\upsilon_1^T\mathbf{x}) &= E[(\upsilon_1^T\mathbf{x} - E[\upsilon_1^T\mathbf{x}])(\upsilon_1^T\mathbf{x} - E[\upsilon_1^T\mathbf{x}])^T] \\ &= E\Big[(\upsilon_1^T\mathbf{x} - \upsilon_1^T\mu_{\mathbf{x}})(\upsilon_1^T\mathbf{x} - \upsilon_1^T\mu_{\mathbf{x}})^T\Big] \\ &= E[\upsilon_1^T(\mathbf{x} - \mu_{\mathbf{x}})(\mathbf{x} - \mu_{\mathbf{x}})^T\upsilon_1] \\ &= \upsilon_1^T E[(\mathbf{x} - \mu_{\mathbf{x}})(\mathbf{x} - \mu_{\mathbf{x}})^T]\upsilon_1 \\ &= \upsilon_1^T\Sigma\upsilon_1 \end{aligned} \tag{65}$$

That means that to maximize the variance of $\upsilon_1^T\mathbf{x}$ we must maximize $\upsilon_1^T\Sigma\upsilon_1$. This is a constrained maximization, since υ must be a unit vector. The constraint can be written $\upsilon_1^T\upsilon_1 = 1$. Using the method of Lagrange multipliers, we then maximize

$$\upsilon_1^T\Sigma\upsilon_1 - \lambda(\upsilon_1^T\upsilon_1 - 1) \tag{66}$$

where λ is a Lagrange multiplier. By differentiating Equation (66) with respect to υ_1 and setting it equal to zero, we obtain

$$(\Sigma - \lambda\mathbf{I}_q)\upsilon_1 = 0 \tag{67}$$

where $\mathbf{I}_q$ is a $q \times q$ identity matrix. From Equation (67),we can see that λ is an eigenvalue of Σ, υ_1 is the corresponding eigenvector, and the maximum value of Equation (65) becomes

$$\upsilon_1^T\Sigma\upsilon_1 = \upsilon_1^T\lambda\mathbf{I}_q\upsilon_1 = \lambda\upsilon_1^T\upsilon_1 = \lambda \tag{68}$$

Therefore, λ should be maximized for the first PC. In other words, the transformation for the first PC is the eigenvector of the covariance matrix that corresponds to the largest eigenvalue.

For the second PC, we need to find the linear function of $\mathbf{x}$, $\upsilon_2^T\mathbf{x}$, that has maximum variance; i.e. $\upsilon_2^T\Sigma\upsilon_2$, and is uncorrelated with the first PC $\upsilon_1^T\mathbf{x}$. As with first PC, we need to create a constraint $\upsilon_2^T\upsilon_2 = 1$. However, in this case, the correlation between $\upsilon_1^T\mathbf{x}$ and $\upsilon_2^T\mathbf{x}$ will be equal to zero.

$$\begin{aligned} cov(\upsilon_2^T\mathbf{x}, \upsilon_1^T\mathbf{x}) &= E[(\upsilon_2^T\mathbf{x} - E[\upsilon_2^T\mathbf{x}])(\upsilon_1^T\mathbf{x} - E[\upsilon_1^T\mathbf{x}])^T] \\ &= E\left[(\upsilon_2^T\mathbf{x} - \upsilon_2^T\mu_{\mathbf{x}})(\upsilon_1^T\mathbf{x} - \upsilon_1^T\mu_{\mathbf{x}})^T\right] \\ &= E[\upsilon_2^T(\mathbf{x} - \mu_{\mathbf{x}})(\mathbf{x} - \mu_{\mathbf{x}})^T\upsilon_1] \\ &= \upsilon_2^T E[(\mathbf{x} - \mu_{\mathbf{x}})(\mathbf{x} - \mu_{\mathbf{x}})^T]\upsilon_1 \\ &= \upsilon_2^T\Sigma\upsilon_1 \end{aligned} \tag{69}$$

From Equation (67), the above equation leads to

$$cov(\upsilon_2^T\mathbf{x}, \upsilon_1^T\mathbf{x}) = \upsilon_2^T\Sigma\upsilon_1 = \upsilon_2^T\lambda_1\mathbf{I}_q\upsilon_1 = \lambda_1\upsilon_2^T\upsilon_1 \tag{70}$$

Equation (70) becomes another constraint.

Now, we have two constraints: the first is $\upsilon_2^T\upsilon_2 = 1$ and the second is $\upsilon_2^T\upsilon_1 = 0$. By using Lagrange multipliers, we then maximize the quantity:

$$\upsilon_2^T\Sigma\upsilon_2 - \lambda(\upsilon_2^T\upsilon_2 - 1) - \vartheta(\upsilon_2^T\upsilon_1 - 0) \tag{71}$$

By differentiating the above equation with respect to υ_2, and setting it equal to zero, it becomes

$$\Sigma\upsilon_2 - \lambda\upsilon_2 - \vartheta\upsilon_1 = 0 \tag{72}$$

After multiplying Equation (72) by υ_1^T, we obtain

$$\upsilon_1^T\Sigma\upsilon_2 - \lambda\upsilon_1^T\upsilon_2 - \vartheta\upsilon_1^T\upsilon_1 = 0 \tag{73}$$

The first two terms of the above equation are equal to zero from Equation (70), and the last term is $\upsilon_1^T\upsilon_1 = 1$, making the value of ϑ zero. We then conclude from Equation (72) that

$$\begin{aligned}\Sigma\upsilon_2 - \lambda\upsilon_2 &= 0\\ (\Sigma - \lambda\mathbf{I}_q)\upsilon_2 &= 0\end{aligned} \tag{74}$$

The above equation is the same as Equation (67). λ is the eigenvalue of the covariance matrix and υ_2 is the corresponding eigenvector.

Therefore, to maximize $\upsilon_2^T\Sigma\upsilon_2 = \upsilon_2^T\lambda\mathbf{I}_q\upsilon_2 = \lambda\upsilon_2^T\upsilon_2 = \lambda$, λ should be as large as possible but not *equal* to λ_1. This is the second largest eigenvalue, λ_2. If λ_2 equals λ_1 then $\upsilon_1 = \upsilon_2$, making the covariance $\upsilon_2^T\upsilon_1$ not equal to zero. We can conclude that the second PC can be computed by finding the second largest eigenvalue, and using the corresponding eigenvector as our vector υ_2. Generally speaking, the k^{th} PC is $\upsilon_k^T\mathbf{x}$, where υ_k^T is the eigenvector of the covariance matrix that corresponds the k^{th} largest eigenvalue, λ_k.

Thus far, we have described principal components and explained how to obtain them. In the next section, we will describe outlier detection using principal components. After explaining outlier identification, we will relate outlier detection to novelty detection.

Outlier Detection using principal components

We will first define what we mean by outliers in a data set. We will demonstrate with a two-dimensional example. Outliers are generally defined as observations that are inconsistent with the remainder of the data. We would like to detect these outliers. Figure 56 illustrates a data set with outliers.

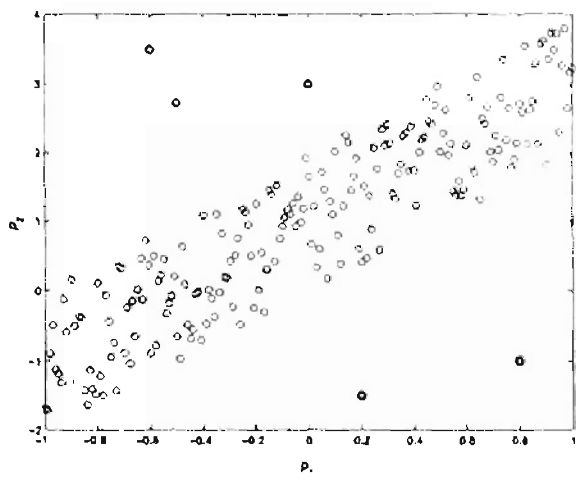

Figure 56 A Data Set with Outliers

We can see that there are five observations that do not follow the majority's underlying correlation. Notice that the outliers are located far away from the densely-populated area. However, no matter which axis, p_1 or p_2, we consider, these outliers fall within the points, thereby making it difficult to detect them.

Now, we will use principal components to detect such observations. After we compute the covariance of the data set, find the eigenvalues and the corresponding eigenvectors. We will then transform the data set in Figure 56 to the principal components, as shown in Figure 57.

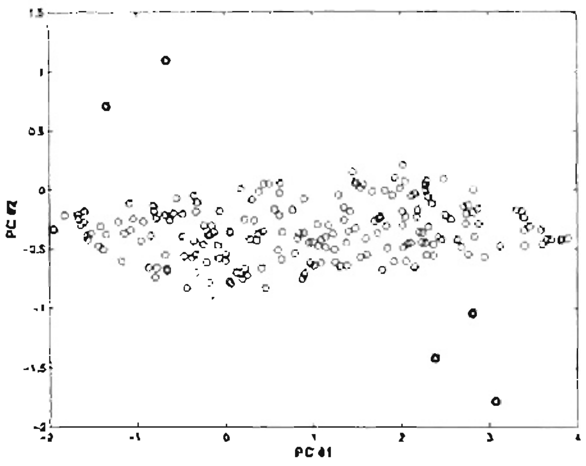

Figure 57 A Transformed Data Set

Because the principal component analysis has decoupled the variables, we can now detect those five outliers by considering the second PC. There are several ways to detect outliers employing principal components, for example [GnKe72], [Hawk74], or [Hawk80]. However, we will use the method proposed in [Rao64] for our outlier detection. [Rao64] suggested that the sum square of the last ν PCs can detect outliers:

$$\xi_j = \sum_{k = q - \nu + 1}^{q} z_{jk}^2 \tag{75}$$

where z_{jk} is the k^{th} PC of the j^{th} observation. The value of ν can be found by experiment. We have found that one value will be acceptable for different data sets. If the value ξ of any data is high, we will pinpoint that data as an outlier. As shown in Figure 57, five observations have a ξ value (with $\nu = 1$), greater than any other data points.

We have described the idea of an outlier and have introduced a technique to detect outliers using principal components. In the next section, we will describe a problem that occurs if we use only outlier identification to solve novelty detection.

The problem of outlier detection for novelty detection

One might think that outlier detection could be used as a method for novelty detection. However, there is one problem. The training data used for function approximator can be more scattered than testing data. Therefore, if we utilize outlier detection, the ξ value for training data are sometimes higher than that for testing data, thereby making the algorithm useless for novelty detection. Figure 58 shows the value of ξ for the last PC of the 638-training data points (shown in Figure 19) and Figure 59 shows the ξ value for the 437-testing data points (shown in Figure 22).

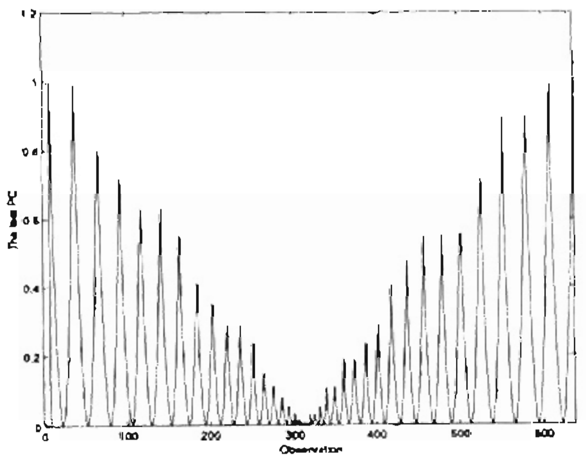

Figure 58 The value of ξ for the last PC of the training data

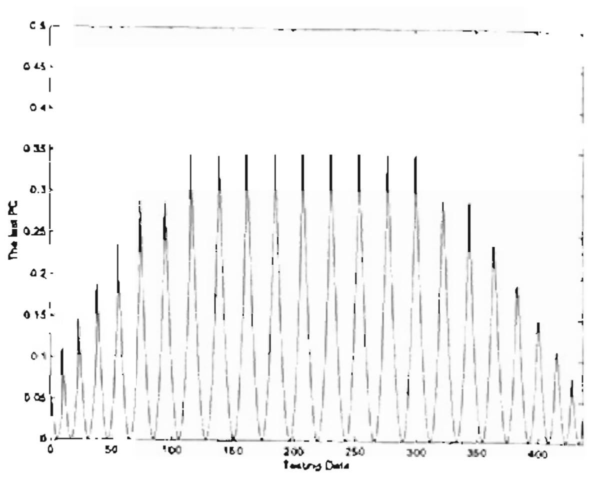

Figure 59 The value ξ for the last PC of the testing data

From the above figures, the value ξ for the training data can be much higher than for the testing data, although the errors for the training data are much smaller than that for the testing data. Note that this result is also true for the composite data set we will describe in the next section.

Even though principal components could not be directly applied to novelty detection, we found that we can use it as a *constraint* for the minimum distance algorithm. We will describe this modified algorithm in the next section.

Minimum distance of the composite data set

Recall from Chapter 6 that we developed a novelty detector employing the minimum distance algorithm. We wanted to improve the performance of the novelty detector (reducing the percentage of misclassifications) by incorporating the effect of the network outputs and targets. We began from the error equation, and finally obtained the minimum weighted distance. However, one drawback of the minimum weighted distance is that the weighting factor α producing the highest performance varies from data set to data set. That means that the α giving the lowest percentage of misclassifications in one data set may perform poorly on another data set. Therefore, the purpose of this section is to use the minimum distance without the weighting factor.

In Chapter 4 and Chapter 5, we found that we can include the effect of network outputs by adding them to the input vectors. In other words, for any training vector, the composite vector for the j^{th} observation will be $\Gamma_j = \left[\mathbf{p}_j \; t_j\right]^T$. And for any testing vector, we use the network output in place of the target. The augmented vector for the k^{th} observation will be $\Gamma_k^t = \left[\mathbf{p}_k^t \; a_k^t\right]^T$. This means that we need to propagate our testing input $\mathbf{p}_k^t$ through the function approximator to get a_k^t before creating the augmented testing vector. Then, we will use these composite vectors to compute the minimum distance for each new testing vector Γ_k^t. Therefore, if a testing input is very close to one of the training inputs but produces an eccentric output, the minimum distance for the composite testing vector will be larger.

We can use minimum distance calculations on the augmented data vectors to perform novelty detection. However, we found that we can improve the efficiency of this method by combining it with the method of outlier identification. The following section will illustrate how outlier detection can improve the performance of the novelty detection.

Application to novelty detection

Recall that the objective of this thesis is to identify data on which the function approximator may perform poorly. The percentage of misclassification is the parameter we will use to compare the performance of each novelty detector. The goal of this chapter is to combine the minimum distance algorithm for the composite data with outlier detection. The minimum distance algorithm is already described in Chapter 6. Furthermore, we also explained in the last section how we obtain the composite data set for training and testing data that will be used to compute the minimum distance. In this section, we will explain how the outlier detection can reduce the percentage of misclassified points when using the minimum distance algorithm on the composite data.

We will start this section by computing the minimum distance of the augmented testing data for the simple example that we used in previous chapters. Figure 60 is a scatter plot relating the minimum distance between the 638-training vectors and the 437-testing vectors to the error from the function approximation.

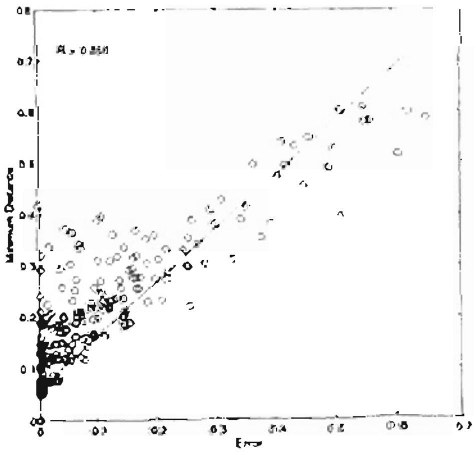

Figure 60 Approximation Error and minimum distance

We can see from Figure 60 that the correlation coefficient R between the network error and the minimum distance is higher than the R value in the previous chapter (compare with Figure 49). However, we can see that there are also some large-error points having the same minimum distance as small-error points.

As in previous chapters, we will use the regression line to determine the threshold for novelty detection. In this case, the regression line is

$$d_m = 1.04 \times error + 0.0598 \tag{76}$$

The threshold required to reject errors greater than 0.15 is 0.2158. If we reject data with minimum distance greater than 0.2158, the percentage of misclassified points is 8.70%. Most of the misclassifications (around 8.47% out of 8.70%) were from type I errors.

