SEQUENCES OF BAYES GAUSSIAN CLASSIFIERS

by

JIMY SHAH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2007

ACKNOWLEDGEMENTS

My various experiences as a graduate student at University of Texas have instilled much stronger personality in me than I could ever have hoped for. I could not have achieved much at UTA were it not for the support given to me by certain individuals, who I will like to acknowledge.

My foremost appreciation goes to Dr. Michael Manry, my advisor, who has been a great source of inspiration throughout my graduate school years. He saw the work from inception to fruition and provided all the help to make this work possible. I admire his subject expertise, contribution and devotion to the field of Neural Networks, and incessant help to his students in various forms viz. teaching, regular laboratory visits, immediate feedback and motivation to better understand the field of Neural Networks. He patiently directed me through my research, and taught me all I know about critical reasoning and analysis, presentation of work done and even basic requirements. I believe he is the best counselor one can ever hope for. I thank Dr Stephen Gibbs and Dr. Bernard Svihel for reviewing my work and also for agreeing to serve on my thesis committee.

The EE coursework at UTA and my undergraduate institute Gujarat University in India provided the fillip and toolkit to weave the pieces of this work together. Hence a sincere thanks to all the teachers who selflessly strive to spread education to whom I dedicate this work.

Finally, I must express my sincere gratitude to my family for all their love and support. My parents, Mr. Vallabh Shah and Mrs. Smita Shah, for all their love, faith and support. My sister and my brother-in-law, Jeny and Jashmin, for being there for me whenever I needed them. Specially Mr. Jashmin Shah, whose ideas influenced my way of thinking and changed my approach towards graduate studies and research activities. He has been and will be a role model for me.

November 26, 2007

ABSTRACT

SEQUENCES OF BAYES GUASSIAN CLASSIFIERS

Publication No. ______

Jimy Shah, MS

The University of Texas at Arlington, 2007

Supervising Professor: Michael T Manry

A new method for designing sequences of Bayes Gaussian Classifiers is presented in this thesis. First, a basic Bayes Gaussian Classifier is designed with an assumption of data being Gaussian. Then, we have used the Output Weight Optimization-Back Propagation (OWO-BP) technique to iteratively modify the coefficients of the classifier, resulting in less classification error. Through use of an iterative Gram-Schmidt procedure, to train linear functional link nets, input features are ordered from most useful to least useful. Another important development in this thesis is the generation of nested feature subsets. This ensures that the curve for error percentage versus the number of features is monotonically non-increasing. Based upon this list of ordered features, nested feature subsets are produced, with a Bayes Gaussian Classifier designed for each subset. These classifiers exhibit reduced probability of error as the subset size (number of selected inputs) increases. Various real world data have been used to test and verify the classifier’s performances.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In a pattern recognition system the given sensor data is segmented and the features are extracted from it. Using these features in an input vector, a classifier is designed. Based on a decision rule, an estimate is made of the class the data belongs to. Actual pattern recognition systems may be more complicated and may have many more elements. A simplified block diagram of a recognition system is presented below with all its main functional blocks.

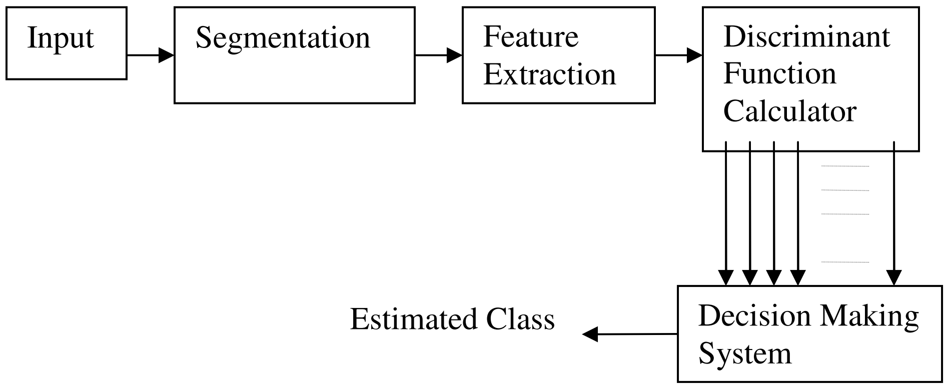

Figure 1.1 Recognition system with classifier

1.1 Classifier application

There are countless pattern recognition applications where classifiers are used. Some examples are given here:

In automatic form reading, character recognition techniques are used to read the forms and identify their content. In automatic mail processing for example, all the different regions of interest on the envelope are segmented such as main address, return address, barcode etc. Out of all these, the zip code is the main region of interest. Face recognition is another developing application which is often used to restrict building access to authorized people. Here we have a database which contains faces of those individuals who are to have access. When a person seeks access, the recognition system extracts features from the face image and decides whether the face is similar to one of the stored ones. Fingerprint processing is similar, but it's a bit simpler and more practical since fingerprints are almost two-dimensional.

Automatic target recognition is a defense related application where we try to locate man-made objects and classify them as friend or foe. Bottle cap recognition is used by various airlines where they need to sort out different beverages bottles automatically.

1.2 Classifier design approach

The task of a classifier is to use the feature vector provided by the feature extractor (fig 1.1) and to assign it to the correct class. We can split a typical classifier design problem into two categories, training and validation. Usually the process of providing data to the classifier and allowing it to adapt itself is called training. The most effective methods for developing classifiers involve learning from a set of example patterns that have already been classified. This type of learning is called supervised learning [10, 11] and the set of example patterns are referred to as the training dataset. A training dataset usually consists of N_v labeled feature vectors $\mathbf{X_p}$, each of dimension N. Each feature vector has its class label, $i_c(p)$, defined, where p is the pattern number and $1 \leq p \leq N_v$. The goal is to design a classifier that estimates $i_c(p)$ from $\mathbf{X}_p$, given the training data ($\mathbf{X_p}$, $i_c(p)$). Learning can also be unsupervised [10,11], in the sense that the system is not given a-priori information about patterns. Instead it establishes the classes, based on the statistical properties of the patterns. Validation of the classifier is extremely important to ensure that the classifier can perform on unseen real world data as well. It also gives us feedback of what changes should be made in the classifier to make it perform better. Sometimes a complex system may allow perfect classification of the training samples, but it may not perform well on new patterns during validation. This situation is known as overfitting [10].

1.3 Commonly used classifiers

Usually classifiers like the Bayesian classifiers [1], k-Nearest Neighborhood classifier (k-NNC) [33], Piecewise Linear Network classifier (PLNC) [34] and Neural Networks classifiers [35, 36, 42] are used.

The k-NNC classifies objects based on the closest training examples in the *N*-dimensional feature space. In the training phase, the feature space is divided into convex polygons or clusters based on the class labels of the various training patterns.

This leads to partitioning of the input space into a Voronoi tessellation [20] as shown in fig. 1.1. In the classification phase, distances from the new test vector to all the stored vectors are computed and the k closest samples are selected. The new vector is predicted to belong to the most numerous class labels within this set. The best choice of k depends on the data; generally larger values reduce the effect of noise on classification but make the decision boundaries less distinct. The algorithm is easy to implement, but can get computationally intense, especially when the size of the training set increases.

Several artificial neural networks have also been used for classification purposes. Neural nets can typically undergo supervised learning [1,19]. In supervised learning, there exist the input feature vector, $\mathbf{X_p}$ and the feature vector's class label, $i_c(p)$. Multi-layer Perceptrons (MLP) [16], radial basis function (RBF) networks [21] and support vector machines (SVM) [17,18] are trained using supervised learning techniques. Neural net classifiers are usually trained to minimize the Mean-Square Error (MSE) over the number of iterations.

1.4 Introduction to Bayes Gaussian Classifier

A Bayes classifier is a probabilistic classifier that makes decisions by combining two sources of information, i.e., the prior and the likelihood, to form a posterior probability using Bayes' rule [22]. When the feature vectors are jointly Gaussian, the result is the Bayes Gaussian Classifier (BGC).

Unfortunately real world feature data is not often Gaussian and data statistics are not known with accuracy. Also, it is not known which subset of the extracted features provides the best performance. In this thesis we try to solve these problems by integrating feature selection & training into the BGC in order to improve its performance.

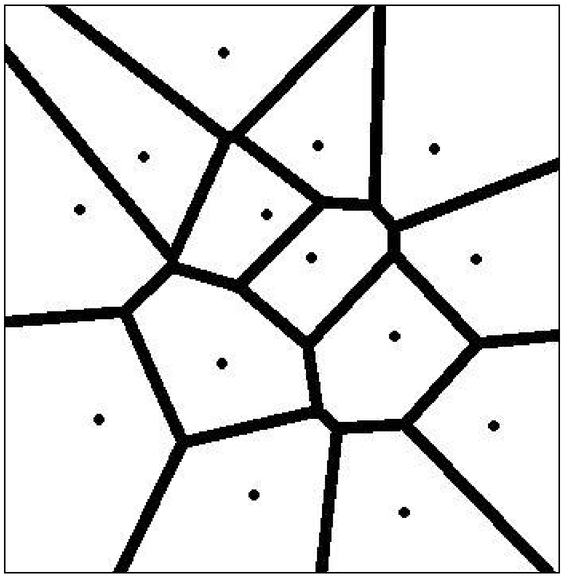

Figure 1.2 Voronoi tessellation of a two-dimensional space

1.5 Chapter flow

In Chapter II we discuss various types of classifiers and their limitations. In Chapter III we discuss the Bayes Gaussian Classifier, its limitations and our approaches for improving its performance. We also discuss a training technique for the classifier. In Chapter IV we discuss at length the Gram-Schmidt orthonormalization algorithm and its implementation. We show how this orthonormalization technique is used for feature ordering. Also, various different forms of this ordering are discussed. In Chapter V we compare the performance of our modified BGC with other available classifiers. In Chapter VI we conclude the thesis and discuss future work.

CHAPTER 2

REVIEW OF CLASSIFIERS

2.1 Bayes Classifier

2.1.1 Introduction

A classifier calculates discriminant functions for each class and makes the decision according to which class's discriminant is largest or smallest.

Bayes Classifier is a simple probabilistic classifier based on the Bayes rule [22]. This classifier can be designed if statistical information of the system including conditional probability density of the feature vectors is available and well defined.

2.1.2 Derivation of Bayes classifier

Let P_i denote the probability that a feature vector **X** is from the ith class and let P_e denote the probability of classification error. Our goal in Bayes classifier design is to develop the discriminant function that minimizes P_e. First, P_e is written as

$$P_e = \sum_{i=1}^{Nc} P(\text{error and } \mathbf{X} \text{ is from i}^{\text{th}} \text{ class}) \tag{2.1}$$

where N_c is the total number of classes. Continuing, we get

$$\sum_{i=1}^{Nc} P\left(error \mid i\right) P_i \tag{2.2}$$

This can be expanded as

$$\sum_{i=1}^{Nc} \int_{Z_i^c} f\left(\boldsymbol{X} \mid i\right) d\boldsymbol{X}\, P_i \tag{2.3}$$

where, f(**X**|i) is the conditional density of **X** given that it comes from the ith class. Z_i is the region or subset of R^N where we decide class i and Z_i^C is the region or subset of R^N where we decide a class other than the ith one. Continuing,

$$P_e = \sum_{i=1}^{Nc} P_i \left[1 - \int_{Z_i} f\left(\boldsymbol{X} \mid i\right) d\boldsymbol{X}\right] = 1 - \sum_{i=1}^{Nc} P_i \int_{Z_i} f\left(\boldsymbol{X} \mid i\right) d\boldsymbol{X} \tag{2.4}$$

The regions Z_i and Z_i^C are disjoint. Interchanging the summation and integral in the right hand side of (2.4), we get,

$$1 - \int_S P_{X(i)} f\left(\boldsymbol{X} \mid i(\boldsymbol{X})\right) d\boldsymbol{X} = P_e \tag{2.5}$$

Minimize P_e by maximizing the integral, which is done by maximizing the integrand, given a value of **X**. In other words, given **X**, evaluate the scalar numbers g(i) = P_if(**X**|i) for $1 \le i \le N_c$. Choose i such that g(i) is maximum. This value of i is $i_c'(p)$. In order to construct this form of Bayes discriminant, we need to estimate P_i and f(**X**|i).

2.1.3 Common forms of Bayes discriminant

We decide the class number of a pattern based on the value of the discriminant function that reduces the classification error (P_e). Three well known types of Bayes discriminant are given as follows.

(B_1) g(i) = P_i f (**X**|i): Find the maximum value of discriminant to minimize P_e

(B_2) $d(i)$ = func (P_i f ($\mathbf{X}$|i)): Find the maximum value, of discriminant to minimize P_e, if func () is an increasing function or

(B_3) $h(i) = P(i|\mathbf{X})$: Find the maximum value of discriminant to minimize P_e

2.2 Nearest Neighbor Classifiers

In practical pattern recognition applications, the nearest neighbor classifier (NNC) is often applied because it does not require an a priori knowledge of the joint probability density of the input feature vectors. Among the various methods of supervised statistical pattern recognition, the nearest neighbor rule achieves consistently high performance. It involves a training set with patterns from each class. A new sample is classified by calculating the distance to the nearest training case. Fig 2.1 gives a diagrammatic representation of a NNC.

2.2.1 Operation

(1) For the ith class we will have K_i clusters, and each cluster will have mean or center vectors $\mathbf{m_{ik}}$ where $1 \leq k \leq K_i$.

(2) Now, for a given test vector $\mathbf{X}$, find i such that, the distance between the test vector and the mean vector, $d(\mathbf{X},\mathbf{m_{ik}})$, is minimum. This value of i is the estimated class i_c'.

(3) The classification is correct if $i_c' = i_c$.

The distance of the input vector $\mathbf{X}$ from the mean vector, $\mathbf{m_{ik}}$, of kth cluster for ith class is given as

$$d\left(\boldsymbol{X},\boldsymbol{m}_{ik}\right)=\sum_{n=1}^{N}\left(X(n)-m_{ik}(n)\right)^2 \tag{2.6}$$

for the Euclidean distance and

$$d\left(\boldsymbol{X},\boldsymbol{m}_{ik}\right)=\sum_{n=1}^{N}\sum_{m=1}^{N}\left(X(n)-m_{ik}(n)\right)\cdot a_i(n,m)\cdot\left(X(m)-m_{ik}(m)\right) \tag{2.7}$$

for the Mahalanobis distance, where $a_i(n,m)$ is the element of the inverse covariance matrix ($\mathbf{A_i}$) for the ith class. The covariance matrix (C_i) element $c_i(n,m)$ is calculated as,

$$c_i(n,m)=\frac{1}{N_v(i)}\sum_{p:i_c(p)=i}^{N_v(i)}\left(X(n)-m_i(n)\right)\cdot\left(X(m)-m_i(m)\right) \tag{2.8}$$

for $1\le n \le N$ and $1\le m \le N$. The inverse of $\mathbf{C_i}$ is denoted by $\mathbf{A_i}$.

2.2.2 Design methods

Three general approaches for obtaining the center vectors $\mathbf{m}_{ik}$ are described here.

(1) Use all example data to form center vectors $\mathbf{m}_{ik}$.

(2) Cluster example or training data $\mathbf{X}_p$ separately for each class, to form center vectors $\mathbf{m}_{ik}$, where p denotes the pattern number such that, $1\le p \le N_v$.

(3) Cluster all $\mathbf{X}_p$ together to get center vectors $\mathbf{m}_k$. Then assign a class number to each $\mathbf{m}_k$ as follows. For each cluster mean or center vector $\mathbf{m}_k$, look at class numbers of $\mathbf{X}_p$ closest to $\mathbf{m}_k$. Use the plurality rule to assign the class numbers $i_c(p)$.

2.2.3 Theory

As the number of example vectors and the cluster numbers are increased, the error probability of the NNC comes closer to the error probability of Bayesian classifier

[10]. However, at the same time the computational complexity of the NNC increases. Also, for a small number of example vectors, training of the NNC is not optimal since clustering algorithms are used to produce the required center vector.

Theorem 1: For a NNC, as the number of clusters, K, approaches infinity we have,

$$P_{eB} \le P_{e(NNC)} \le 2 \cdot P_{eB} \tag{2.9}$$

where $P_{e(NNC)}$ and P_{eB} respectively denote the NNC and Bayes probabilities of error. For a proof of Theorem 1 refer to [44].