Although the percentage of misclassifications using this method (computing the minimum distance for the augmented data set) was acceptable, we found that the percentage of misclassified points could be lower if we utilize outlier detection. Figure 61 shows the sum-square values of the last two PC (ξ, using $\nu = 2$ in Equation (75)), versus the minimum distances. The figure also indicates points where the network errors are less than 0.15. (We will explain why we chose $\nu = 2$ in the following example.)

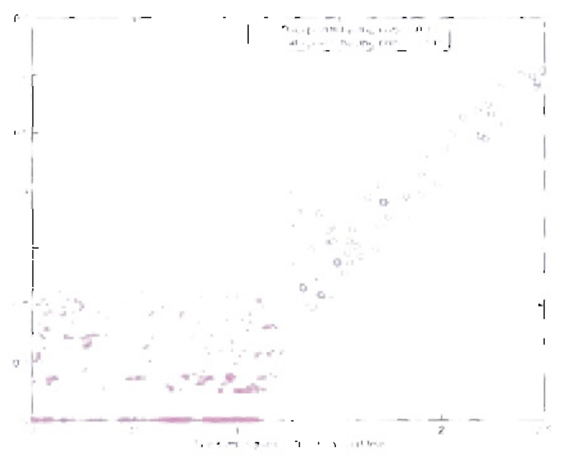

Figure 61 ξ and the minimum distance

From the above figure, we can see that data with small minimum distance (less than 0.2) have small errors. When the minimum distance is between 0.2 and 0.3, the small-error data were more likely to have ξ less than one. On the other hand, the large-error points tended to fall where the sum-square value is large. However, when the minimum distance was greater than 0.3, the ξ cannot distinguish between large and small-error points. This result is understandable because the larger the minimum distance, the more likely large errors can be found, since the testing point is far from a small-error data point (training data). Besides, at a *certain* minimum distance, the larger ξ implies the data point tends to be far away from the densely-populated area occupied by the training data. Therefore, we would like to accept those data having small minimum distance (close to training data) and small ξ value (lying in the highly-populated area). On the other hand, we will discard any vector having very large minimum distance *or* any data creating somewhat large minimum distance but very large ξ (lying in regions far from the densely-populated area of the training data).

We decided that the rejection region should be $T_1^d < d_m \le T_2^d$ and $\xi > T^\xi$ or $d_m > T_2^d$. Recall from Equation (76) that we selected the threshold based on the regression line. The notation T^d denotes such threshold. Then, T_1^d is the minimum distance around 75% of T^d, while T_2^d is the minimum distance around 125% of T^d. In other words,

$$T_1^d = 0.75T^d, \ \ T_2^d = 1.25T^d \tag{77}$$

T^ξ is the average value of ξ between the small-error points and the large-error points within the range of $T_1^d < d_m \le T_2^d$. In other words,

$$T^\xi = \frac{T_S^\xi + T_L^\xi}{2} \tag{78}$$

where T_S^ξ is the average ξ value of the small-error points between the range $T_1^d < d_m \le T_2^d$. Similarly, T_L^ξ is the average ξ value for the large-error points within the range of $T_1^d < d_m \le T_2^d$. Note again that the small-error point is defined as the vector generating approximation error less than 0.15.

With the threshold we created, it means that any data generating minimum distance less than T_1^d, and also any data point making minimum distance within the range $(T_1^d, T_2^d]$ with ξ less than T^ξ, will be accepted. From the rejection region we described above, for any composite training data, although they can have very high ξ (greater than T^{ξ}), their minimum distances are small so they are not detected as novel data.

In this section, we explained how to incorporate the method of outlier identification with the minimum distance algorithm. The threshold for discarding abnormalities was introduced as well. We will demonstrate the ability of this algorithm in the following example.

Simulation of the simple example

This section will illustrate the ability of the minimum distance algorithm with outlier detection constraints. First, we will transform the three-dimensional training data to principal components. The unbiased estimate for the covariance matrix is

$$\Sigma = \begin{bmatrix} 0.5182 & -0.0001 & -0.0008 \\ -0.0001 & 0.1824 & -0.0012 \\ -0.0008 & -0.0012 & 0.1020 \end{bmatrix} \tag{79}$$

The eigenvalues and eigenvectors of the covariance matrix are

$$\begin{aligned} \lambda_1 &= 0.5182, \upsilon_1^T = \begin{bmatrix} 1.0000 & -0.0003 & -0.0019 \end{bmatrix} \\ \lambda_2 &= 0.1824, \upsilon_2^T = \begin{bmatrix} 0.0003 & 0.9999 & -0.0149 \end{bmatrix} \\ \lambda_3 &= 0.1020, \upsilon_3^T = \begin{bmatrix} -0.0019 & -0.0149 & -0.9999 \end{bmatrix} \end{aligned} \tag{80}$$

We decide to use the last two PCs to compute ξ (i.e. set $\nu = 2$ in Equation (75)) since it has the smallest percentage of misclassified points. However, we found that this value worked best for different data sets as well. After setting the ν value, υ_2^T and υ_3^T shown in Equation (80) will be used to transform the training data to the second and third

PCs. We calculate $z_{i2} = \upsilon_2^T \Gamma_i$ and $z_{i3} = \upsilon_3^T \Gamma_i$ for all i in the training data. We are now ready to compute ξ for the i^{th} training data:

$$\xi_i = z_{i3}^2 + z_{i2}^2 \qquad (81)$$

Figure 62 plots ξ for the composite training data.

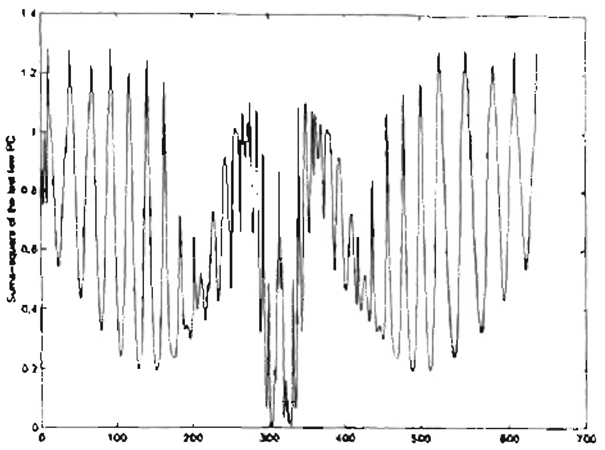

Figure 62 The value ξ from the composite training data set

For the testing data, we first need to propagate the input vector $\mathbf{p}_j'$ through the function approximator to get the network output a_j'. Then, we compute the second PC $z_{j2}' = \upsilon_2^T \Gamma_j'$ and the third PC $z_{j3}' = \upsilon_3^T \Gamma_j'$ to obtain

$$\xi_j' = (z_{j3}')^2 + (z_{j2}')^2 \qquad (82)$$

Figure 63 illustrates the value ξ_i' for the 437 testing data points.

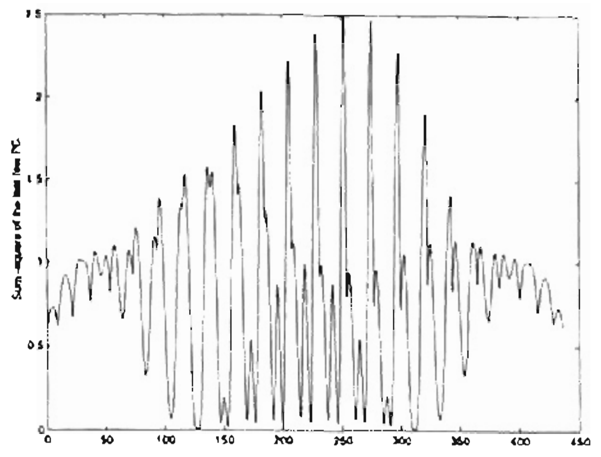

Figure 63 The value ξ' from the composite testing data set

We can see from the figures that ξ' values for the testing data were higher than ξ value for some training data. This phenomenon does not generally happen, especially when the network outputs for the testing data are bounded within the normalization range, which is $[-1, 1]$. The data we use in Chapter 8 is another example showing that although the sum-square value ξ' of augmented testing data could be lower than that of composite training data, the testing data created large errors. As we explained, this is the reason why we cannot use only outlier detection for novelty detection. The next procedure is to find the minimum distance of the augmented testing vectors. The minimum distance for the 437 testing vectors is shown in Figure 60.

After calculating both the minimum distance and ξ for the composite testing data, we plotted the values in Figure 61. Based on this figure and Equation (76), we found that the minimum distance T^d was 0.2158. Then, $T_1^d = 0.75T^d = 0.1618$ and

$T_2^d = 1.25T^d = 0.2697$. Within the range of $(0.1618, 0.2697]$, we found that the average ξ value for small-error points is 0.6702, while the average ξ value for large-error points is 1.3414. T^ξ is then equal to $\frac{0.6702 + 1.3414}{2} = 1.0058$. (Notice that within the same range of minimum distance, small-error points have on-average smaller sum-square value than large-error data. This can help us distinguish small-error points and large error points within the range of minimum distance.)

From the values we obtained, it means that any data generating minimum distance less than 0.1618 will be accepted. In addition, we will accept any data having minimum distance in the range $(0.1618, 0.2697]$, as long as ξ is less than 1.0058. In other words, the rejection region is

$$(0.1618 < d_m \leq 0.2697 \text{ and } \xi' > 1.0058) \text{ or } (d_m > 0.2697) \qquad (83)$$

Equation (83) means that any data generating minimum distance within the range $(0.1618, 0.2697]$ with ξ greater than 1.0058, and any data with minimum distance larger than 0.2697 will be considered novel. Figure 64 demonstrates the network errors, and novel data identified by this algorithm are flagged with $\mathbf{x}$.

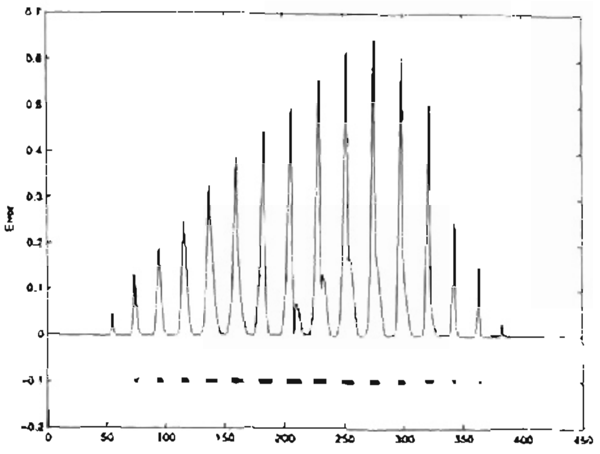

Figure 64 Novelty Detection

The percentage of misclassifications for this algorithm is 8.24%. All of the misclassifications were from the small-error points in the rejection region (type I errors).

There exists another rejection region that makes fewer misclassifications than the number we showed. It will depend on the application as to what types of misclassifications we can tolerate more. For example, we can increase the thresholds T_1^d, T_2^d or T^ε to obtain less type I error.

From the simulation outcomes, we can see that the percentage of misclassifications is somewhat low. Also, the execution time for this method was acceptable. Although this algorithm seems to have very promising results, the main drawback is a complicated threshold for discarding abnormalities.

Summary

The purpose of this chapter was to use a minimum distance algorithm that includes network outputs but does not include a weighting factor. We saw in the last chapter that the weighting factor can vary from one data set to another. We found that a principal component method for outlier detection could improve the efficiency of the minimum distance algorithm. We found that, at a certain value of minimum distance, the principal component statistic could differentiate between large and small errors. This phenomenon could reduce the percentage of misclassified data. The simulation results showed that the percentage of misclassified points for the new algorithm was acceptable. The computation time for this algorithm is comparable to the minimum distance algorithm.

The new algorithm has one parameter that must be set. Although we originally set it heuristically, it turned out to work for many different data sets. One drawback of the method is that it requires a somewhat complicated thresholding mechanism.

CHAPTER 8

SIMULATION RESULTS

Introduction

We will dedicate this chapter to demonstrating the performance of the various novelty-detection algorithms we described in the previous chapters on two real world applications. First, we will describe the data sets (e.g. training data, and two sets of testing data). After that, we will discuss the outcomes from the following methods:

1. Neural tree
2. Gaussian kernel estimator and joint density
3. Minimum distance and the minimum weighted computation
4. Minimum distance with outlier detection using principal components analysis

We chose all the methods we discussed in this thesis except the autoassociative multilayer perceptrons. The time required for training the autoassociative perceptrons was too long for our high dimensional data.

After showing the simulation results for these methods, we will summarize the performance of each algorithm in terms of the percentage of misclassified points, including the two types of misclassifications made by the novelty detectors.

Function approximation I

In this section, we will briefly describe the function we are going to estimate, and some details of training data and testing data for the first data set.

Formation resistivity is a key parameter for estimating the presence of oil or gas. Signal transmitters (coils) energized by alternating current (AC) at frequencies ranging from 8-40 kHz create an oscillating magnetic field. This magnetic signal causes induced currents, which are approximately proportional to the conductivity (reciprocal to resistivity), of the earth formation. The currents then contribute to induced voltages at receiver coils. We will process the received voltages to obtain the values of formation resistivity. Generally speaking, the activated voltage is a non-linear function of the formation resistivity.

Since the parameter we are interested in is the formation resistivity, we use a neural network to model the inverse of the non-linear function. In other words, neural networks (function approximators) are used to convert the received voltage to the formation resistivity. That means that the *input* to the function approximator is the voltage, and the *output* of the function approximator is the formation resistivity. We have several neural networks for different receiver coils, but we will discuss only one coil, which is Coil_1c.

At Coil_1c, we collected 19,558 data points, which consisted of the induced voltages (inputs) and the resistivity values (targets). We used most of them (13,175 points) to train our neural network. The rest of the collected data points were used as the validation data set during the training process. This error from the validation data set was monitored during training so that we could stop training the neural network if the error from this data set increased.

For Coil_1c, the structure of the neural network was $51 - 20 - 10 - 1$, meaning that we used the induced voltages from 51 different depths to predict the resistivity at one central depth. After we completed training, the function approximator could correctly estimate the resistivity for training data and validation data. The maximum error for the network output was 0.232. (Note that this is an error on data that has been normalized to the range $[-1, 1]$.) There were 0.29% of the total training data that generated errors larger than 0.15.

We also have nine testing sets, each containing 581 data points. However, we will demonstrate the results of only six test sets (set 01, set 02, set 06, set 08, set 10, and set 11). We will divide these testing data into two groups. The first test group consists of set 01, set 06, and set 10, while the remaining sets are placed in the second test group. For the first test group, we will assume that we know the error from the function approximator; however, the errors from the second test group are presumed unknown. We will find the threshold to reject abnormalities from the first test group, and will use the second test group to compute the percentage of misclassifications based on the threshold. The threshold we create should apply to arbitrary data sets. We will use the percentage of misclassifications to compare the performance of the various novelty detectors. Figure 65 shows some example plots of voltage and resistivity utilized for training our neural network.

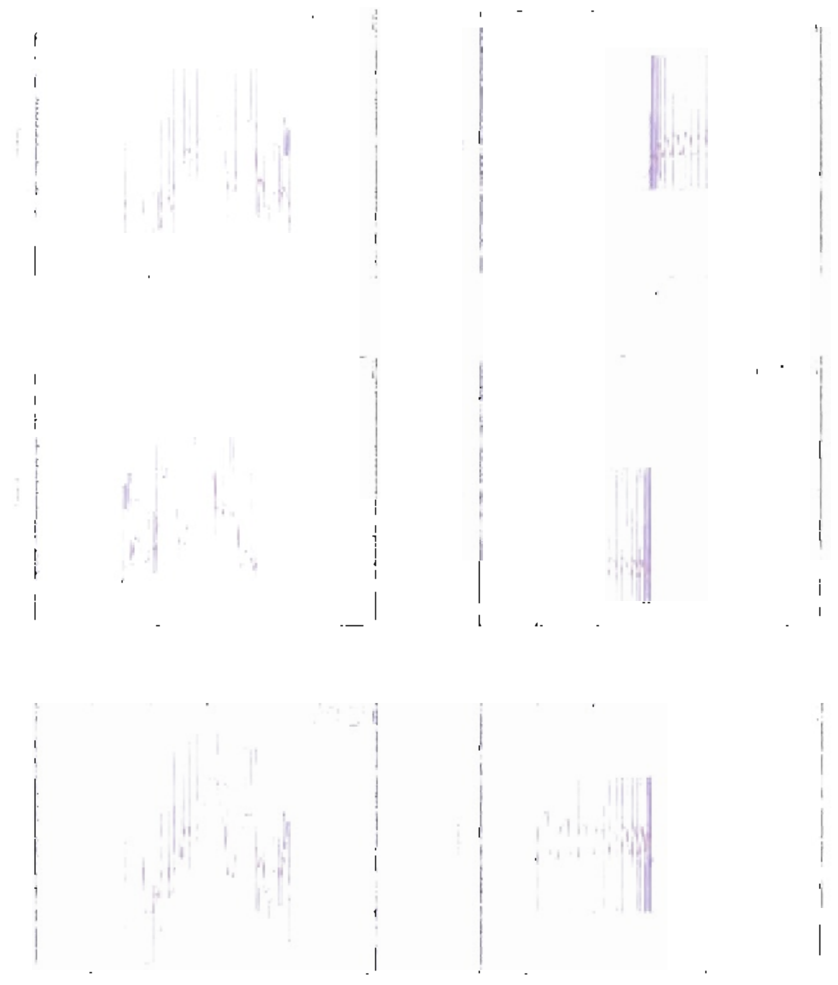

Figure 65. Example plots of voltage and resistivity: Training data

The plots demonstrating voltage and resistivity used for testing the neural network are shown in Figure 66.