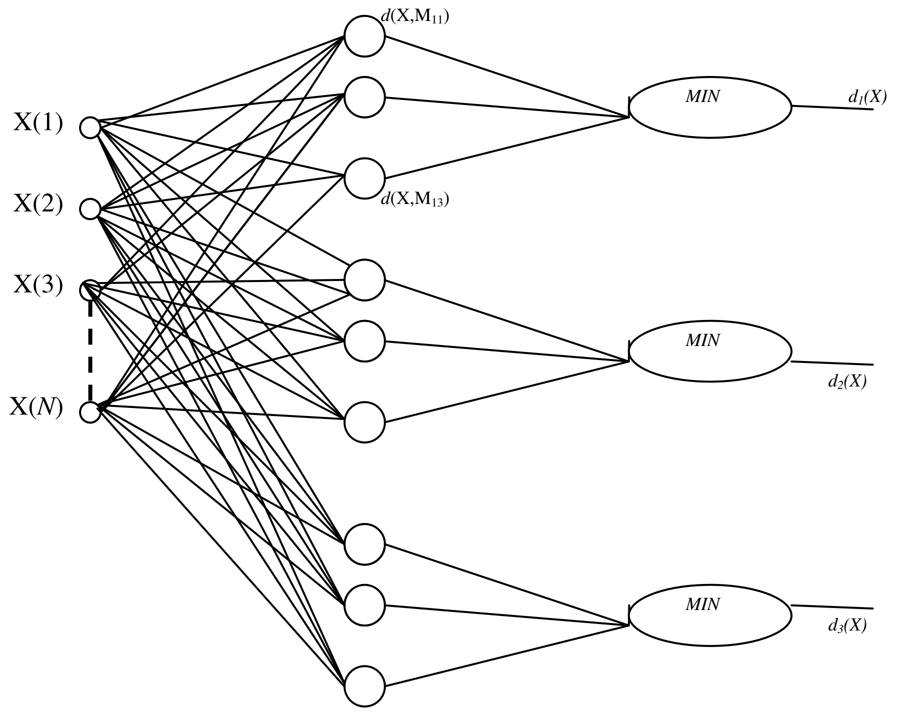

Figure 2.1 Nearest Neighborhood Classifier

The k nearest neighbor classifier (k-NNC) is another version of NNC with some different characteristics and features. Here k is the number of nearest training vectors or samples to the test sample **X**. An interesting theorem on the k-NNC is presented below.

Theorem 2 [10]: As k and (N_v/k) approach infinity, the k-NNC can be viewed as an attempt to estimate the a-posteriori probabilities from the training samples. Under this condition, k-NNC hence becomes optimal and

$$\lim_{k \to \infty} P_{e(k-NNC)} = P_{eB} \tag{2.10}$$

As k increases, the probability of error gets closer to the lower bound – the Bayes rate. In the limit as k goes to infinity, the two bounds meet and the k-NNC becomes optimal. We want to use a large value of k to obtain a reliable estimate. On the other hand, we want all of the k neighbors to be very near to the test sample, **X**. This forces us to choose a compromise k that is a small fraction of the total number of training samples, N_v. It is only in the limit as N_v goes to infinity that we can be assured of the nearly optimal behavior of the k-NNC.

2.3 Piecewise Linear Network Classifier

2.3.1 Operation

A piecewise linear network classifier (PLNC) divides the given data into K different clusters and then processes each cluster as a separate linear network. Piecewise linear networks (PLNs) have long been used for function approximation and classification tasks [53,54] where speed of operation and simplicity are very important. One design approach is training an MLP having piecewise linear activations [55,56].

The network structure is shown in fig. 2.2. The PLNC consists of three layers with input elements in the first layer, the hidden units in the second and the output units in the third. The feature vector elements are first normalized as

$$\mathbf{X}_{pn} \leftarrow \frac{\mathbf{X}_{pn} - \mu_n}{\sigma_n} \text{ , for } 1 \leq \mathrm{p} \leq \mathrm{N_v} \text{ and } 1 \leq \mathrm{n} \leq \mathrm{N} \tag{2.11}$$

where the means and standard deviations of the feature vector elements $\mathbf{X_p}$ are respectively μ_n and σ_n, where $1 \leq n \leq N$. This N-dimensional vector $\mathbf{X_p}$ forms the input to the PLNC. The hidden layer consists of K clusters, each cluster having its N-dimensional cluster mean vector $\mathbf{m_c}$, where $1 \leq c \leq K$. Given an input vector $\mathbf{X_p}$, we find c such that $d(\mathbf{X_p}, \mathbf{m}_c)$ is minimized. The normalized feature vector is then augmented as

$$\mathbf{X}_{pa} \leftarrow \left(\mathbf{X}_p^T : 1\right)^T \tag{2.12}$$

to form the (N+1) dimension vector, $\mathbf{X_{pa}}$, to the PLNC. Each cluster also has a weight matrix $\mathbf{A}_c$ of dimension N_c by (N+1), where N_c is the number of classes in the classification problem.

The output discriminant vector of the network, $\mathbf{y_p}$ has N_c elements. The vector $\mathbf{y_p}$ is calculated by multiplying the input vector with the weight matrix of the cluster it has been assigned to. Thus we form the output vector, $\mathbf{y_p}$, as

$$\boldsymbol{y_p} = \boldsymbol{A_c} \cdot \boldsymbol{X_{pa}} \tag{2.13}$$

The estimate of the correct class i_c is given by,

$$i_c{}'(p) = \arg\max\left[y_{pi}\right]_i \tag{2.14}$$

where y_{pi} is the i^{th} element of the output vector $\mathbf{y_p}$ and $1 \le i \le N_c$.

2.3.2 Design

A classification problem typically involves a feature space with numerous feature vectors or samples that have to be classified into various class labels. In supervised learning, the training dataset includes the feature vector $\mathbf{X_p}$ and the class label, $i_c(p)$, for each of the N_v feature vectors. The label is transformed into an N_c-dimensional target vector $\mathbf{t_p}$ such that

$$t_{pi} = \begin{cases} +b & i = i_c(p) \\ -b & \text{otherwise} \end{cases} \tag{2.15}$$

where $1 \le p \le N_v$, $1 \le i \le N_c$ and b is any positive integer. Before the network can be used for classification itself, it has to be trained. Training involves designing the PLNC weight matrices given numerous training patterns.

The process of training a PLNC is divided into two parts. The first part involves partitioning of the input feature space into K clusters. The second part involves the calculation of the network weights by solving a set of linear equations whose solution minimizes the MSE of the network,

$$E = \frac{1}{N_v} \sum_{k=1}^{K} \sum_{p:i_c(p)=k}^{N_v(k)} \left\| \boldsymbol{t}_p - \boldsymbol{A}_k \cdot \boldsymbol{X}_{pa} \right\|^2 \tag{2.16}$$

We can calculate the weights of the network $\mathbf{A_c}$ such that it minimizes the error in (2.16).

2.3.3 Theory

Now we consider the relationship between the PLNC and NNC. As K, the number of clusters approaches infinity, the convex Voronoi cells in the feature space get smaller in volume and the optimal decision boundaries in each cluster become linear. Hence each cluster can have its own linear discriminant and overall, a more complex decision boundary is achieved. Therefore, for a given value of K, the PLNC should perform better than the NNC. PLNC could be used over NNC where speed of operation and simplicity are very important as NNC converges slowly towards the Bayes probability of error. Fig 2.2 gives a diagrammatic representation of a piecewise linear network classifier.

Theorem 3: If a PLNC and a NNC have the same distance measure and identical cluster mean vectors, then as K approaches infinity,

$$P_{eB} \leq P_{e(PLNC)} \leq P_{e(NNC)} \leq 2 \cdot P_{eB} \tag{2.17}$$

where $P_{e(PLNC)}$, $P_{e(NNC)}$ and P_{eB} respectively denote the PLNC, NNC and Bayes probabilities of error.

For a proof of Theorem 3 refer to [44].

Theorem 4: As the cluster number (K) and the number of patterns belonging to that cluster ($N_v(c)$) approach infinity, the output of a PLNC approximates the a-posteriori probability functions of the class labels, given the input vector. Under this condition, the PLNC hence becomes optimal and

$$\lim_{K, N_v(c) \to \infty} \mathrm{P}_{\mathrm{e(PLNC)}} = \mathrm{P}_{\mathrm{eB}} \, . \tag{2.18}$$

For a proof of Theorem 4 refer to [44].

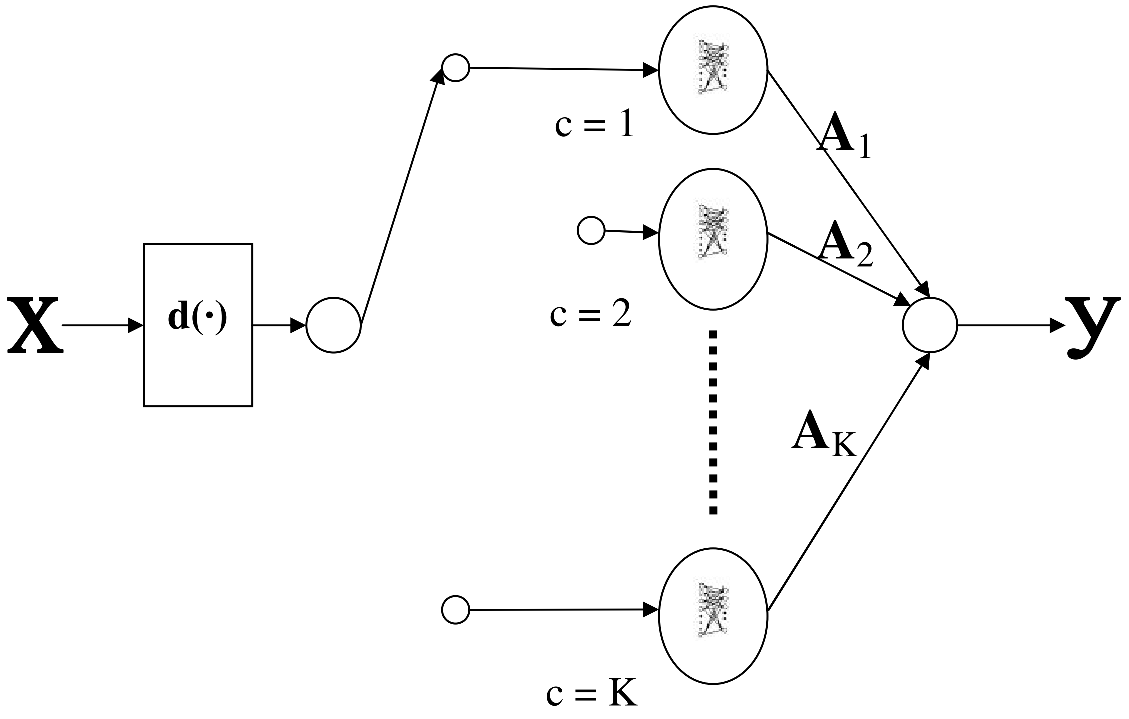

Figure 2.2 Piecewise Linear Network Classifier

2.4 Neural Network Classifiers

There are many types of neural network classifiers, including the multilayer perceptron (MLP) [37], radial basis function (RBF) network [35] and the support vector machine (SVM) [36]. In this section we investigate the operation, training and theory of a multilayer perceptron.

2.4.1 Operation

As the name suggests this classifier consists of multiple layers of functional units usually interconnected in a feed-forward way. In many applications the hidden units of these networks apply a sigmoid function as an activation function at the unit's output. The structure of the multi-layer perceptron (MLP) is shown in fig. 2.3. The output of a MLP can be computed as,

$$y(i) = \theta_o(i) + \sum_{k=1}^{N_h} w_{ho}(i,k) \cdot O(k) + \sum_{n=1}^{N} w_{io}(i,n) \cdot X(n) \tag{2.19}$$

where $\theta_o(i)$ is the threshold of the ith output, $w_{ho}(i,k)$ is the weight connected from the kth hidden unit to the ith output unit, $w_{io}(i,n)$ is the weight connected from the nth input unit to the ith output unit and O(k) is the output of the kth hidden unit with sigmoidal activation.

$$O(k) = \frac{1}{1 + e^{-NET(k)}} \qquad \text{for } 1 \le k \le N_c \tag{2.20}$$

$$NET(k) = \theta(k) + \sum_{n=1}^{N} w(k,n) \cdot X(n) \qquad \text{for } 1 \le k \le N_c \tag{2.21}$$

where θ(k) is the threshold of the kth hidden unit and w(k,n) is the weight connecting the nth input unit to the kth hidden unit.

Output weights are calculated and trained using any gradient technique, while the hidden weights are trained using back propagation [7, 23] technique. In back propagation, the computed output values are compared with the desired output to calculate the value of an error function. The error is then fed back through the network, which is then reduced by adjusting the weights by a general optimization technique like the conjugate gradient [24]. A multilayer perceptron with one hidden layer is shown in fig 2.3.

2.4.2 Training

First of all we initialize all the network weights with random numbers. Then we train these weights iteratively to get better performance of MLP. Training of MLP is divided into two parts, training of output weights and training of hidden weights. Let us look at each one separately.

(1) Training of output weights:

The error function can be expressed as

$$E = \sum_{i=1}^{N_c} E(i) \tag{2.22}$$

$$E(i) = \frac{1}{N_v(i)} \sum_{p:i_c(p)=i}^{N_v(i)} \left\| t_p(i) - y_p(i) \right\|^2 \quad \text{for } 1 \le \mathrm{i} \le \mathrm{N_c} \tag{2.23}$$

where N_v is the total number of patterns, $N_v(i)$ is the total number of patterns belonging to the ith class, $t_p(i)$ is the desired output for the ith class and $y_p(i)$ is the actual calculated output for the ith class and N_c is the total number of classes. Now let L, the number of basis functions, be $N+N_h+1$. The new basis functions are X(k) for k between 1 and N, X(N+k) = O(k) for k between 1 and N_h, and X(L)= 1. Taking the partial of error, for the

i^{th} output, with respect to the output weight and equating it to zero we get a set of linear equations as,

$$C_1(m,i) = \sum_{k=1}^{L} w_o\left(i,k\right) \cdot r\left(k,m\right) \qquad \text{for } 1 \le m \le L \text{ and } 1 \le i \le N_c \qquad (2.24)$$

where $w_o(i,k)$ is the weight from the k^{th} basis function to the i^{th} output. Also the autocorrelation and cross correlation elements are given as,

$$r\left(k,m\right) = \frac{1}{N_v} \sum_{p=1}^{N_v} X_p(k) \cdot X_p(m) \qquad \text{for } 1 \le k, m \le L$$

$$C_1(m,i) = \frac{1}{N_v} \sum_{p=1}^{N_v} t_p(i) \cdot X_p(m) \qquad \text{for } 1 \le m \le L \text{ and } 1 \le i \le N_c$$

From (2.24) we have L equations in L unknowns. Since the equations are often ill-conditioned, meaning that the determinant of **R** is close to 0, it is often unsafe to use Gauss-Jordon elimination [58], so the SVD [57] or the conjugate gradient [10] approaches are better.

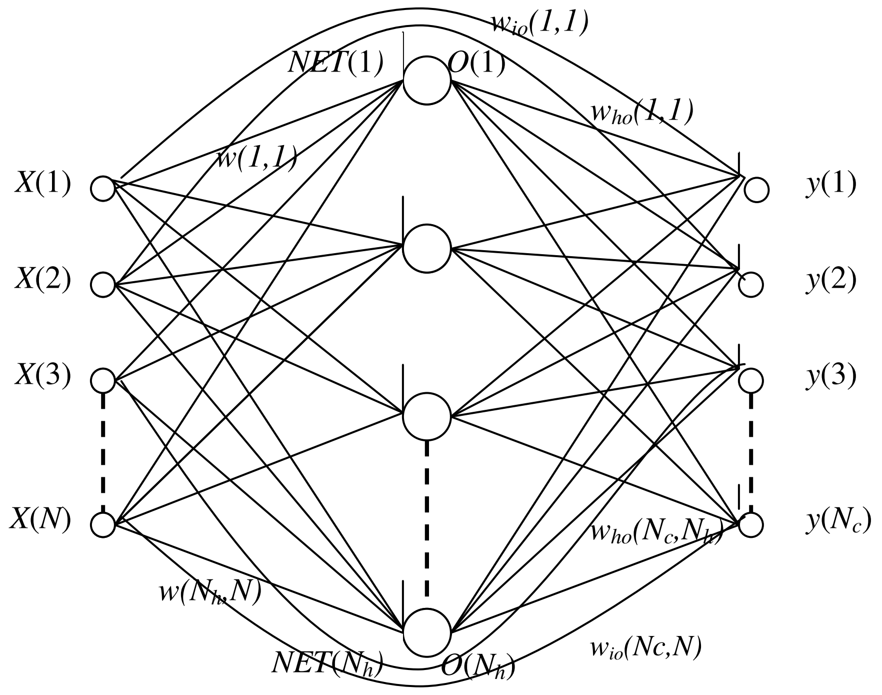

Figure 2.3 Structure of a Multi-Layer Perceptron

In the conjugate gradient approach the direction vector for the ith class is initialized to zero and is updated on every iteration as,

$$p(m) = -g(m) + B_2\, p(m) \text{ , for } 1 \le m \le L \tag{2.25}$$

where the gradient and the B_2 are calculated as,

$$g(m) = -2 \cdot C_1(i,m) + 2 \cdot \sum_{k=1}^{L} w_o(i,k) \cdot r(k,m) \text{ , for } 1 \le i \le N_c \text{ and } 1 \le m \le L \tag{2.26}$$

$$B_2 = \frac{\sum_{m=1}^{L} p(m)\left[\left(C_1(i,m)\right) - \sum_{k=1}^{L} w_o(i,k) \cdot r(k,m)\right)}{\sum_{m=1}^{N_c} \sum_{k=1}^{L} p(m) \cdot p(k) \cdot r(k,m)} \text{ , for } 1 \le i \le N_c \text{ and } 1 \le m \le L \tag{2.27}$$

Now the output weights are updated as,

$$w_o(i,k) = w_o(i,k) + B_2 \cdot p(k) \text{ , for } 1 \le k \le L \text{ and } 1 \le i \le N_c \tag{2.28}$$

(2) Training of hidden weights:

In the earlier step we just updated the weights from input to output and hidden unit to output. But the weights from input to hidden unit are still unchanged. So here we try to update these hidden weights such that it reduces the MSE. For training of hidden weights of MLP we generally use Output Weight Optimization-Back Propagation (OWP-BP) [10]. This technique is discussed at length in section 3.3.1.