Figure 66. Voltage and resistivity testing data

In the following sections, we will apply our testing vectors to the function approximator to determine the accuracy of the resistivity estimates. Then we will test the ability of the novelty detectors to identify poor estimate

Neural tree

We will first use the neural tree algorithm for novelty detector. Refer to Chapter for the details of the neural tree algorithm and how to use it for novelty detection

First, we need to specify the number of cells for the tree. A 5,000-cell tree was initialized to partition the fifty-one dimensional space. Then we applied the 13,175 training vectors, using the same learning rate we used in Chapter 3. After limiting the size of every cell with infinite volume, we applied our testing data to the trained tree to find abnormalities. The algorithm to define novel data was the same as the example in Chapter 3 (data are identified as novel if they are outside of the cell boundaries). Although we do not need to separate testing data into two groups for this algorithm, since the threshold to identify abnormalities is defined by the algorithm, we will show the number of misclassifications of the two test sets separately. Figure 67 illustrates the error from the function approximator for the first test set, and the abnormalities identified by the 5,000-cell tree.

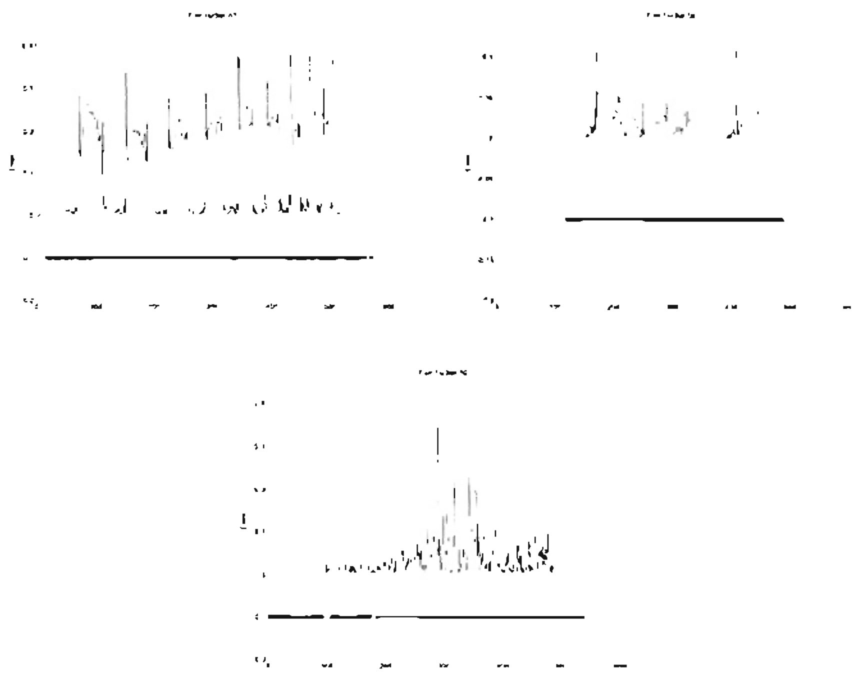

Figure 67 Error and abnormalities: The first test group (Neural Tree)

There are two types of misclassifications made by a novelty detector. For type I misclassification, data points generating small approximation error are flagged as novel data. For type II misclassification, data points generating large approximation error are flagged as normal. (Note again that approximation errors greater than 0.15 will be considered large error for this thesis.)

Recall that the neural tree algorithm flags data as novel when they are out of the cell boundaries. From Figure 67, we can see that most of the data points were identified as novel data for the function approximator. The percentage of misclassifications was 67.87%. All of the misclassifications were from small-error points that were flagged as novel data (type I error). This number is extremely high (similar to the simple example shown in Chapter 3). This may be due to the fact that flagged data are close to the training inputs but they are out of the cell boundaries. We found that around 93.8% of the 5,000 cells are limited by the training data, compared with 31% of the 200 cells in the example in Chapter 3. With a very high percentage of limited cells, testing data close to training data (which are used to limit cells) are more likely to be out of the cell boundaries. We expect that the result for the second group will not be much different.

Next, we applied the second test group to the tree. Figure 68 shows the network errors for the second test group and the abnormalities.

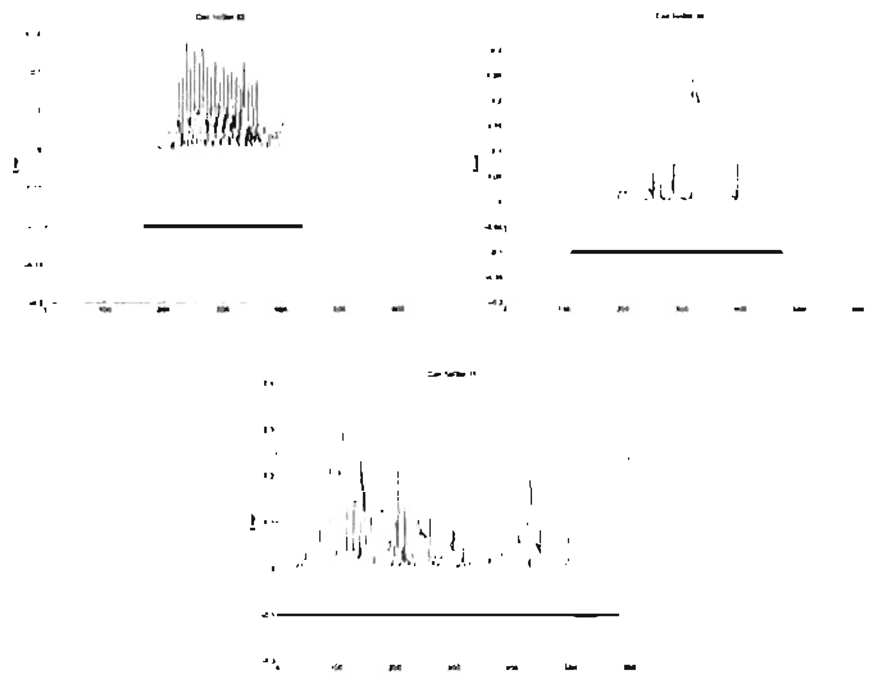

Figure 68 Error and abnormalities: The second test group (Neural Tree)

We can see from Figure 68 that most data were marked as novel points. The percentage of misclassifications in this case was 67.01%, which is about the same as the first test set. All of the misclassifications also were from type I error. This result is similar to what we expected from the first group.

When we applied the training data to the novelty detector, we found that the percentage of misclassifications was equal to 0.29%. All of the misclassifications were from the large-error points flagged as normal data (type II error). This is because all of the training data were inside the 5,000 cells.

In this section, we used the neural tree algorithm to identify abnormalities for the function approximator. In the next section, the Gaussian kernel estimator will be utilized as our novelty detector for the function approximator.

The Gaussian kernel estimator (GKE)

In this section, we will use the Gaussian kernel estimator from Chapter 4 to estimate the density of inputs to the function approximator, and to estimate the joint density between the inputs and the target outputs. The estimated density will be used to identify novel data for the function approximator.

The estimated density of input

We will use the Gaussian kernel estimator to estimate the density function of the input to the function approximator. If the estimated density is low for a new input, we will reject that input as novel. The details of this algorithm were explained in Chapter 4.

For the smoothing parameter matrix, we will use the same approach as we used in Chapter 4 – the average distance to the ten-nearest neighbors. We found that the smoothing parameter matrix in this case will be:

$$\Sigma_p = \begin{bmatrix} 0.0698^2 & 0 & \dots & 0 \\ 0 & 0.0698^2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0.0698^2 \end{bmatrix} \tag{84}$$

The matrix in Equation (84) is 51×51 since the dimension of the input to the function approximator is 51. After using Equation (33), we obtained the estimated density for the first test set. Figure 69 plots the estimated density versus the error from the function approximator for the first test group.

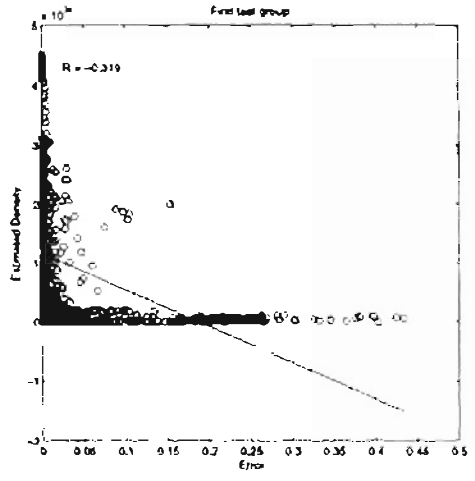

Figure 69 Estimated density and approximation error: The first test group

To create the threshold for discarding novel data, we consider the regression line shown in Figure 69. The equation for that line is

$$density = -6.1756 \times 10^{36} \times error + 1.1663 \times 10^{36} \tag{85}$$

From Equation (85), the density that corresponds to an error of 0.15 is 2.3993×10^{35}. This means that, on average for the first test group, data points having a density of 2.3993×10^{35} generate an error for the function approximator of 0.15. Therefore, we will use this value as the threshold to reject novel data. In other words, any data generating an estimated density less than the threshold will be considered as novel. The errors from the function approximator and the novelties for the first test group identified by the Gaussian kernel estimator are shown in Figure 70.

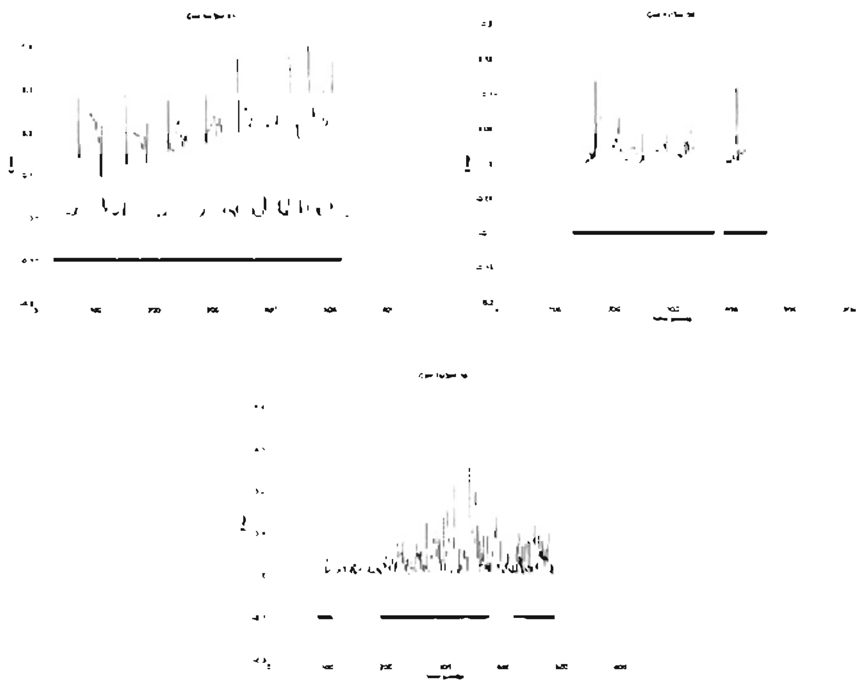

Figure 70 Error and abnormalities: The first test group (GKE: input)

We found that the percentage of misclassifications was 62.25%. About 61.79% out of 62.25% were from type I errors.

Next, the second test group will be applied to the novelty detector with the specified threshold in order to test whether the novelty detector can identify abnormalities from another testing region. Figure 71 illustrates the error from the function approximator, and the data points detected as novel data for the second test group.

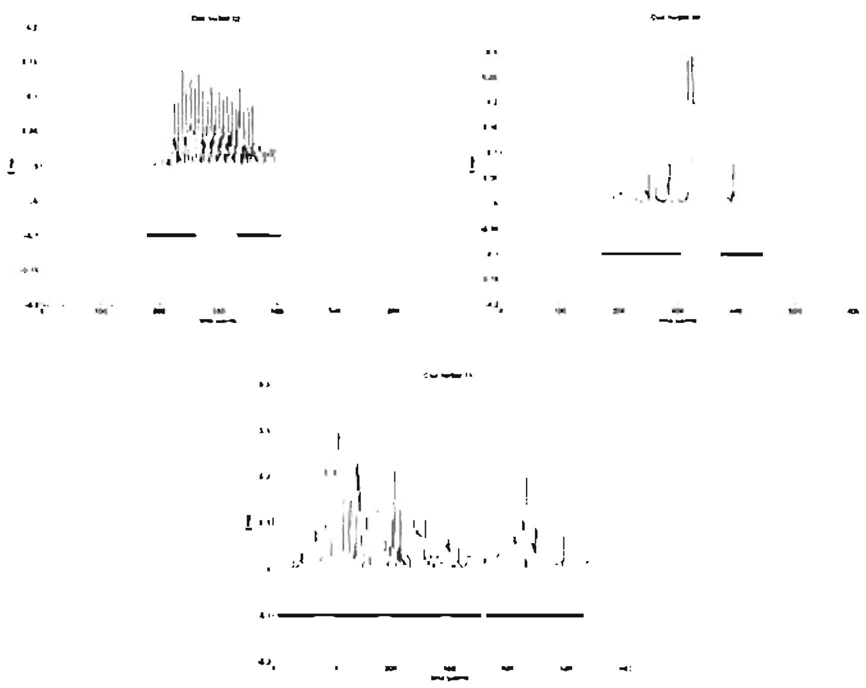

Figure 71 Error and abnormalities: The second test group (GKE: input)

We can see that the algorithm not only identified abnormalities, but small-error points as well. The percentage of misclassifications in this second test set was 51.58%. The majority of these misclassified points (around 50.03% out of 51.58%) were type I errors.

When the training data were applied to the novelty detector using the Gaussian kernel estimator with the threshold we described above, the percentage of misclassified points was equal to 34.73%. Most of the misclassifications (around 34.64% out of 34.73%) were from type I errors.

The misclassification percentages for the two test groups are fairly high but less than that from the neural tree algorithm. However, the percentage of misclassifications was very high in the training data set. This might be due to the fact that the smoothing parameter matrix is not well-fitted and the threshold we set up from the first test group is too high for the training data. We already discussed in Chapter 4 that this method tends to have many type I misclassification.

In this section, we used the Gaussian kernel estimator as the novelty detector for the function approximator. We found that the percentage of misclassifications was reduced when compared with the neural tree algorithm. In the next section, we will incorporate the effect of the network output to compute the estimated joint density.

The estimated input-output density

We assume in this case that our smoothing parameter matrix is the identity matrix. Each element of the input is uncorrelated. The diagonal elements are calculated by using the average distance to the ten nearest neighbors. The smoothing parameter (52×52) was

$$\Sigma_\Gamma = \begin{bmatrix} 0.0771^2 & 0 & \dots & 0 \\ 0 & 0.0771^2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0.0771^2 \end{bmatrix} \tag{86}$$

The following figures demonstrate the simulation results by showing the relationship between the error from the function approximator and the estimated density.

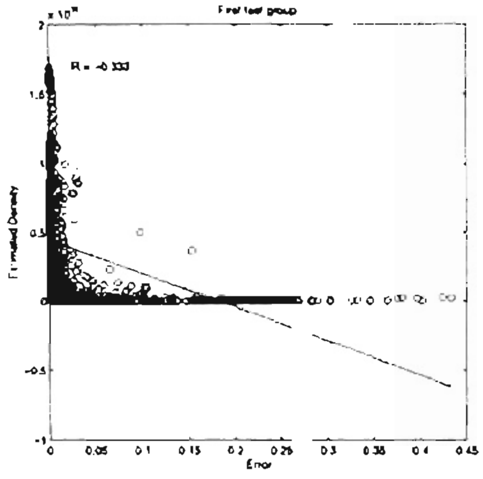

Figure 72 Estimated density and approximation error: The first test group

To reject novel data, the regression line will be used. The regression line in this case was

$$density = -2.4606 \times 10^{35} \times error + 4.4895 \times 10^{34} \qquad (87)$$

After substituting 0.15 into the error term in Equation (87), the corresponding density is 7.9865×10^{33}. That means that, based on the first test group, data points generating an error of 0.15 have estimated joint density around 7.9865×10^{33}. Therefore, 7.9865×10^{33} will be used as the threshold to reject abnormal data. Therefore, any composite data generating joint density lower than this threshold will be considered novel. Figure 73 illustrates the network errors and novel data are flagged with an **x**.

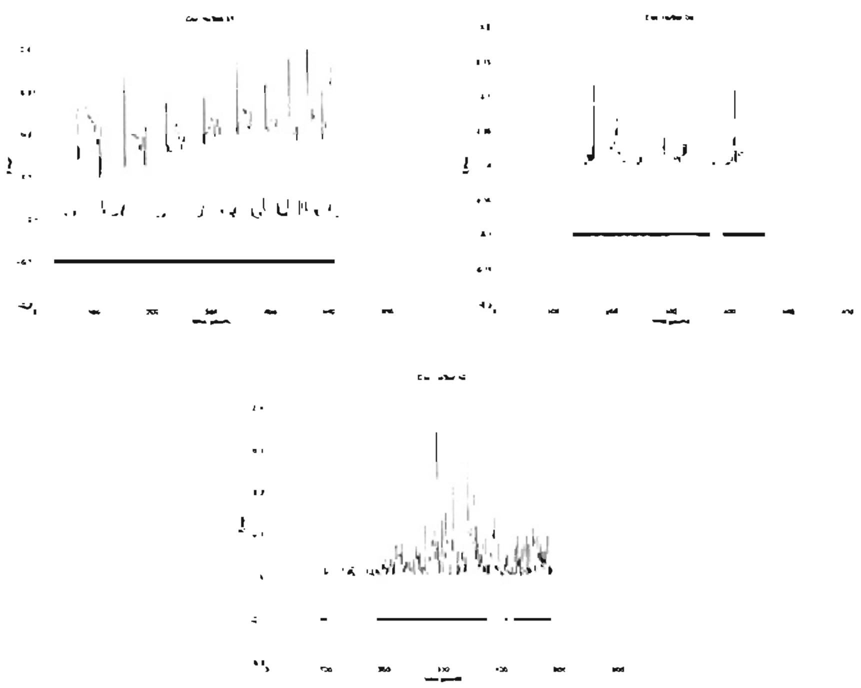

Figure 73 Error and abnormalities: The first test group (GKE: input and output)

The percentage of misclassifications in this case was 60.01%. The majority of the misclassifications (around 59.44% out of 60.01%) were type I errors, and the rest of the misclassifications was type II errors.

The second test group is now applied to the novelty detector with the specified threshold. Figure 74 illustrates the network errors and flags novel data.

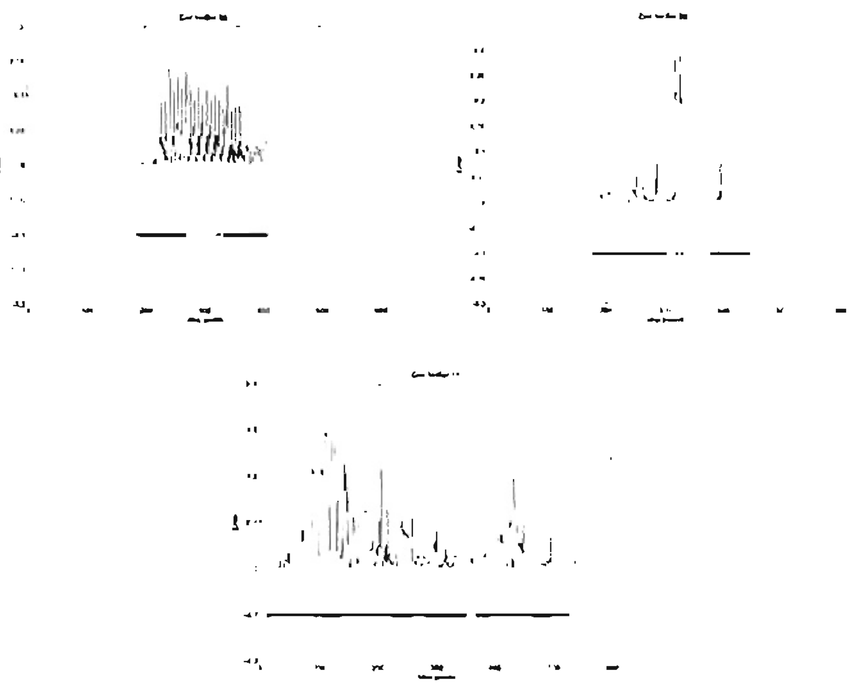

Figure 74 Error and abnormalities: The second test group (GKE: input and output)

The percentage of misclassified points for the second test group was 50.14%. Around 49.22% out of 50.14% were from type I errors.

If we applied the training data to the novelty detector, the percentage of misclassifications was equal to 35.39%. We found that around 35.31% out of 35.39% were the type I errors. The rest of the misclassified data points were type II error.

We can see from the results that the joint density (between input and output) provided lower misclassifications for both the first and second test groups. However, for the training data, the percentage of misclassified points using the joint density (between inputs and output) is larger than the percentage of misclassifications using the density of inputs. That means that some training data have very low density compared with the joint density of some novel data. These low-joint-density points in the training data are the points in regions where the density of target outputs and inputs are low. We also experienced this phenomenon when illustrating the simulation results in Chapter 4 (i.e. simulation #2)

In this section, we showed the simulation results for novelty detection using the Gaussian kernel estimator method to estimate the joint density. The results illustrated that the joint density between inputs and outputs tended to distinguish novel data clearer than the estimated density of the input alone. In other words, the percentage of misclassifications for the joint density is lower than that for the estimated density of inputs. In the next section, we will test the minimum distance algorithm for novelty detection.

Minimum distance algorithm

This section will illustrate the ability of minimum distance and minimum weighted distance as the novelty detectors for the function approximator. Refer to Chapter 6 for the details of these two methods.

The minimum distance of input

The minimum distance algorithm for novelty detection will be shown in this section. We first compute the minimum distance from the testing inputs to the training inputs. Figure 75 presents the correlation between the error from the function approximator and the minimum distance on the first test group.

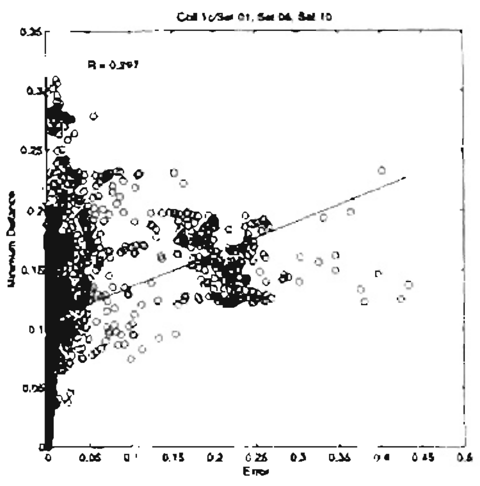

Figure 75 Minimum distance and approximation error: The first test group

From Figure 75, we found that the regression line that represented the relationship between the minimum distances and the errors for the first test group was

$$d_m = 0.276 \times error + 0.108 \qquad (88)$$

After substituting 0.15 in the error term, the corresponding minimum distance is 0.1494. We will use this value as the threshold to identify novel data. Figure 76 illustrates the errors from the function approximator, and novel data are flagged with **x**.

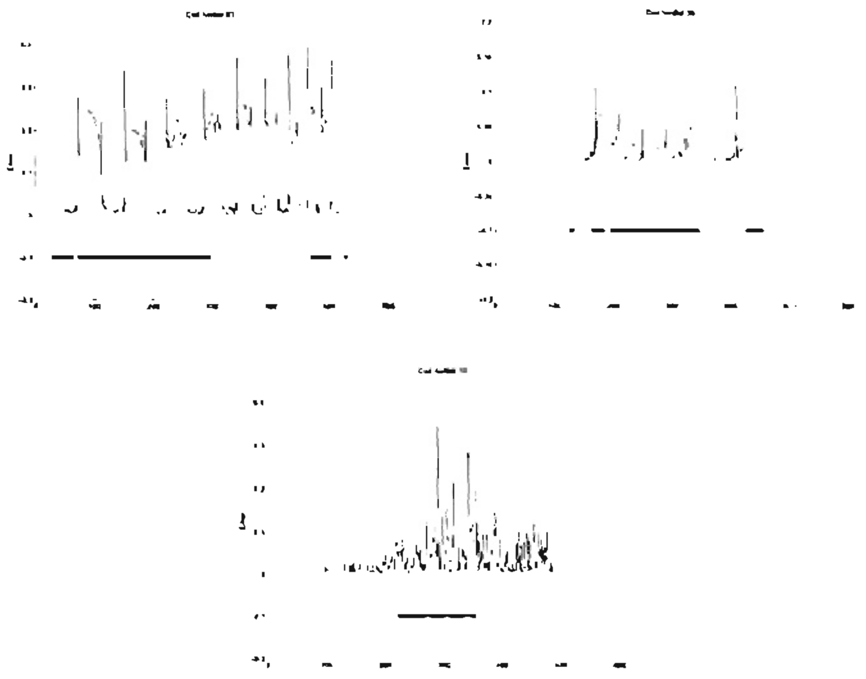

Figure 76 Error and abnormalities: The first test group (Minimum distance)

We can see that there were many points detected as novel data though they generated small errors. The percentage of misclassified points was 34.31%. The majority of the misclassifications (around 28.86%) were from type I errors.

Next, we applied the second test group to the novelty detector. Figure 77 shows the network error and the abnormalities identified by minimum distance with threshold of 0.1494.

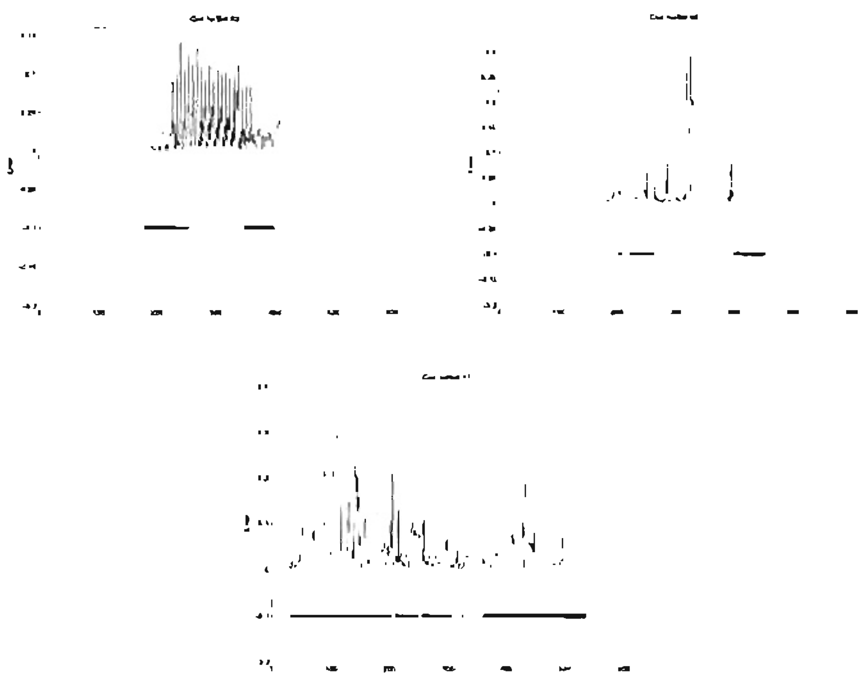

Figure 77 Error and abnormalities: The second test group (Minimum distance)

Most large-error points in this test set were identified. However, we missed many data points with small errors (as in the first test group). The percentage of misclassifications in this data set was about the same (35.23%). Around 34.65% out of 35.23% were from type I errors.

If we applied the training data to the novelty detector using the minimum distance algorithm with the threshold obtained above, the percentage of misclassifications was equal to 0.29%. All of the misclassifications were from type I errors. This is due to the fact that any training data will have zero minimum distance.

In this section, the results of novelty detection using the minimum distance algorithm were demonstrated. The percentage of misclassifications was about 34-35%. In the next section, we will test the minimum weighted distance for novelty detection.

The minimum weighted distance

In this section, we will utilize the minimum weighted distance for novelty detection. Refer to Chapter 6 for the details of this method. We will start this section by demonstrating the effect of the weighting factor on the correlation coefficient for the first test group. In other words, the inputs of the first test group were applied to the function approximator to get the network outputs. After that, the minimum weighted distances in the first test set were calculated at a specific weighting factor. The correlation coefficient between the errors from the function approximator and the minimum weighted distances (for the first test group) are plotted in Figure 78 as we varied the value of the weighting factor.

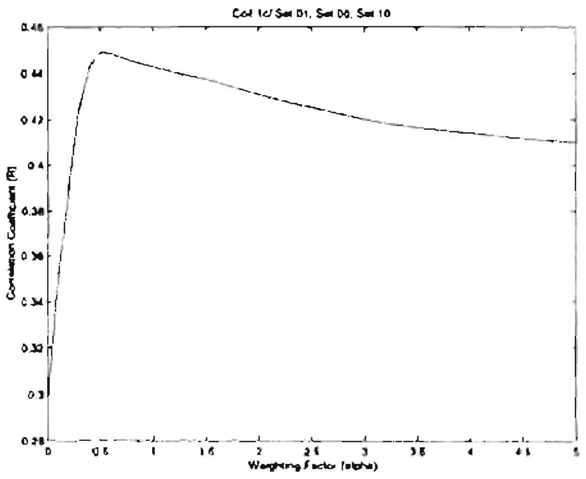

Figure 78 The weighting factor and the correlation coefficient: The first test group

For small values at weighting factor, the correlation coefficient R between error and the minimum weighted distance increases with the increased weighting factor. Furthermore, when the weighting factor is 0.5 the correlation coefficient is highest. Therefore, we decided to use this weighting factor for novelty detection. Figure 79 illustrates the error of the function approximator and the minimum weighted distance of the three testing sets (the first test group) at a weighting factor of 0.5.

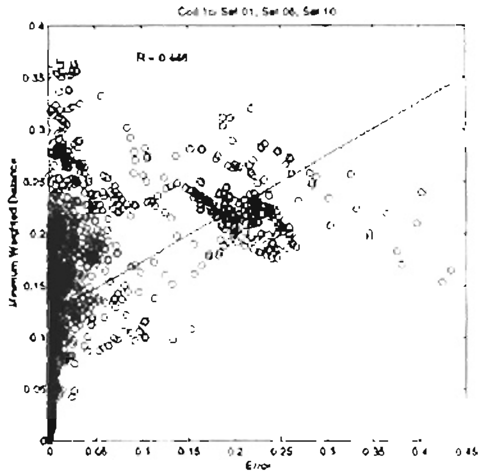

Figure 79 Minimum weighted distance and approximation error: The first test group

From Figure 79, we will use the regression line to create the threshold for discarding novel data. The regression line for the first test group was

$$d_m^{0.5} = 0.51 \times error + 0.122 \tag{89}$$

By substituting 0.15 into the error term, we get the minimum weighted distance 0.1985. That means that, at the weighting factor 0.5, data points generating the error 0.15 have minimum weighted distance 0.1985. Therefore, in order to identify data points with errors larger than 0.15, the minimum weighted distance 0.1985 will be used as the threshold. Figure 80 shows the network errors and the identified abnormalities.