2.4.3 Theory

MLPs are very popular and exhibit good performance in classification as well as estimation problems. Lets look at few established theorems on MLP.

Theorem 4: When neural net classifiers are trained to minimize the mean-squar error (MSE), the MSE approaches a constant value plus the expected squared error between the neural net output and Bayes discriminant, as the number of training patterns approaches infinity. Specifically,

$$\lim_{N_v \to \infty} \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{N_c} \left[t_p(i) - y_p(i) \right]^2 = \sum_{i=1}^{N_c} E\left[\left(h(i) - y(i) \right)^2 \right] + C \tag{2.28}$$

where C is a constant, independent of p, $t_p(i)$ is the i^th desired output for the p^th pattern, and $y_p(i)$ is the i^th output of the network. The Bayes discriminant h(i), is the probability that the i^th class is correct, given **X**, which is written as $P(i|\mathbf{X})$ and N_c is the total number of classes. For a proof of Theorem 4 refer to [43].

2.5 Support Vector Machines

Support vector machines (SVMs) [42] are a set of related supervised learning networks used for classification and regression. They usually consist of a feature extractor containing the Radial Basis Function (RBF) hidden units, followed by a classifier that makes decisions based on a linear combination of features. A special property of SVMs is that they minimize the empirical classification error and maximize the geometric margin between the various classes.

2.5.1 Operation

Let the dimension of feature space be h_{svm}, which is also the number of Support Vectors (SVs). Note that h_{svm} is equivalent to number of hidden units in a MLP. Figure 2.4 shows a diagram of an SVM.

Let $\left\{\mathbf{X}_{\mathrm{p}}, d_p\right\}_{p=1}^{N_v}$ be the training dataset. Let $\left\{\Phi_j(\mathbf{X})\right\}_{j=1}^{h_{svm}}$ denote a nonlinear transformation from input space to feature space. In our case, they represent the RBF.

$$\Phi_j(\mathbf{X}) = \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{X} - \mathbf{X}_j\right\|^2\right) \qquad (2.29)$$

for $1 \le j \le h_{svm}$. These $\Phi_j(\mathbf{X})$ are equivalent to the hidden unit activation functions in the MLP. The output of the SVM is given by

$$s_p = \sum_{j=1}^{h_{svm}} w_j \Phi_j\left(\mathbf{X}_p\right) + b \ \text{, for } 1 \le p \le N_v \qquad (2.30)$$

where w_j is the weight and b is the threshold of the hidden unit.

The main structural differences between MLPs and SVMs are

- Unlike MLPs, bypass weights are not present in case of SVMs (weights connecting inputs directly to outputs)
- There is only one output for SVMs whereas MLPs can handle multiple outputs.

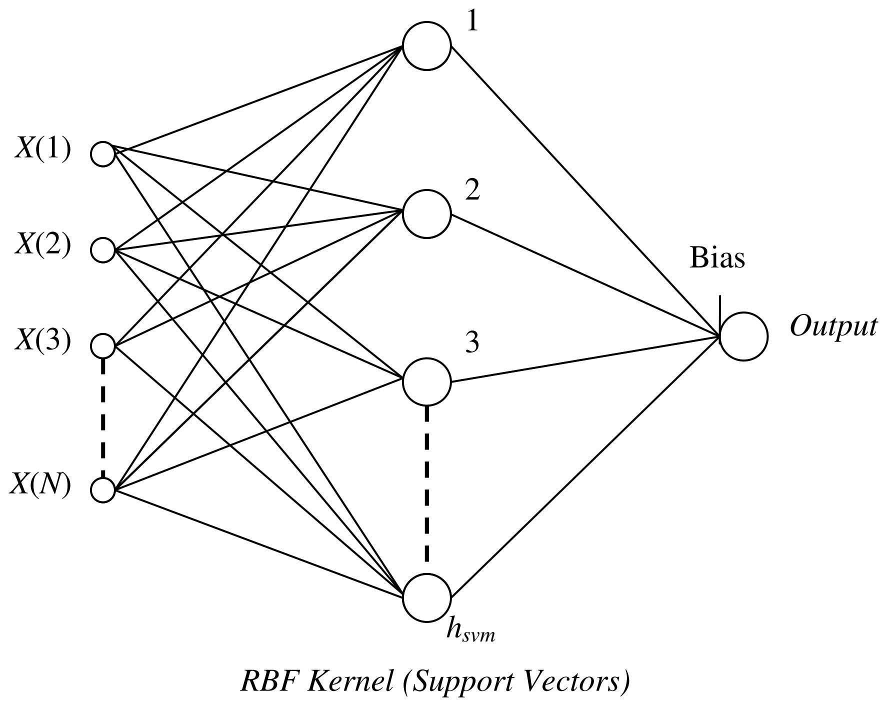

Figure 2.4 Support Vector Machine with RBF kernel.

2.5.2 Theory

A typical classification problem involves separating *N*-dimensional data into different classes by an *(N-1)*-dimensional hyperplane. This could be done using a typical form of linear classifier. Possible boundaries for such a classification problem are shown in fig. 2.5. However, if it is also desired to achieve maximum separation between the different classes, this could be obtained using an SVM or other maximal margin classifier [11].

By maximizing this margin, SVMs avoid overfitting. A maximum-margin hyperplane for an SVM trained with samples from two classes is shown in fig. 2.6. Samples along the hyperplanes are called the support vectors.

The parameters of the maximal margin hyperplane are commonly derived by solving a quadratic programming (QP) optimization problem using Platt's Sequential Minimal Optimization (SMO) algorithm [27,28]. This algorithm breaks the problem down into 2-dimensional sub-problems that may be solved analytically, eliminating the need for a numerical optimization algorithm such as the conjugate gradient method [29].

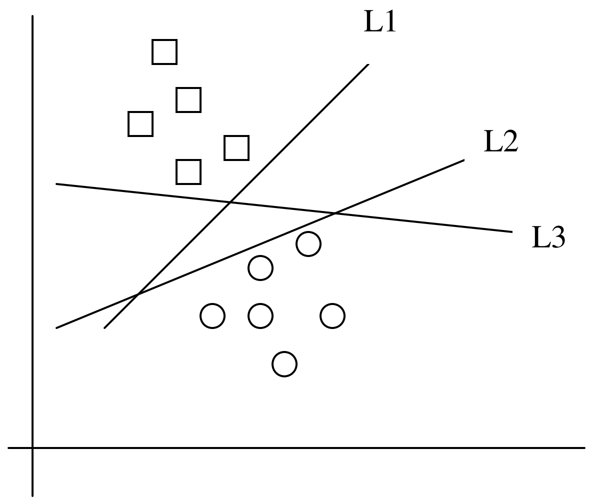

Figure 2.5 Possible boundaries to a simple classification problem

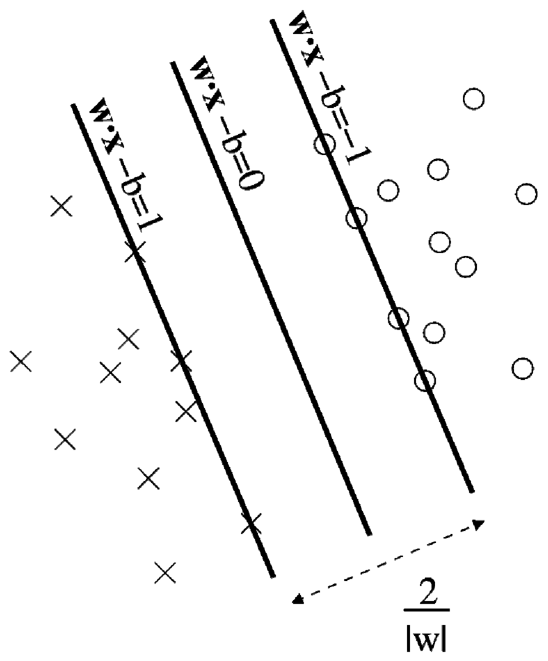

Figure 2.6 A maximum margin hyper plane for a support vector machine

2.6 Problems with classifiers

In Bayes Classifiers the required conditional probability densities are usually not available. Only approximations from parametric and non-parametric modeling approaches are available.