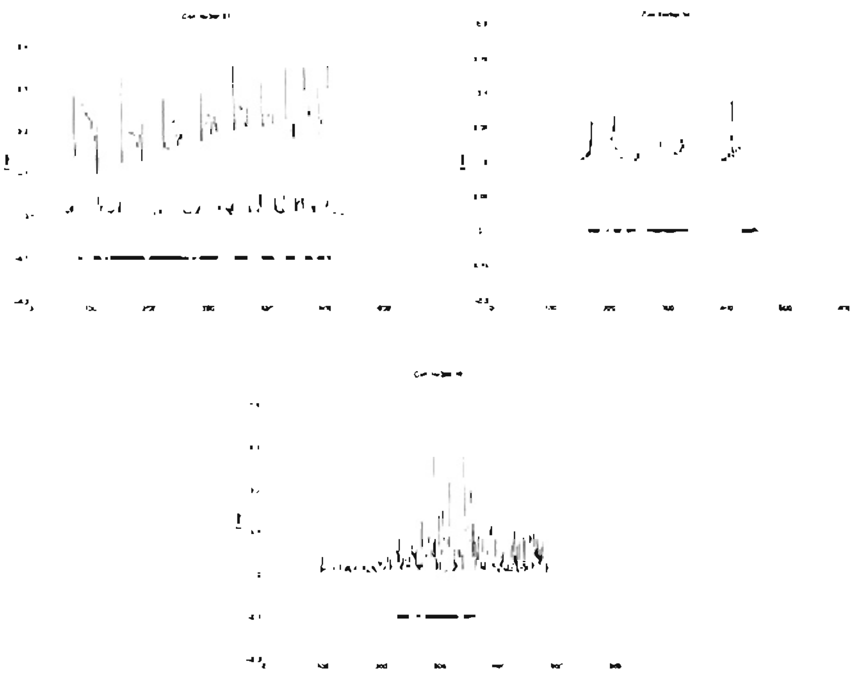

Figure 80 Error and abnormalities: The first test group (Minimum weighted distance)

We found that the percentage of misclassifications was 19.40%. The major misclassifications (17.10% out of 19.40%) were from type I errors. The rest (2.30%) were from type II errors.

Next, we will apply the second test group to the function approximator to get the network outputs. The minimum weighted distance at the weighting factor 0.5 for this data set will be then computed. These minimum weighted distances for this test group will be filtered by using the threshold we created from the first test group. Any data points generating minimum weighted distance larger than the threshold will be discarded as abnormalities. Figure 81 demonstrates the error from the function approximator for the second test group, and the detected abnormalities are flagged with **x** .

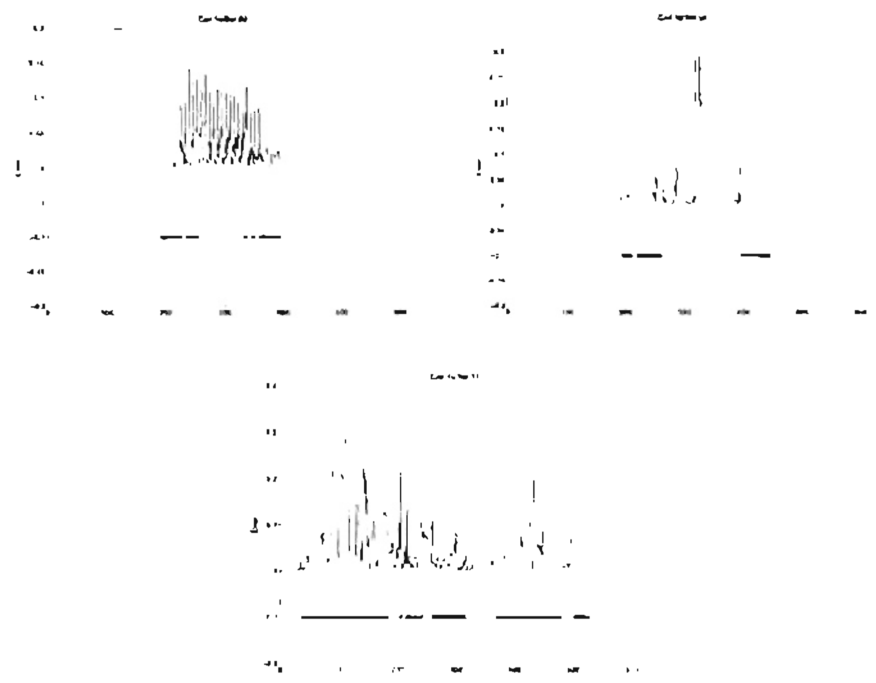

Figure 81 Error and abnormalities: The second test group (Minimum weighted distance)

The percentage of misclassifications in this case was equal to 30.36%. The majority of the misclassified points (around 29.78% out of 30.36%) were from type I errors. We can see that the percentage of misclassifications was reduced by a certain amount for both test sets, compared with the percentage of misclassifications from the minimum distance algorithm.

If we applied the training data (inputs to the function approximator) and the corresponding network outputs to the novelty detector using the minimum weighted distance, the percentage of misclassifications was equal to 0.29%. All of the misclassifications were from the type II errors.

In this section, we used the minimum distance and the minimum weighted distance algorithm for novelty detection. The minimum weighted distance algorithm turned out slightly fewer misclassifications than the minimum distance method. In the next section, we will show the result of novelty detection using minimum distance with outlier identification.

Minimum distance with outlier detection

This section will present the simulation results for the minimum distance with outlier detection algorithm for novelty detection. Chapter 7 explained the details of this method.

First, we need to create composite training data by combining the targets and the inputs to the function approximator. These composite training data will be used to compute the minimum distance to the composite testing data, which combines between the network outputs and inputs.

Next, we will compute the covariance matrix for the composite training data. The eigenvectors of this covariance matrix will be used to transform the composite training data to principal components. Then, the transformed data set (i.e. principal components) will be utilized to compute the sum-square value in the process of outlier identification. The ν value, which determines which principal components we will use to compute the sum-square value, was selected by looking at the ordered variance of the transformed data set. Figure 82 shows the ordered variance of each principal component of the composite training data in Coil_1c.

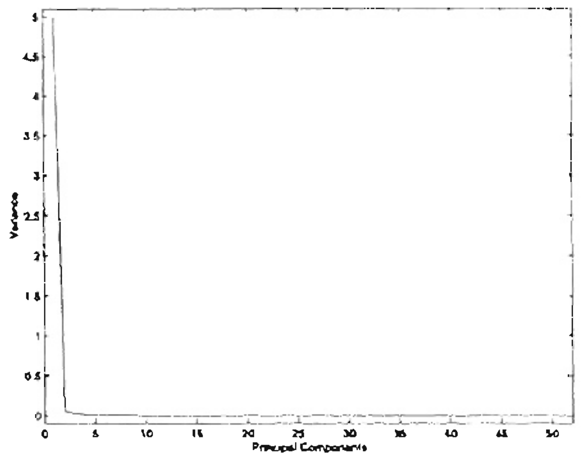

Figure 82 Variance of principal components: Training data

We will utilize the last PC to the PC that first gives variance equal to or above 0.02 for calculating the sum-square values. In this case, the variance of the third PC is 0.0302. Therefore, we will use the last PC to the third PC for outlier detection. Note that this value produced the fewest misclassifications for every coil (not only for Coil_1c). By using this criterion, the ν value shown in Equation (75) for Coil_1c was set to 50. After computing the sum-square values from the third PC to the 52^{th} PC, the highest sum-square value for the composite training data was 0.6574.

We need to transform the augmented testing data to the principal components via the eigenvectors and then compute the sum-square value of the last 50 PCs. Figure 83 plots the minimum distance between the composite training data and the augmented testing data (the first test group) versus the sum-square values. The data points in the first test group having small error (i.e. less than 0.15) are indicated by an $\mathbf{x}$.

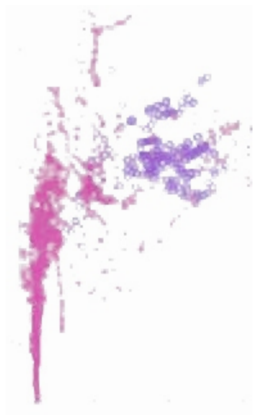

Figure 8.3. Minimum distance and the sum-square value: The first test group

We can see that though the minimum distance was as high as 0.25, most of the small-error data had a sum-square value less than 0.1. For minimum distances between 0.15 and 0.25 and sum-square values less than 0.1 (lying in the densely populated area of the training data), most data points had small errors.

Next, we will create the rejection region for discarding novel data (large-error points), based on this first test group. We first need to calculate minimum distance. Figure 8.4 shows the relationship between the approximation error and minimum distance on the first test group.

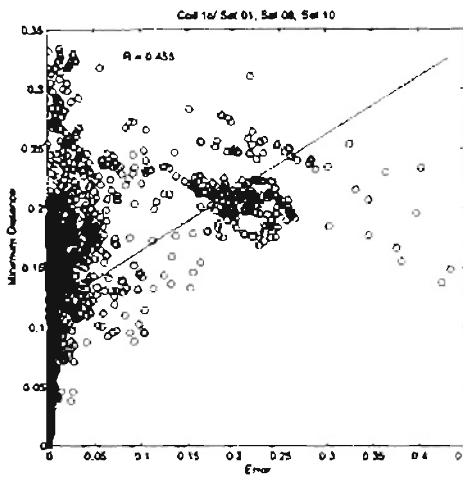

Figure 84 Minimum distance and approximation error: The first test group

The regression line is

$$d_m = 0.487 \times error + 0.1149 \tag{90}$$

The minimum distance corresponding to an approximation error of 0.15, T_d, is 0.188. Next, for minimum distances between $0.75 \times 0.188 = 0.141$ and $1.25 \times 0.188 = 0.235$, the sum-square value will be considered. Within this range, the average of the sum-square value for the small-error points, T_S^{ξ}, is 0.0684, while that for the large-error points (i.e. approximation error greater than 0.15), T_L^{ξ}, is 0.1946. Therefore, any data yielding minimum distance between 0.141 and 0.235 and producing the sum-square value greater than $\frac{0.0684 + 0.1946}{2} = 0.1315$ will be considered novel. In addition, data generating minimum distance larger than 0.235 is also presumed novel. In other words, the rejection region for discarding novel data is

$$(0.141 < d_m \leq 0.235 \text{ and } \xi > 0.1315) \text{ or } (d_m > 0.235) \tag{91}$$

Figure 85 plots the network error for the first test group. The abnormalities are flagged as $\mathbf{x}$.

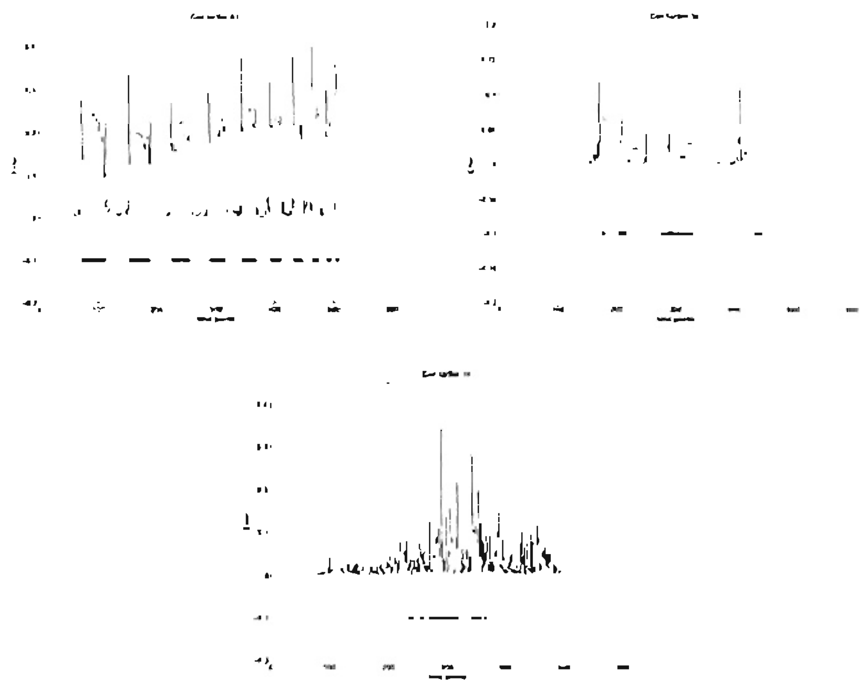

Figure 85 Error and abnormalities: The first test group (Minimum distance and PCA)

Most large-error points in set 01 and set 10 were identified as novel; however, in set 06 and set 10, many small-error points were detected as novel. The percentage of misclassifications was 10.44%. The majority of misclassifications (9.46% out of 10.44%) were from type I.

Next, we will apply the second test set to the novelty detector to verify that the rejection region works. Figure 86 demonstrates the network error for the second test group. The abnormalities identified by the novelty detector are indicated by an $\mathbf{x}$.

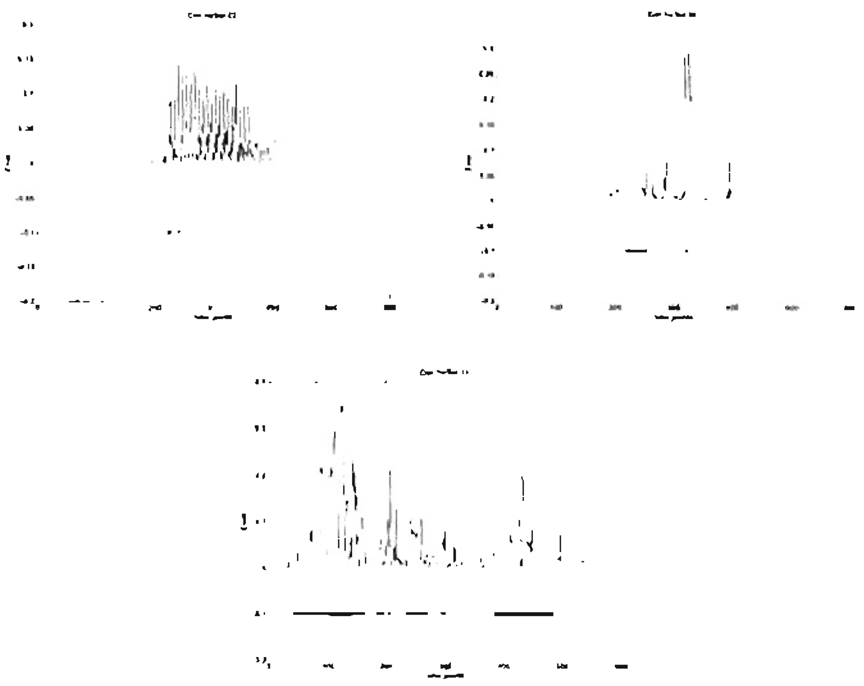

Figure 86 Error and abnormalities: The second test group (Minimum distance and PCA)

From Figure 86, we can see that some abnormalities in set 08 and all novel data in set 11 were identified, although some small-error points in every set were also detected. The total percentage of misclassifications for this test group was 15.37%. Most of misclassifications (14.91% out of 15.37%) were from type I errors.

The percentage of misclassified data points was equal to 0.22% if we applied the training data to the novelty detector. Around 0.19% (out of 0.22%) were from type II errors. Note that all of the data detected as novel for the training data set were from the rejection region of ($0.141 < d_m \leq 0.235$ and $\xi > 0.1315$).

From these results, this method provided the lowest percentage of misclassifications for the first test group and for the training data relative to another method.

In the next section, we will summarize the results for all of the algorithms.

Result summary

Table 2 summarizes the results from the first test group for all of the novelty detectors.

Table 2 Percentage of misclassifications: The first test group

Algorithm	Percentage of misclassifications		
	Type I	Type II	Total
Neural tree	67.87	0	67.87
Density of input	61.79	0.46	62.25
Density of input and output	59.44	0.57	60.01
Minimum distance	28.86	5.45	34.31
Minimum weighted distance	17.10	2.30	19.40
Minimum distance and outlier detection	9.46	0.98	10.44

Table 3 summarizes the results for the second test group.

Table 3 Percentage of misclassifications: The second test group

Algorithm	Percentage of misclassifications		
	Type I	Type II	Total
Neural tree	67.01	0	67.01
Density of input	50.03	1.55	51.58
Density of input and output	49.22	0.92	50.14
Minimum distance	34.65	0.58	35.23
Minimum weighted distance	29.78	0.58	30.36
Minimum distance and outlier detection	14.91	0.46	15.37

We can see that the neural tree algorithm turned out the highest percentage of misclassifications for both test groups. The minimum distance of the composite data with the outlier detection provided the minimum percentage of misclassifications for both the first and the second test groups. The Gaussian kernel estimators tended to reject more small-error points than any other method, excluding the neural tree. The minimum weighted distance had fewer misclassifications than the minimum distance algorithm for both test groups.

Table 4 shows the percentage of misclassifications when we applied training data to the novelty detectors. Note again that the percentage of large-error points (i.e. error greater than 0.15) in the training data set was 0.29%.

Table 4 Percentage of misclassifications: Training data

Algorithm	Percentage of misclassifications		
	Type I	Type II	Total
Neural tree	0	0.29	0.29
Density of input	34.64	0.09	34.73
Density of input and output	35.31	0.08	35.39
Minimum distance	0	0.29	0.29
Minimum weighted distance	0	0.29	0.29
Minimum distance and outlier detection	0.03	0.19	0.22

The joint density using the Gaussian kernel estimator had the highest percentage of misclassifications in the training data set. The majority of misclassified points were from the small-error data flagged as novel points (Type I). However, it gave the lowest percentage of misclassified points for the large-error data (i.e. error greater than 0.15). The minimum distance with outlier detection had the lowest percentage of misclassifications for the training data set.

In the next section, we will utilize another application to test the performance of the various novelty detectors.

Function approximation II

We will begin this section by briefly describing the data set in this part, followed by the simulation results of each novelty detector.

For this example, we will create a neural network to model a diesel engine. The inputs to the function approximator are comprised of speed and fueling, while the output is torque. We chose to have a two-layer network, with 20 neurons in the first layer. The output layer must have one neuron. In other words, our neural network structure is $2-20-1$.

We collected 1,049 observations from speed, fueling, and torque. The observations were divided into two groups – 599 samples used for training, and 450 samples used for validation. The network was trained using the Bayesian regularization algorithm. The error from the validation set monitored so that we could stop training if the error increased. Note that we normalized our data within the range $[-1,1]$.

After training the network, we found that the function approximator was generally accurate. However, 12.69% of the training data still yield large approximation error. Figure 87 shows the locations of the training inputs, and the red points represent data producing large approximation error.

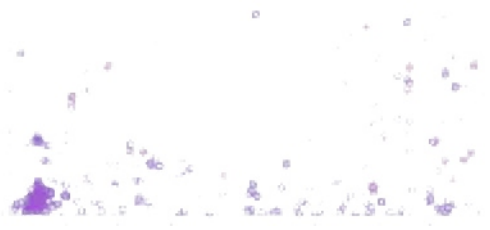

Figure 87. Data points with large and small approximation error - Training data

We also have another 450 observations used for testing. We sample 225 observations for the first test group in order to set up parameters, such as the threshold to discard novel data. The other 225 observations will be used for the second test group in order to verify the performance of novelty detectors. The locations of these testing data will be shown in the next section, along with the performance of the neural tree novelty detector.

In this section, we briefly described the application where the function approximator was used. In the next section, we will see the performance of the neural tree novelty detector.

Neural Tree

In this example, we will use a 200-cell tree to partition the two dimensional space. Then we apply the 599 training vectors, using the same learning rate we used in the previous example. After limiting the size of every cell with infinite volume, we apply our testing data (first group) to the trained tree to find abnormalities. The algorithm to define novel data was the same as the example in Chapter 3 (data are identified as novel if they are outside of the cell boundaries). Figure 88 demonstrates the data points identified as novel.

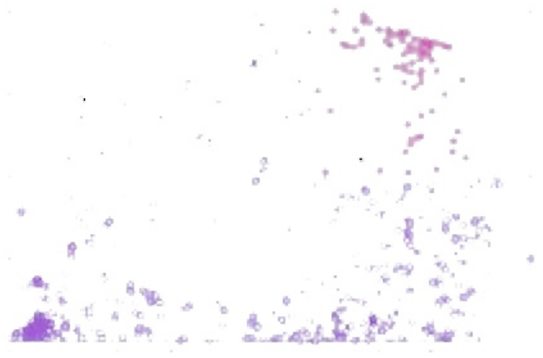

Figure 88: Novel Data - The first test group (Neural Tree)

We can see that from the figure that most data in the first quadrant are identified as novel. Figure 89 illustrates the approximation error for the first test set, and the abnormalities identified by the 200-cell tree.

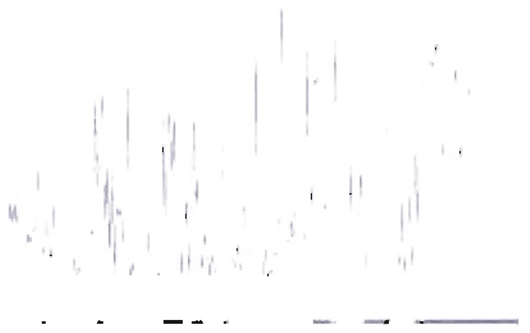

Figure 89: Error and abnormalities - The first test group (Neural Tree)

There are 23.11% misclassifications in total. About 17.33% are type II errors. We can see that, in this case, the percentage of type II misclassifications are the majority, and we do not have as many type I errors as in the previous example. This is because many large-error data points are within the cells (This phenomenon was discussed in Chapter [illegible]

Next, we applied the second test group to the trained tree. Figure 90 illustrates novel data pinpointed by the 200-cell tree.

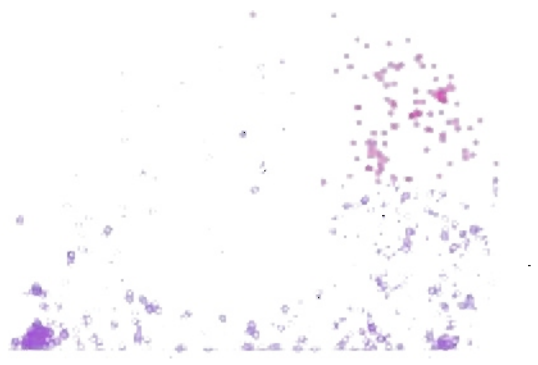

Figure 90. Novel data: The second test group (Neural Tree)

We can see that the result is quite similar to the first test group. Figure 91 shows the approximation error and novel data for the second test group.

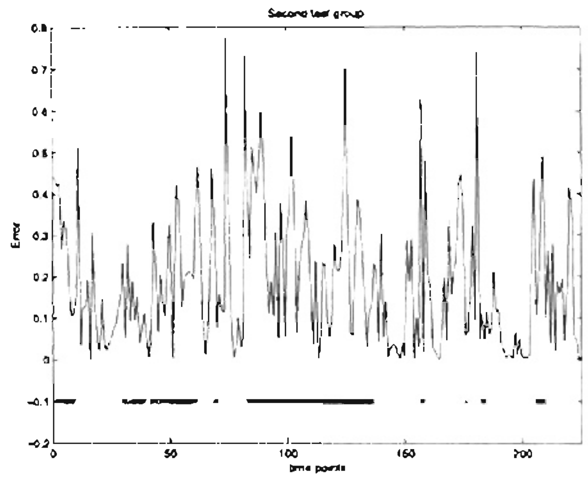

Figure 91 Error and abnormalities: The second test group (Neural Tree)

In this test set, there are 35.56% misclassifications. Around 16% are type I error, and 19.56% are type II error. Although type I error increases by a certain amount from the first test group, it is still less than type II error. We also found that the percentage of limited cells is only 12.5%. Therefore, when compared with the previous example, we are *less* likely to find data points close to training data but outside the boundaries, thereby reducing the probability of existing type I misclassification.

Now, when we applied the training data to the novelty detector, there were 12.69% misclassifications. All of the misclassifications are type II errors. This is because all of these large approximation error points are within the cells (since they are training data).

In the next section, we will test the performance of the Gaussian kernel estimator.

The Gaussian kernel estimator (GKE)

In this section, we will use the Gaussian kernel estimator to estimate the input density, and to estimate the input-output density of the function approximator. The estimated density will be used to identify novel data.

The estimated density of input

For the smoothing parameter matrix, we will use the same approach that we used in the previous example – the average distance to the ten-nearest neighbors. We found that the smoothing parameter matrix in this case is:

$$\Sigma_p = \begin{bmatrix} 0.0517^2 & 0 \\ 0 & 0.0517^2 \end{bmatrix} \tag{92}$$

After using Equation (33), the estimated density for the first test group is obtained. Figure 92 plots the estimated density versus the approximation error for the first test group.

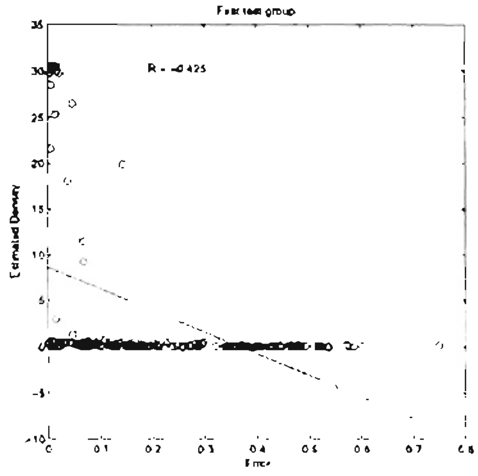

Figure 92 Estimated density and approximation error: The first test group

Again, the figure indicates that, at the low density value, we are more likely to find large-error points. We found that the regression line for this data set is

$$density = -23.338 \times error + 8.5863 \tag{93}$$

When substituting 0.15 in the error term, we end up with 5.075. We will use this value to be the threshold to reject novel data.

After using this threshold, we found that there are 36.89% misclassified points, and all of the misclassifications are type I errors. Figure 93 shows the novel data indicated by the algorithm.

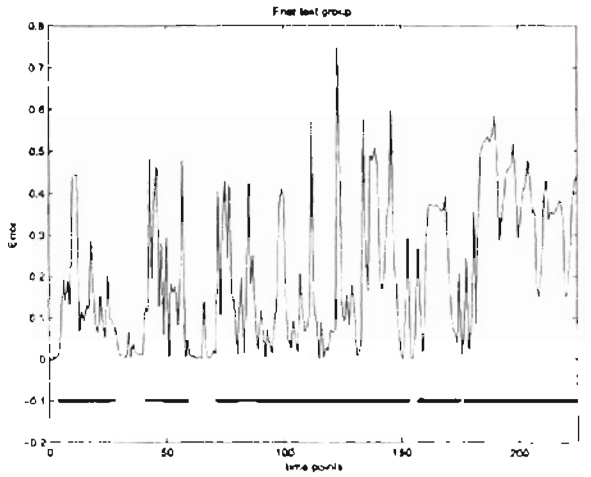

Figure 93 Error and abnormalities: The first test group (GKE: input)

Next, we applied the second test group and computed the density with the same smoothing parameter matrix. The density is fed to the novelty detector. The graph showing the approximation errors and novel data pinpointed by the algorithm for the second test group is shown in Figure 94.

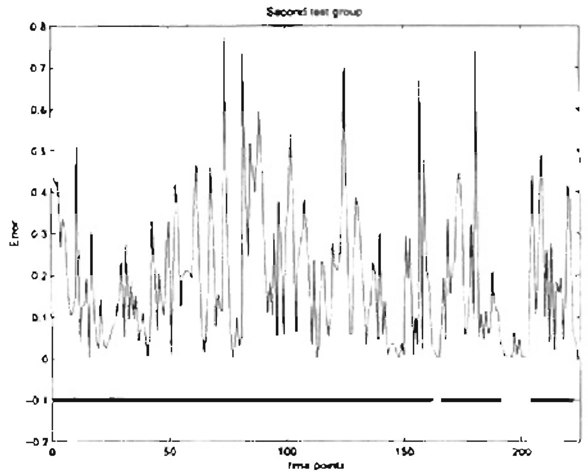

Figure 94 Error and abnormalities: The second test group (GKE: input)

We found that there are 43.11% misclassified points and all of these are type I errors. When applying the training data (inputs) to the novelty detector, we found that the percentage of misclassifications are 31.05%. All of the misclassifications are type I errors.

We can see from the results that the percentage of type I misclassifications is very high. This outcome is similar to what we had in the previous examples, and it follows the analysis we gave in Chapter 3 that this method tends to reject more small-error points, thus causing more type I errors.

In this section, we applied the Gaussian kernel estimator to estimate the density of inputs to the function approximator. In the next section, we will use the estimator to compute the density of inputs and outputs of the function approximator.

The estimated input-output density

In this part, we will use the Gaussian kernel estimator to compute the density of inputs and outputs of the function approximator. Since we have two dimensions for the inputs and the outputs are in one dimension, the composite data are in three dimensions.

We will use the same method to compute the smoothing parameter matrix, the ten-nearest neighbors for the b value. By using this algorithm, the smoothing parameter matrix will be:

$$\Sigma_{\Gamma} = \begin{bmatrix} 0.0846^2 & 0 & 0 \\ 0 & 0.0846^2 & 0 \\ 0 & 0 & 0.0846^2 \end{bmatrix} \tag{94}$$

After using Equation (36), we plot the estimated density and the approximation error for the first test group as follows:

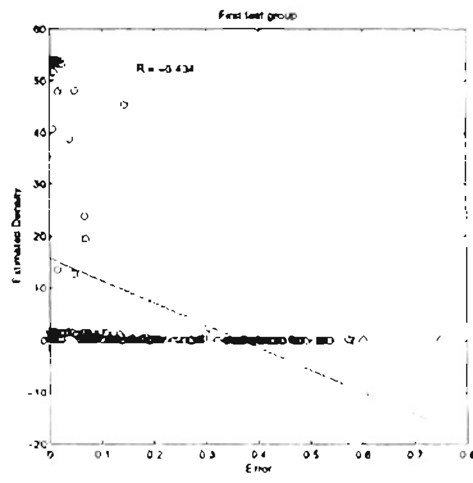

Figure 95 Estimated density and approximation error: The first test group

Again, the graph demonstrates a relationship between the approximation error and the estimated density. We found that the regression line in this example is

$$density = -42.61 \times error + 15.711 \tag{95}$$

Once again, we substitute 0.15 into the error term to obtain the threshold. From the equation, the density threshold is equal to 9.31. We will use this value to reject any data generating density lower than 9.31 as novel data. After discarding novel data based upon the threshold, there are 36% misclassified data in this case, which is slightly less than the percentage of misclassifications when we computed the density of inputs. All of the misclassifications are from type I errors. Figure 96 shows the novel data identified by the algorithm.

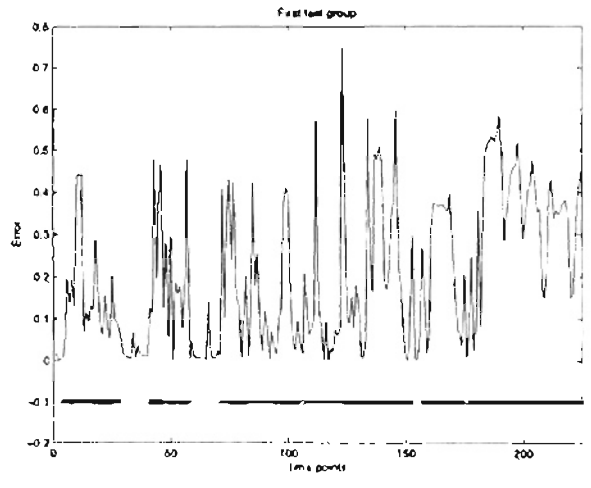

Figure 96 Error and abnormalities: The first test group (GKE: input and output)

Now, we applied the second group (composite data) to the density estimator. Figure 97 illustrates the novel data identified by the novelty detector.

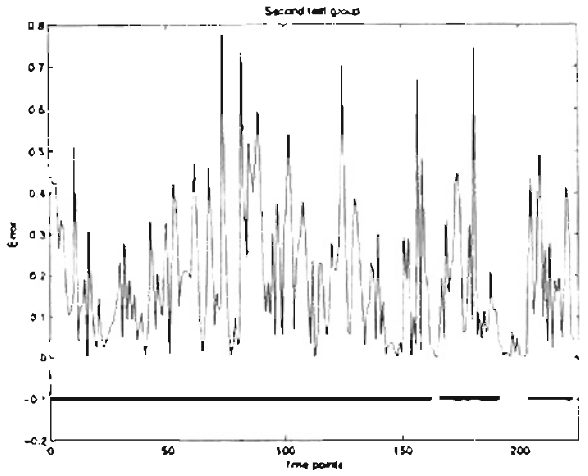

Figure 97 Error and abnormalities: The second group (GKE: input and output)

We found that there are 43.11% misclassifications for this group. Again, all of the misclassifications are from type I error. The percentage of misclassifications in this case is about the same as the outcome from the densities of inputs. However, we found that a strong relationship exists when we reduce the smoothing parameter matrix from the value we used. This effect is similar to the previous example (function approximator I). Therefore, a further analysis may be necessary for choosing the smoothing parameter matrix to reduce the percentage of misclassifications.