The k-Nearest Neighbor classifier is quite simple, but very computationally intensive to design. Even for simple classification problem it requires many distance calculations, which makes the classification a complex process. Theorems on convergence to Bayes error do exist for nearest neighbor classifiers (NNCs) and k-NNCs [10,11], which also have the advantage of being easy to design in a short period of time. However, the NNC and k-NNC are rarely used because they are very time-consuming to apply.

Neural net classifiers have several problems. Typical problems of the back propagation algorithm in MLP training are the speed of convergence and the possibility of ending up in a local minimum of the error function. Training time for MLP and RBF classifiers can be long and they may suffer from over fitting [10]. SVM classifiers avoid over fitting but usually require several orders of magnitude more hidden units than RBF and MLP networks. Also, MLP can suffer from memorization problems subjected to number of patterns provided during training. SVMs frequently require hundreds or thousands of parameters, and can take too long to apply. For satisfactory performance SVM's require large numbers of support vectors.

CHAPTER 3

BAYES GUASSIAN CLASSIFIER

The Bayes Gaussian Classifier (BGC) is a Bayes classifier where the conditional pdf f(**X**|i) is assumed to be Gaussian. Most of the data available in the real world is Gaussian because of the "Central Limit Theorem" [48], so this classifier is applicable in many real world applications.

3.1 Derivation of Bayes discriminant

The conditional probability density is given as,

$$f\left(\boldsymbol{X}\,|\,i\right)=\frac{1}{(2\pi)^{\frac{N}{2}}\,|\boldsymbol{C}_i|^{\frac{1}{2}}}\,e^{-\frac{1}{2}\left(\boldsymbol{X}-\boldsymbol{m}_i\right)^T A_i\left(\boldsymbol{X}-\boldsymbol{m}_i\right)} \qquad \text{for } 1\le i\le N_c \qquad (3.1)$$

where, $\mathbf{A_i}$ is the inverse covariance matrix and $\mathbf{C_i}$ is the covariance matrix calculated as,

$$C_i(n,m)=\frac{1}{N_v(i)}\sum_{p:i_c(p)=i}^{N_v(i)}\left(X_p(n)-m_i(n)\right)\left(X_p(m)-m_i(m)\right),\ \text{for } 1\le n\le N \text{ and } 1\le m\le N$$

Here $N_v(i)$ is the number of patterns belonging to the ith class and $\mathbf{m_i}$ is the mean input vector belonging to the ith class and is calculated as ,

$$m_i(n)=\frac{1}{N_v(i)}\sum_{p:i_c(p)=i}^{N_v(i)}X_p(n) \qquad \text{for } 1\le n\le N \qquad (3.2)$$

The type B_1 discriminant is calculated as,

$$g(i)=\frac{P_i\cdot\exp\left[-\frac{1}{2}\left(\boldsymbol{X}-\boldsymbol{m}_i\right)'\boldsymbol{A}_i\left(\boldsymbol{X}-\boldsymbol{m}_i\right)\right]}{(2\pi)^{\frac{N}{2}}\,|\boldsymbol{C}_i|^{\frac{1}{2}}} \qquad \text{for } 1\le i\le N_c \qquad (3.3)$$

where P_i is the probability for occurrence of the ith class.

Now, for the type B_2 discriminant, use d(i) = -2·ln (g(i)), which is a decreasing function, where g(i) is the type B_1 discriminant. We get

$$d(i) = \left(\boldsymbol{X} - \boldsymbol{m}_i\right)' \boldsymbol{A}_i \left(\boldsymbol{X} - \boldsymbol{m}_i\right) + B_i \qquad \text{for } 1 \le i \le N_c \tag{3.4}$$

Where the constant term B_i is given as,

$$B_i = N \ln(2\pi)^{\frac{N}{2}} + \ln\left(|\boldsymbol{C}_i|\right) - 2\ln\left(P_i\right) \qquad \text{for } 1 \le i \le N_c \tag{3.5}$$

For the type B_2 discriminant, we find the value of i such that d(i) is minimum. The resulting value of i is our estimate of i_c.

3.2 Limitations of BGC

One of the most important requirements for the BGC which may be considered as its limitation is that the distribution of the given data needs to be Gaussian. However, the covariance matrix is often singular (non-invertible), which causes problems during calculation of the inverse covariance matrix required in the discriminant function. The features might not be arranged in order of their importance. So, the error curve might not be monotonically non-increasing. Also the weights calculated by the statistical information of the data might not be exact.

3.3 Iterative improvement of the BGC

Various parameters calculated from the statistical information of the data might not be optimal as noted in section 3.2. So now we will try to modify few important parameters so that it improves the overall classifier performance. In this thesis we are modifying the constant term (B_i) in the discriminant function. We will be using the gradient and the back propagation technique to accomplish our goals.

There are various weights associated with the BGC, such as, mean vector, covariance matrix, inverse covariance matrix and constant term. Out of all these weights only the constant term is the one that is independent of all the classes. All other weights are function of a class. So the first step towards training was to train the constant term and try to modify it such that it improves the performance of the BGC.

3.3.1 Derivation and algorithm for the back propagation technique

Here we derive the equations for the gradient and the learning factor. These equations are required in the back propagation technique which is used to iteratively improve the constant term.

Converting the type B_2 discriminant, given in (3.3), to the type B_1 discriminant,

$$g(i) = e^{-\frac{1}{2}d(i)} \qquad \text{for } 1 \leq i \leq N_c \qquad (3.6)$$

where g(i) is the type B_1 discriminant and d(i) is the type B_2 discriminant. Converting the type B_1 discriminant to the type B_3 discriminant we get,

$$h(i) = \frac{g(i)}{\sum_{j=1}^{N_c} g(j)} \qquad \text{for } 1 \leq i \leq N_c \qquad (3.7)$$

where h(i) is the type B_3 discriminant. The mean square error between the desired and the estimated output is calculated as,

$$E = \frac{1}{N_v}\sum_{i=1}^{N_c}\sum_{p:i_c(p)=i}^{N_v(i)} \left\| t_p(i) - h_p(i) \right\|^2 \qquad (3.8)$$

where $h_p(i)$ is h(i) for the pth pattern and the ith class, and $t_p(i)$ is the ith desired output for the pth pattern, defined as

$t_p(i) = \delta(i-i_c)$ for $1 \le i \le N_c$ (3.10)

where i_c is the correct class number obtained from the training data file. Now we want to get the value of B_i such that it gives the minimum error. Thus we need to find the gradient of error with respect to B_i

$$\therefore \frac{\partial E}{\partial B_i} = -2 \sum_{p:i_c(p)=i}^{N_v(i)} \sum_{i=1}^{N_c} \left[t_p(i) - h_p(i)\right] \cdot \frac{\partial h_p(i)}{\partial B_i} \qquad \text{for } 1 \le i \le N_c \qquad (3.11)$$

Also, after few mathematical steps we calculated $\frac{\partial h_p(i)}{\partial B_i}$ as,

$$\frac{\partial h_p(i)}{\partial B_i} = \frac{\frac{1}{2} g(i) \sum_{j=0, j \ne i}^{N_c} g(j)}{\left[\sum_{j=1}^{N_c} g(j)\right]^2} \qquad \text{for } 1 \le i \le N_c \qquad (3.12)$$

Algorithmic description of OWO-BP:

1. Initialize network weights, $P_{old} = 10^{20}$, Z= 0.01
2. Calculate P_e. If $P_e < P_{old}$, replace P_{old} with P_e and save the constant term B_i. Also, increment Z by 10%.
3. If $P_e \ge P_{old}$, read back old B_i and do not save the currently calculated B_i. Also decrement Z and Z_1 by 10%. Skip the next step of updating the Z_1 and go to 5.
4. Calculate Z_1 as, $Z_1 = \frac{Z \cdot P_e}{\sum_{i=1}^{N_c} \left(\frac{\partial E}{\partial Bi}\right)^2}$ (3.13)
5. Update B_i as, $B_i = B_i + (Z_1 \cdot \frac{\partial E}{\partial B_i})$ (3.14)

6. Go to 2.

3.3.2 Performance analysis

Here we show the plot of error percentage versus iteration number, $N_{it,}$ for the BGC. This curve shows how training of constant term improves the classifier performance at each iteration. It takes into account all the features in its natural order, as given in training data file.