When we applied the training data to the novelty detector, we found that there are 29.88% misclassifications, and all of these are from type I error. We can see that the percentage of misclassifications are less than the outcome when we utilized the density of inputs for novelty detection (29.88% versus 31.05%).

In this section, we applied the Gaussian kernel estimator to estimate the density of the inputs and outputs of the function approximator. The results showed that the percentages of misclassifications were reduced when we used both the outputs and inputs to compute density (when compared with the results from using only density of inputs).

Minimum distance algorithm

In this section, we will test the performance of novelty detector using minimum distance and minimum weighted distance algorithms.

Minimum distance

By using Equation (45) and Equation (46) to compute the minimum distance of each testing data, we end up with Figure 98. The figure illustrates the approximation errors and the minimum distances for the first test group.

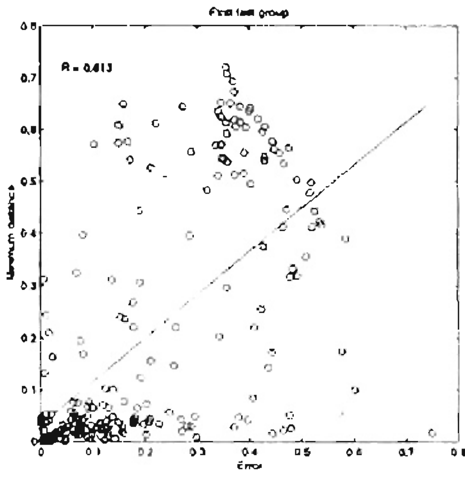

Figure 98 Minimum distance and approximation error: The first test group

As we see, when minimum distance gets higher, it is likely to find more large-error points. On the other hand, when minimum distance is low, we are likely to have more small-error data. Another parameter that indicates this relationship is the correlation coefficient (R value). This value is quite high implying that there is a relationship between the approximation errors and minimum distances. From the figure, we found that the regression line is

$$d_m = 0.8264 \times error + 0.0344 \tag{96}$$

By substituting 0.15 in the error term, we obtain $d_m = 0.1584$. We will use this value to be the threshold. For the first test group, Figure 99 shows the approximation errors and the data points identified by the algorithm as novel data.

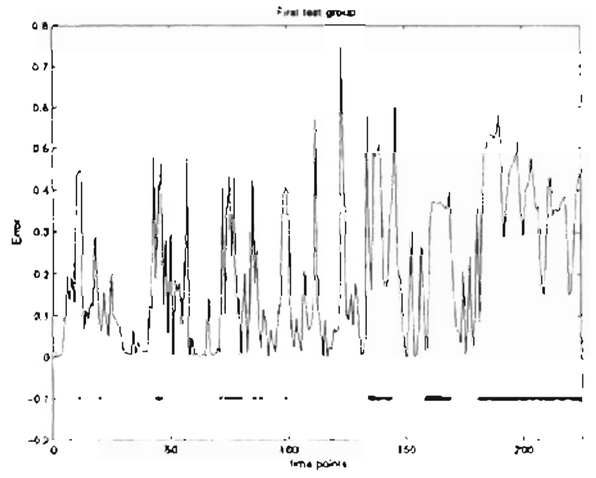

Figure 99 Error and abnormalities: The first test group (Minimum distance)

We found that there are 21.78% misclassifications in the first test group. The majority (16.44% out of 21.78%) are from type II error. That implies that, for this data set, we have many data close to training data that generate large errors (since type II misclassifications are quite high, compared with the example of the function approximator I). This phenomenon makes some sense, because even the training data generate large approximation errors (see Figure 87).

Next, we applied the second test group to the novelty detector. Figure 100 demonstrates the approximation errors and novel data are identified by **x**.

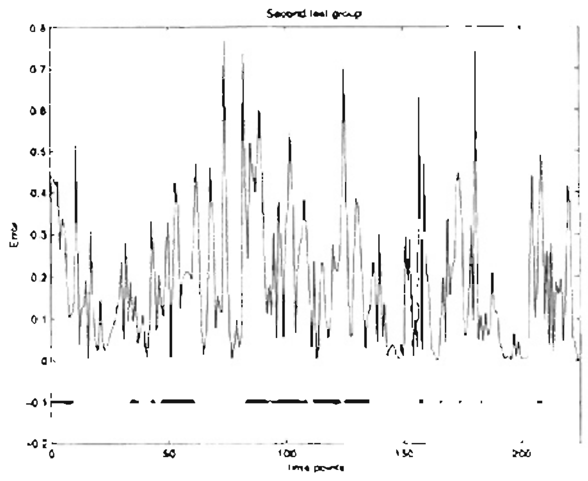

Figure 100 Error and abnormalities: The second test group (Minimum distance)

We found that around 33.33% are misclassifications. Again, most of these misclassified points (18.22% out of 33.33%) are from type II errors. We can also see that we had more type II misclassifications when compared with the Gaussian kernel estimator. However, this algorithm turned out less total misclassifications than the Gaussian kernel estimator.

When applying the training data set to the novelty detector, we found that 12.69% are misclassified, and all of these are from type II error. This is due to the fact that the minimum distances of training data equal zero, which is less than the threshold (0.1584). Therefore, the novelty detector accepts all of these points as old data.

In this example, we applied the minimum distance algorithm to be a novelty detector for the function approximator. The results showed that the percentage of misclassifications was less than the outcomes from the neural tree and the Gaussian kernel estimator. In the next section, we will test the performance of minimum weighted distance for novelty detection.

Minimum weighted distance

In this section, we will apply the minimum weighted distance for novelty detection. We will use Equation (52) and Equation (53) to find minimum weighted distances for the first test group. After computing the correlation coefficient (R value) between the minimum weighted distances and the approximation errors for different values of weighting factor (α), we obtain Figure 101. This figure illustrates the effect of weighting factor to the correlation coefficient as we varied the value of weighting factor.

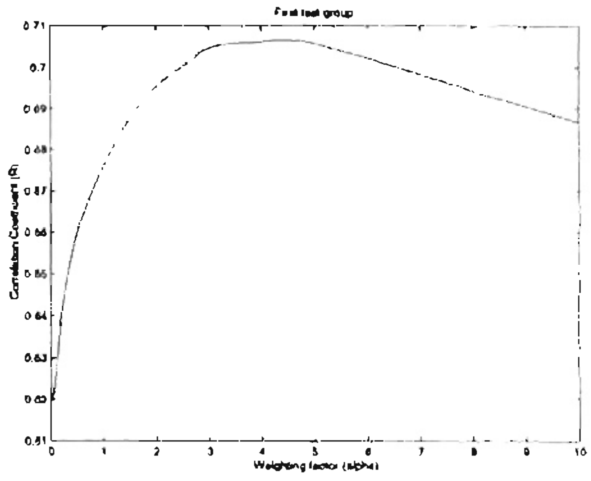

Figure 101 The weighting factor and the correlation coefficient: The first test group

We can see from the figure that when we added the effect of the difference between the targets and network outputs, the correlation coefficients are higher. However, when the weighting factor is too high, the R value will be lower. In this data set, the weighting factor

$\alpha = 4.35$ causes the highest correlation coefficient ($R = 0.71$), which is higher than the R value in the case of using minimum distance ($R = 0.61$).

We will then use the weighting factor $\alpha = 4.35$ for our novelty detector. At this weighting factor, Figure 102 shows the correlation between the minimum weighted distances and the approximation errors for the first test group.

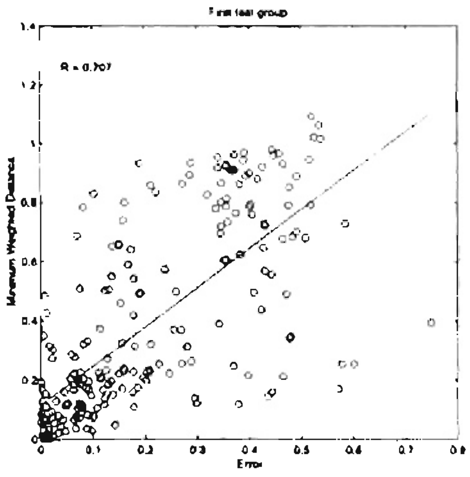

Figure 102 Minimum weighted distance and approximation error: The first test group

Now, we will use the figure to create the threshold to reject novel data. We found that the regression line for this data set is

$$d_m^{4.35} = 1.3198 \times error + 0.1158 \tag{97}$$

After substituting 0.15 into the error term, we obtain 0.3137 for the minimum weighted distance at weighting factor of 4.35. We will use this value to be the threshold to discard novel data. That means that any data generating minimum weighted distance (at $\alpha = 4.35$) larger than 0.3137 will be identified as novel. Figure 103 shows the approximation errors and novel data are marked with $\mathbf{x}$.

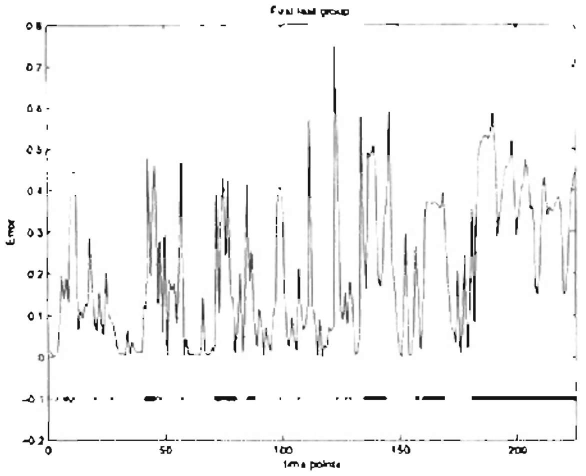

Figure 103 Error and abnormalities: The first test group (Minimum weighted distance)

We found that there are 19.11% misclassifications. Most of the misclassifications (12% out of 19.11%) are from type II error. The percentage is slightly less than we had using minimum distance.

Next, the second test group will be applied to the novelty detector. Figure 104 illustrates the novel data for the second test group.

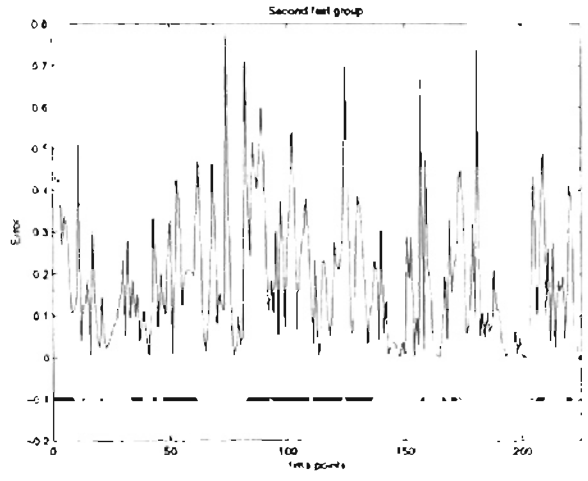

Figure 104 Error and abnormalities: The second test group (Minimum weighted distance)

In this data set, 33.33% are misclassifications. Around 18.22% out of 33.33% are from type II error. We can see that the percentage of misclassifications in this case is equal to the case for minimum distance. This result is a good example showing that, for different data sets, the performance of the weighting factor can vary. In this particular case, the performance of the minimum weighted distance is equal to the performance of minimum distance.

When the training data set is applied to the novelty detector, there are 12.69% misclassifications. Around 10.35% out of 12.69% are from type II error. Although type II misclassification in this case is less than minimum distance, the total percentage of misclassifications is equal. This is due to the fact that we choose the best weighting factor based on the first test group, not the training data set. This is another example showing the flaw of this algorithm.

In this section, we demonstrated the performance of the novelty detector employing minimum distance and minimum weighted distance. The outcomes illustrated that the performance of minimum weighted distance is better for the first test group. However, it also showed the drawback of minimum weighted distance on the second test group and the training data set. In the next section, we will use outlier detection and minimum weighted distance of composite data for novelty detection.

Minimum distance and outlier detection

In this section, we will apply outlier detection using principal component analysis and minimum distance on composite data for novelty detection.

We begin this example by creating composite training data by augmenting the targets and the inputs to the function approximator. These composite training data will be used to compute the minimum distance for the composite testing data, which includes both network inputs and outputs.

Next, we will compute the covariance matrix from the composite training data. The covariance matrix is

$$\Sigma = \begin{bmatrix} 0.3438 & 0.0816 & 0.0789 \\ 0.0816 & 0.1002 & 0.0987 \\ 0.0789 & 0.0987 & 0.1195 \end{bmatrix} \tag{98}$$

With the above equation, the associated eigenvalues and eigenvectors are as follow

$$\begin{aligned} \lambda_1 &= 0.4083, \upsilon_1^T = \begin{bmatrix} 0.8692 & 0.3441 & 0.3552 \end{bmatrix} \\ \lambda_2 &= 0.1447, \upsilon_2^T = \begin{bmatrix} -0.4937 & 0.5638 & 0.6621 \end{bmatrix} \\ \lambda_3 &= 0.0105, \upsilon_3^T = \begin{bmatrix} 0.0276 & -0.7508 & 0.6599 \end{bmatrix} \end{aligned} \tag{99}$$

The eigenvectors will be used to transform the composite training data to principal components. Then we will utilize the principal components to compute the sum-square value in the process of outlier identification. The ν value, which determines which principal components we will use to compute the sum-square value, was selected by looking at the ordered variance of the transformed data set – the ordered eigenvalues. By using the same criterion as the previous example – use from the last PC up to the PC giving variance equal to 0.02 or above. For this example we will use the second and third PC's.

Next, we will use the first test group to compute the minimum distances and the sum-square values. Figure 105 shows the minimum distance and the sum-square value for the first test group.

Figure 105. Minimum distance and the sum-square value: The first test group

From the above figure, we can see that, for small error points, the sum-square value can be large at low minimum distance. However, when minimum distance is higher, the more likely we will experience large-error points. As we can see in this example, the small error points are not as distinguishable from large-error points as in the previous example (resistivity estimation) at higher minimum distances. It might due to the fact that the estimation for data points in the densely-populated area of the training data is not very accurate. (Recall that the estimation for the training data is not as precise as the previous example.) Therefore, at high minimum distance, no matter where the data points are (in densely-populated or sparsely-populated areas) the torque approximations are not accurate. With the above explanation, although not expecting to see much improved performance, we will see if this method will have an acceptable percentage of misclassifications.

In order to compute the percentage of misclassified points, we first need to have the regression line between the approximation error and the minimum distance (for the first test group). The relationship between these two variables is shown in Figure 106.

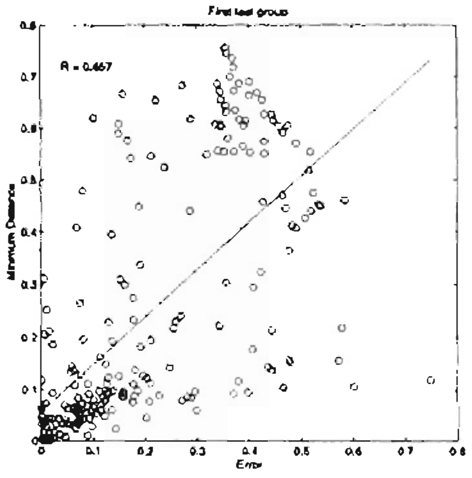

Figure 106 Approximation error and minimum distance: The first test group

The regression line shown in the figure is

$$d_m = 0.8976 \times error + 0.0585 \qquad (100)$$

By substituting 0.15 in the error term, the corresponding minimum distance, T_d, is 0.1931. Any data generating minimum distance greater than $1.25 \times 0.1931 = 0.243175$ will be discarded as novel data. On the other hand, we will accept any data producing minimum distance less than $0.75 \times 0.1931 = 0.144825$ as standard data. Based on the first test group, within the minimum distance range $(0.75 \times 0.1931, 1.25 \times 0.1931]$, the average sum-square value for data points generating approximation error less than 0.15, T_S^{ξ}, is 0.3285, while that for data creating approximation error greater than 0.15, T_L^{ξ}, is 0.3615.

Therefore, data having minimum distance within this range will be considered novel if their sum-square values are greater than $\frac{0.3285 + 0.3615}{2} = 0.345$. The rejection region is as follows

$$(0.144825 < d_m \leq 0.243175 \text{ and } \xi > 0.345) \text{ or } d_m > 0.243175 \tag{101}$$

We mark novel data with an **x** in Figure 107.

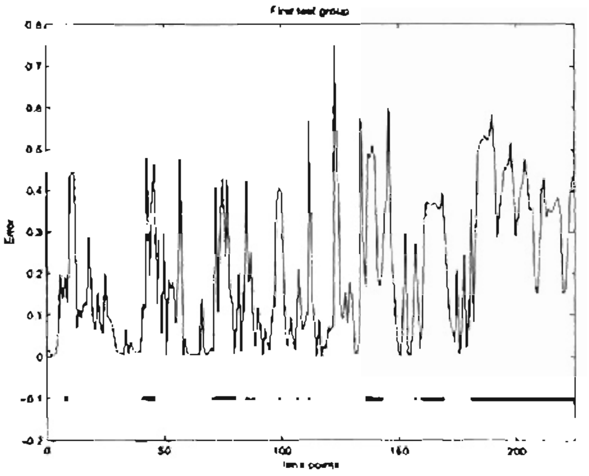

Figure 107 Error and abnormalities: The first test group (Minimum distance and PCA)

We found that there are 21.33% misclassified points in total. The majority (16% out of 21.33%) are from type II error. The result showed fewer misclassifications than any other method, except minimum weighted distance. This is because we maximized the correlation coefficient in the first test group in the case of minimum weighted distance.

Next, we will use the rejection region shown in Equation (101) for the second test group. Figure 108 illustrates the novel data and approximation error for the second test group.

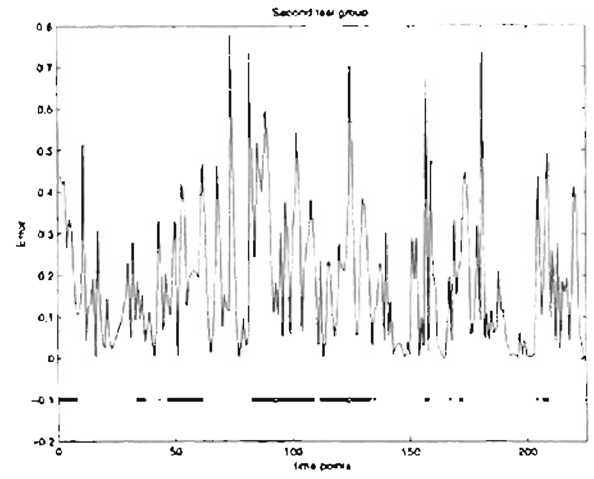

Figure 108 Error and abnormalities: The second test group (Minimum distance and PCA)

There are 30.67% misclassifications. Most of these misclassifications (20.89%) are from type II error. This method produced fewer misclassifications than any other method.

If we apply the training data to the novelty detector, there are 10.68% misclassifications. All of the misclassified points are from type II error. This percentage from this algorithm is less than for any other method.

What we can notice from this example is that type II error is always the majority of the total percentage of misclassifications (which is different from the first approximator). The main reason is that the second function approximator did not estimate its target as well as the first one does. Thus, this outcome is likely to increase type II misclassification.

In this section, we utilized the minimum distance algorithm and outlier detection employing principal components analysis over the composite data for novelty detection. The result is promising in terms of the percentage of misclassifications. We will summarize the outcomes of all of the novelty detectors in the next section.

Result Summary

We will summarize the performance of the novelty detectors in terms of percentage of misclassified points. We begin with Table 5, which shows the percentages of misclassified points for the first test group.

Table 5 Percentage of misclassifications: The first test group

Algorithm	Percentage of misclassifications		
	Type I	Type II	Total
Neural tree	5.78	17.33	23.11
Density of input	36.89	0	36.89
Density of input and output	36	0	36
Minimum distance	5.33	16.44	21.78
Minimum weighted distance	7.11	12	19.11
Minimum distance and outlier detection	5.33	16	21.33

The percentages of misclassifications for the second test group are summarized in Table 6.

Table 6 Percentage of misclassifications: The second test group

Algorithm	Percentage of misclassifications		
	Type I	Type II	Total
Neural tree	16	19.56	35.56
Density of input	43.11	0	43.11
Density of input and output	43.11	0	43.11
Minimum distance	11.56	21.78	33.33
Minimum weighted distance	15.11	18.22	33.33
Minimum distance and outlier detection	9.78	20.89	30.67

We can see that the Gaussian kernel method (density of input and density of input and output) turned out the highest percentage of misclassifications for both test sets, with very high percentage of type I error. The minimum weighted distance provided the minimum percentage of misclassifications for the first test set. (This is because we optimized the weighting factor from this test group.) However, for the second test group, the minimum weighted distance did not provide the best outcome (since the effect of weighting factor varies from data set to data set). Minimum distance with outlier detection produced an acceptable result for the first test group, but had the fewest misclassifications for the second test group.

Next, Table 7 shows the percentage of misclassifications when we applied training data to the novelty detectors. Note again that the percentage of large-error points (i.e. error greater than 0.15) in the training data set was 12.69%.

Table 7 Percentage of misclassifications: Training data

Algorithm	Percentage of misclassifications		
	Type I	Type II	Total
Neural tree	0	12.69	12.69
Density of input	30.89	0	30.89
Density of input and output	29.88	0	29.88
Minimum distance	0	12.69	12.69
Minimum weighted distance	0	12.69	12.69
Minimum distance and outlier detection	0	10.68	10.68

The density of input using the Gaussian kernel estimator had the highest percentage of misclassifications in the training data set. The majority of misclassified points was from type I error. It provided no type II misclassifications. Therefore, the density approximation method might be well fitted to problems where one needs very reliable network outputs. Finally, minimum distance with the outlier detection had the lowest percentage of misclassifications for training data set.

Summary

In this chapter, we tested and compared the ability of the novelty detectors described in Chapter 3 to Chapter 7 using two real-world applications. We began each application by explaining the objective for function approximation. We also described the training data and testing data for each application. After that, we used testing data to demonstrate and compare the capability of each novelty detector in terms of the percentage of misclassifications.

The objective of the first application was to estimate the resistivity of the earth formation in order to explore for existing oil or gas. In terms of the percentage of misclassifications, the simulations showed that the neural tree algorithm provided the highest percentage of misclassifications for both test groups. The minimum distance with outlier detection produced the fewest errors on every data set, e.g. the first, second test groups and the training data. For the misclassifications in the training data set, the joint density between input and output method produced the most misclassifications; however, it had the fewest type II errors.

For the second application, the objective was to estimate the torque of the diesel-engine system. The simulation indicated that the method of estimating density of input had the most misclassifications for every data set. The minimum weighted distance algorithm yielded the lowest percentage of misclassified points for the first test group. However, the minimum distance of composite data with outlier detection produced the lowest misclassifications for the second test group. For the training data, the minimum distance with outlier detection yielded the fewest misclassifications. The Gaussian kernel estimator had no type II misclassifications on any data set.

CHAPTER 9

CONCLUSIONS

In this chapter, we will briefly summarize the results of our work. It will be followed by recommendations for future work.

Summary of the results

We started this work by proposing a problem for function approximation: neural networks are very good at interpolating while poor at extrapolating. We have discussed the key novelty-detection methods to distinguish between data that require interpolation and data that require extrapolation. Each of these methods has its own advantages and drawbacks in terms of misclassified data. One of the contributions of this work is to compare the performances of each algorithm in terms of the percentage of misclassifications.

The neural tree algorithm is a very fast method. It can be implemented as an online novelty detector. However, in some applications, the percentage of misclassifications can be very high for small-error points that are identified as novel data (type I errors). On the other hand, the autoassociative method, although providing very good results in terms of the percentage of misclassifications, is very slow to train. Therefore, this method is not feasible for data having very high dimension.

The estimated density for inputs and the estimated joint density between inputs and targets using the Gaussian kernel are somewhat slow relative to other algorithms, and have a slightly higher percentage of misclassifications, especially for type I errors. This is due to the fact that these methods tend to reject training data or interpolation points whose approximation errors are small. A main drawback of the density estimation is the smoothing parameter that we must heuristically choose, which sometimes may not be well-suited to the true density.

The minimum distance algorithm has its own strengths in that both the computation time and the percentage of misclassifications are acceptable. There is no unknown parameter in this algorithm, making this algorithm the simplest algorithm to implement. We proposed a method to decrease the percentage of misclassifications by using the minimum weighted distance. A major drawback of the minimum weighted distance is the weighting factor, which is a varying parameter for different data sets. In other words, the weighting factor with the best result for novelty detection on one data set may not perform well on another data set. Another contribution derived from the minimum distance algorithm is the analysis for the approximated gradient to estimate the error of the function approximator. We concluded that the estimated derivative was not helpful for novelty detection for function approximation, but it was useful for estimating errors for interpolations. We also presented a way to reduce the computing time for the minimum distance algorithm by applying the Kohonen rule.

Another major contribution of this work is to apply outlier detection using principal components to the minimum distance algorithm. This method has several advantages over the minimum distance algorithm because of acceptable computing time, and a small percentage of misclassifications. Although this method seems to be very effective, the main disadvantage of this algorithm is the complicated threshold, and there is a parameter to be chosen in the process of the outlier detection.

We applied some of our novelty detectors to solve the real world applications, which were described in Chapter 8. We found that minimum distance with outlier detection had the best results. Furthermore, the simulation results confirm that the neural tree algorithm is very fast.

Recommendations for future work

In real world applications containing very high dimensional data, we concluded that it is almost impossible to utilize the autoassociative multilayer perceptron as our novelty detector, though this algorithm performed very well in our simple example. Future work could reduce the dimension of the data before applying them to train the perceptron. By employing a principal component transformation to decrease the number of dimensions, the network could be more efficiently trained.

The minimum weighted distance is another area for future work in that there is an unknown weighting factor in the algorithm. Thus, it will be desirable to have an in-depth analysis for predicting what range the maximum weighting factor should have in order to increase the correlation coefficient.

In the algorithm which uses minimum distance and outlier detection, a potential future task would be to discover a technique to get rid of the unknown parameter in the process of the outlier identification.

Finally, a combination of the joint density using the Gaussian kernel and the minimum distance algorithm may be used to improve the efficiency of identifying novel data and to reduce the percentage of misclassifications, especially type I errors (small-error points flagged as novel data).

REFERENCES

[AdWe99] J. F. D. Addison, S. Wermter and J. MacIntyre, "Effectiveness of feature extraction in neural network architectures for novelty detection," in *Proceedings of the 9th International Conference on Artificial Neural Networks*, vol. 2, pp. 976-981, 1999.

[Bhar00] S. Bhartiya, "Enhancements for Model Predictive Control and Inferential Measurement," *Ph.D. Dissertation*, Oklahoma State University, Stillwater, 2000.

[Bish93] C. M. Bishop, "Neural network validation: an illustration from the monitoring of multi-phase flows," in *Proceedings of the 3rd International Conference on Artificial Neural Networks*, pp. 41-45, Brighton, UK, 1993.

[Bish94] C. M. Bishop, "Novelty detection and neural network validation," in *Proceedings of the IEE Vision, Image and Signal Processing*, vol. 141, pp. 217-222, 1994.

[Bish95] C. M. Bishop, *Neural Networks for Pattern Recognition*, New York: Oxford University Press, Inc., 1995.

[Fahl89] S. E. Fahlman, "Fast learning variations on back-propagation: An empirical study," in *Proceedings of the 1998 Connectionist Models Summer School*, D. Touretzky, G. Hinton and T. Sejnowski, eds., San Mateo, CA: Morgan Kaufmann, pp. 38-51, 1989. (Chapters 12-19)

[FrGo96] A. Frosini, M. Gori and P. Priami, "A Neural Network-Based Model for Paper Currency Recoginition and Verification," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1482-1490, 1996.

[GnKe72] R. Gnanadesikan and J. R. Kettenring, "Robust estimates, residuals, and outlier detection with multiresponse data," *Biometrics*, vol. 28, pp. 81-84, 1972.

[HaDe96] M. T. Hagan, H. B. Demuth and M. Beale, *Neural Network Design*, Boston: PWS Publishing Co., 1996.

[HaMe94] M. T. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989-993, 1994. (Chapter 12)

[Hawk74] D. M. Hawkings, "The detection of errors in multivariate data using principal components," J. Amer. Statist. Assoc., vol. 69, pp. 340-344, 1974.

[Hawk80] D. M. Hawkings, *Indentification of Outliers*, London: Chapman and Hall, 1980.

[HiAu00] S. J. Hickinbotham and J. Austin, "Neural networks for novelty detection in airframe strain data," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 6, pp. 375-380, 2000.

[Hotel33] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *J. Educ. Psychol.*, vol. 24, pp. 417-441, 498-520, 1933.

[Hotel36] H. Hotelling, "Simplified calculation of principal components," *Psychometrika*, vol. 1, pp. 27-35, 1936.

[HwCh99] B. Hwang and S. Cho, "Characteristics of Autoassociative MLP as a Novelty Detector," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 5, pp. 3086-3091, 1999.

[Joll86] I. T. Jolliffe, *Principal Component Analysis*, New York: Springer-Verlag Inc., 1986.

[MaRo85] J. Markhoul, S. Roucos and H. Gish, "Vector quantization in speech coding," in Proceedings of IEEE, vol. 73, no. 11, pp. 1551-1588, Nov. 1985.

[Mart98] D. Martinez, "Neural tree density estimation for novelty detection," *IEEE Transactions on Neural Networks*, vol. 9, no. 2, pp. 330-338, 1998.

[NaCo97] A. Nairac, T. Corbett-Clark, R. Ripley, N. Townsend and L. Tarassenko. "Choosing an appropriate model for novelty detection," in *Proceedings of the 5th International Conference on Artificial Neural Networks*, pp. 117-122, Cambridge, 1997.

[Pear01] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Phil. Mag.*, vol. 2, pp. 559-572, 1901.

[PeMa96] T. Petsche, A. Marcantonio, C. Darken, S. J. Hanson, G. M. Kuhn and I. Santoso, "A neural network autoassociator for induction motor failure prediction," *Advances in Neural Information Processing Systems*, vol. 8, pp. 924-930, 1996.

[PoGi90] T. Poggio and F. Girosi, "Networks for approximation and learning," in *Proceedings of the IEEE*, vol. 78, no. 9, 1990.

[Rao64] C. R. Rao, "The use and interpretation of principal component analysis in applied research," *Sankhya A*, vol. 26, pp. 329-358, 1964.

[RiGr91] E. A. Riskin and R. M. Gray, "A greedy tree growing algorithm for the design of variable rate vector quantizers," *IEEE Transactions on Signal Processing*, vol. 39, pp. 2500-2507, 1991.

[Silv86] B. W. Silverman, *Density Estimation*, New York: Chapman and Hall, 1986.

[TaHa95] L. Tarassenko, P. Hayton, N. Cerneaz and M. Brady, "Novelty detection for the identification of masses in mammograms," in *Proceedings of the 4th International Conference on Artificial Neural Networks*, pp. 442-447, 1995.

[TaNa99] L. Tarassenko, A. Nairac, N. Townsend, I. Buxton and P. Cowley, "Novelty detection for the identification of abnormalities," *International Journal of Systems Science*, 1999.

[TaNa99] L. Tarassenko, A. Nairac, N. Townsend and P. Cowley, "Novelty Detection in Jet Engine," in *IEE Colloquium on Condition Monitoring: Machinery, External Structures and Health*, pp. 4/1-4/5, 1999.

[WaMe95] D. Wackerly, W. Mendenhall III and R. L. Scheaffer, *Mathematical Statistics with Applications*, Fifth Edition, Boston: PWS Publishing Co., 1995.

VITA

Arjpolson Pukrittayakamee

Candidate for the Degree of

Master of Science

Thesis: NOVELTY DETECTION FOR FUNCTION APPROXIMATION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Bangkok, Thailand, on June 10, 1977, the son of Ongarj and Boontida Pukrittayakamee.

Education: Graduated from Traim Udom Suksa School, Bangkok, Thailand, in April 1993; received Bachelor of Engineering degree in Electrical Engineering from Chulalongkorn University, Bangkok, Thailand, in April 1997. Completed the requirements for the Master of Science degree in Electrical Engineering at Oklahoma State University in December, 2001.

Experience: Employed by The National Electronics and Computer Technology Center (NECTEC) as an Engineer from 1997 to 1998; employed by Electrical and Computer Engineering department, Oklahoma State University, as a Research Assistant from 2000 to 2001.

Professional Status and Memberships: Licensed as a First-level Engineer in Power System from and Member of Engineering Institute of Thailand under H.M. the King's Patronage since June, 1997.