(1) GRNG: This file is a geometric shape recognition data file consists of four geometric shapes, ellipse, triangle, quadrilateral, and pentagon. Each shape consists of a matrix of size 64x64. For each shape, 200 triangle patterns were generated using different degrees of deformation. The deformations include rotation, scaling, translation, and oblique distortion. The feature set is ring-wedge energy (RNG), and it has 16 features [33].

Figure 3.1 Training error percentage versus iteration number, N_{it} for grng

The plot in fig 3.1 is for the grng training data which has 16 inputs and 4 classes. It shows percentage error for varying number of iterations. The curve indicates that there is improvement in the error percentage in most of the iterations. At the end of 25 iterations, the final error percentage comes out to be 2.5%, which is 50% of the initial error.

(2) GONGTRN: The raw data consists of images from hand printed numerals collected from 3,000 people by the Internal Revenue Service. We randomly chose 300 characters

from each class to generate 3,000 character training data. Images are 32 by 24 binary matrices. An image scaling algorithm is used to remove size variation in characters. The feature set contains 16 elements. The 10 classes correspond to 10 Arabic numeral [37].

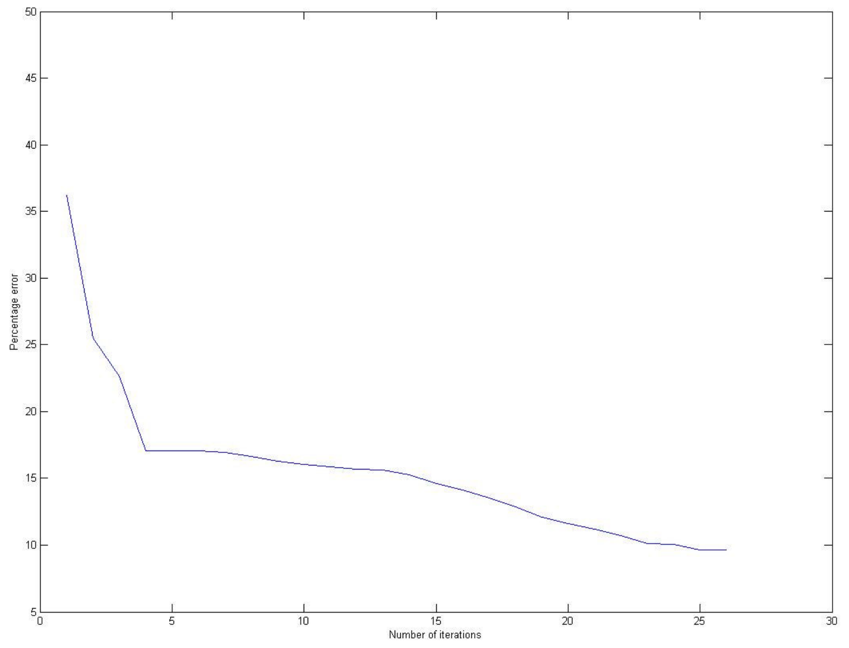

Figure 3.2 Training error percentage versus iteration number, N_{it} for gongtrn

The plot in fig 3.2 is for the gongtrn training data which has 16 inputs and 10 classes. It shows percentage error for varying number of iterations. The curve indicates that there is improvement in the error percentage in most of the iterations. At the end of 25 iterations, the final error percentage comes out to be 9.27% which is 25% of the initial error. There is more improvement on this training file as compared to the grng file.

(3) COMF18: This training data file is generated segmented images. Each segmented region is separately histogram equalized to 20 levels. Then the join probability density of pairs of pixels separated by a given distance and a given direction is estimated. We use 0, 90, 180 and 270 degrees for the directions and 1, 3, and 5 pixels for the separations. The density estimates are computed for each classification window. For each separation, the co-occurrences for the four directions are folded together to form a triangular matrix. From each of the resulting three matrices, six features are computed: angular second moment, contrast, entropy, correlation, and the sums of the main diagonal and the first off diagonal. This results in 18 features for each classification window [38].

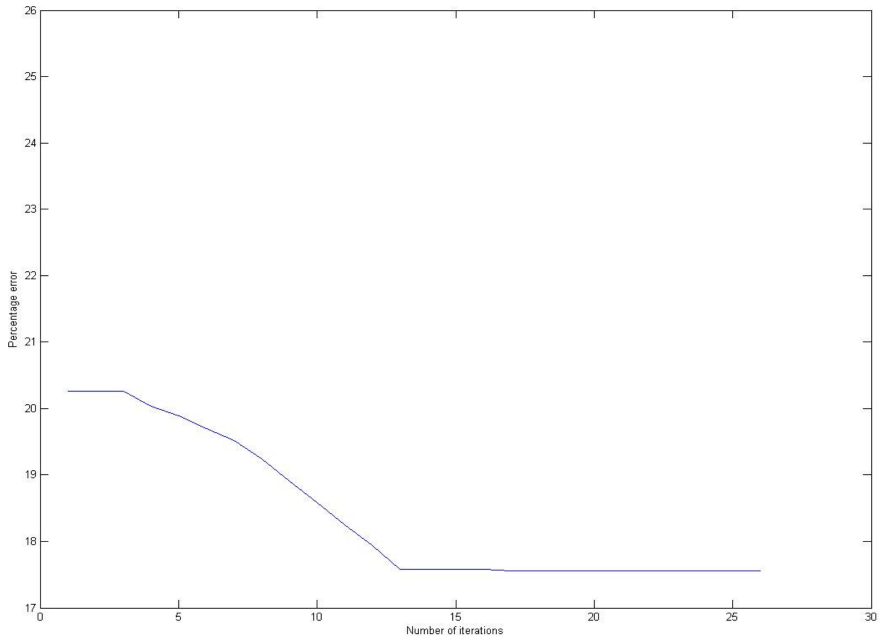

Figure 3.3 Training error percentage versus iteration number, N_{it} for comf18

The plot in fig 3.3 is for the comf18 training data which has 18 inputs and 4 classes. It shows percentage error for varying number of iterations. The curve indicates that there is improvement in the error percentage at every iteration. At the end of 25 iterations, the final error percentage comes out to be 5.4% which is 80% of the initial error.

CHAPTER 4

FEATURE SELECTION

The features in their natural order might not be arranged according to their contribution in reducing the Mean Square Error (MSE). This might result in the error curve not decreasing fast enough. This says that, some of the important features are not in the best position in the feature vector. Due to this problem Structural Risk Minimization [46] might not result in its best performance. To get the best result with the smallest possible classifier structure, we need to arrange the features in order of their contribution to minimize the MSE.

4.1 Feature selection using autocorrelation function

Now, we want to develop a feature selection technique, to reduce the MSE, using the autocorrelation function of the input vectors. Thus we would order the features such that the performance is better for smaller classifiers. MSE to be reduced is given by,

$$E = \frac{1}{N_v} \sum_{i=1}^{N_c} \sum_{p:i_c(p)=i}^{Nv(i)} \left\| t_p(i) - y_p(i) \right\|^2 \tag{4.1}$$

where N_c is the total number of classes, $N_v(i)$ is the total number of patterns belonging to ith class, $t_p(i)$ is the desired output for the ith class defined by (3.10) and $y_p(i)$ is the estimated output for the ith class, calculated as,

$$y_p(i) = \sum_{n=0}^{N-1} w(i,n)\, X_p(n) \text{ , for } 1 \le i \le N_c \tag{4.2}$$

where w(i,n) is the output weight and $\mathbf{X}_p$ is the input feature vector. Now to get the network weights we will calculate the gradient of error with respect to the weights and equate it to zero.

$$\frac{\partial E}{\partial w(i,m)} = -2\frac{1}{N_v(i)}\sum_{p:i_c(p)=i}^{N_v(i)}\left[t_p(i) - \sum_{n=1}^{N} w(i,n)\left(X_p(n)^i - m_i(n)\right)\right]\left(X_p(m)^i - m_i(m)\right)$$

(4.3)

for $1 \le i \le N_c$ and $1 \le m \le N$. Simplifying (4.3) further and equating it to zero we get,

$$\boldsymbol{C}_1 = \boldsymbol{R} \cdot \boldsymbol{W}^T \qquad (4.4)$$

where **R** is the auto-correlation matrix defined as,

$$r(k,i) = \frac{1}{N_v}\sum_{p=1}^{N_v} X_p(k)\,X_p(i) \quad \text{for } 1 \le k \le N \text{ and } 1 \le i \le N \qquad (4.5)$$

and $\mathbf{C_1}$ is the cross-correlation matrix defined as,

$$C_1(k,i) = \frac{1}{N_v}\sum_{p=1}^{N_v} t_p(k)\,X_p(i) \quad \text{for } 1 \le i \le N \text{ and } 1 \le k \le N_c \qquad (4.6)$$

The expression in (4.4) represents the sets of linear equations which we will solve to get the weights of the network.

Now, in Gram-Schmidt procedure we order the inputs according to their usefulness. J is the order vector which will store the index of the inputs in order of their contribution to reduce MSE.

$J_i = i$, for $0 \le i \le N$. The mth orthonormal basis function X'_m can be given as:

$$X'(m) = \sum_{i=0}^{m} a(m,i) \cdot X\left(J_i\right) \qquad \text{for } 1 \le m \le N \qquad (4.7)$$

where a(m,i) is the element of the orthonormalizing triangular matrix **A.** The first basis function can be defined as,

$$X'(0) = a\left(0,0\right) X\left(J_0\right) \tag{4.8}$$

Here a(0,0) is being calculated as,

$$a\left(0,0\right) = \frac{1}{\| X(0) \|} = \frac{1}{r\left(0,0\right)^{\frac{1}{2}}} \tag{4.9}$$

Now, for $1 \leq i \leq N-1$, perform the following operations

$$c\left(j\right) = \sum_{k=0}^{j} a\left(j,k\right) r\left(J_k, J_i\right) \qquad \text{for} \qquad 0 \leq j \leq i-1 \tag{4.10}$$

$$b(i) = 1 \tag{4.11}$$

$$b\left(j\right) = -\sum_{k=j}^{i-1} c(k)\, a\left(k,j\right) \qquad \text{for} \qquad 0 \leq j \leq i-1 \tag{4.12}$$

$$a\left(i,k\right) = b\frac{k}{\left[r\left(J_i, J_i\right) - \sum_{l=0}^{i-1} c(l)^2\right]^{\frac{1}{2}}} \qquad \text{for} \qquad 0 \leq k \leq i \tag{4.13}$$

$$w'\left(k,m\right) = \sum_{i=0}^{m} a(m,i)\, C_1\left(k,J_i\right) \qquad \text{for} \qquad 0 \leq k \leq i \tag{4.14}$$

4.1.1 Calculation of Mean Square Error (MSE)

When the basis function elements of **X** are transformed into **X'**, the system is mapped into new weights W'(k,i) for the kth estimated output $y_p(k)$ and ith orthonormal basis function X'(i) for given training dataset with N_v patterns.

$$y_p(k) = \sum_{i=0}^{N-1} w'\left(k,i\right) X'_p(i) \qquad \text{for} \quad 1 \le \text{p} \le \text{N}_\text{v} \qquad (4.15)$$

The MSE in terms of the new weights for N_v desired values of t(k) can be written as

$$E(k) = \frac{1}{N_v}(k) \sum_{p:i_c(p)=k}^{N_v(k)} \left[t_p(k) - \sum_{i=0}^{N-1} w'\left(k,i\right) X'_p(i) \right]^2 \qquad \text{for } 1 \le \text{k} \le \text{N}_\text{c} \qquad (4.16)$$

$$E = \sum_{k=1}^{N_c} E(k) = \sum_{k=1}^{N_c} E\left[t(k)^2\right] - \sum_{k=1}^{N_c} \sum_{i=0}^{N-1} \left[w'\left(k,i\right)\right]^2 = \quad (\text{e}_1) - (\text{e}_2) \qquad (4.17)$$

where, e_1 referred as the energy of the system is given by,

$$\text{e}_1 = \sum_{k=1}^{N_c} E\left[t(k)^2\right] = \frac{1}{N_v} \sum_{k=1}^{N_c} \sum_{p=1}^{Nv} \left[t_p(k) \cdot t_p(k)\right] \qquad (4.18)$$

and, e_2 referred as the power function is given by,

$$\text{e}_2 = \sum_{k=1}^{N_c} P(k) = \sum_{k=1}^{N_c} \sum_{i=1}^{N-1} \left[w'\left(k,i\right)\right]^2 \qquad (4.19)$$

The desired new order of basis functions, J, that reduces the MSE is thus obtained by maximum value of P(k) and is given by,

$P_{J(0)} \ge P_{J(1)} \ge P_{J(2)} \ge \ldots\ldots P_{J(N-1)}$

4.1.2 Detecting the linearly dependent basis functions

If any element of the matrix **A** (a(m,i)) is very large or close to infinity, it means that the orthonormal projection of that corresponding basis function will be zero. Hence we can conclude that that basis function does not contain any unique and useful information. We can derive that basis function by any linear combination of other basis functions.

4.2 Feature selection using the covariance function

Here we would like to bring some variations in the Gram-Schmidt procedure discussed in 4.1, such that it improves the performance of the classifier. For this reason, we tried using the covariance information of the input vectors.

4.2.1 Generating single order vector for all the classes

Our aim in this section is to come up with a single order vector, J, at the end of Schmidt procedure. Still, we would like to check for linear dependency of a feature on each and every class. If a feature is found linearly dependent, then instead of throwing it away, we would make its contribution towards the calculation of power term zero. This means we would make the corresponding row of orthonormalizing matrix, **A**, to be zero. By this we are not throwing away a feature if it is detected linearly dependent on any of the class. The idea behind this is that a feature found dependent for one class may be useful for other class. So we would save the information in that feature and use it for all other classes.

Now, let E(i) be the MSE for the ith class, which could be written as,

$$E(i) = \frac{1}{N_v(i)} \sum_{p:i_c(p)=i}^{N_v(i)} \left[t_p(i) - \sum_{n=1}^{N} w(i,n)\left(X_p(n)^i - m^i(n) \right) \right]^2 \quad \text{for } 1 \le \text{i} \le \text{N}_\text{c} \qquad (4.20)$$

where w(i,n) is the element of the weight matrix (W) connecting the nth input to the ith output. In minimizing E(i) with respect to the weights, we need the gradient, which is written as,

$$\frac{\partial E}{\partial w(i,m)} = -2\frac{1}{N_v(i)}\sum_{p:i_c(p)=i}^{N_v(i)}\left[t_p(i) - \sum_{n=1}^{N} w(i,n)\left(X_p(n)^i - m_i(n)\right)\right]\left(X_p(m)^i - m_i(m)\right)$$

(4.21)

$$\frac{1}{N_v(i)}\sum_{p:i_c(p)=i}^{N_v(i)} t_p(i)\left(X_p(m)^i - m^i(m)\right) = \sum_{n=1}^{N} w(i,n)\frac{1}{N_v(i)}\left(X_p(n)^i - m^i(n)\right)\left(X_p(m)^i - m^i(m)\right)\,]$$

(4.22)

Therefore,

$$\boldsymbol{C}_2 = \boldsymbol{C}_i \cdot \boldsymbol{W}^{iT} \tag{4.23}$$

The expression in 4.23 represents a set of linear equations for the ith class, which needs to be solved to get the minimum MSE. This is similar to the equations in 4.4, only the autocorrelation matrix is replaced by the class-covariance matrix. The equation in 4.23 could be also written as,

$$\frac{N_v(i)}{N_v}\mathbf{m}_i = \boldsymbol{C}_i \boldsymbol{W}^{iT} \tag{4.24}$$

where the cross correlation vector for the ith class, is given by $\frac{N_v(i)}{N_v}\cdot \boldsymbol{m}(i)$

Thus, the new cross correlation matrix can be written as,

$$\therefore\ \mathbf{C_2} = [\frac{N_v(1)}{N_v}\cdot \boldsymbol{m}(1)\ ;\ \frac{N_v(2)}{N_v}\cdot \boldsymbol{m}(2)\ ;\ \ldots\ldots\ ;\ \frac{N_v\left(N_c\right)}{N_v}\cdot \boldsymbol{m}\left(N_c\right)] \tag{4.25}$$

Using the equations in (4.23), we solve for the weights and calculate the orthonormalizing **A** matrix which would ultimately be useful in generating the order vector, J, as discussed in section 4.1.

4.2.2 Generating separate order vector for each class

In this approach we are trying to detect linear dependency of each feature on each class separately. So by the end of this procedure we would have N_c number of separate order vectors, one for each class. This method is computationally very inefficient as it requires developing a separate classifier for each class. So the network becomes more complicated and requires lot more calculations as compared to the previously described methods.

We tried using this approach, but the algorithm got too complex during its implementation and the results were not satisfactory. So we haven't used this approach in our final software.

CHAPTER 5

PERFORMANCE COMPARISON

5.1 Error curves for feature selection using the autocorrelation approach

Here we show the plots for error percentage (% error) versus number of features, N, for various training data files where the autocorrelation method generates the order vector. Iterative improvement is also used with 23 iterations, after the classifiers are designed.

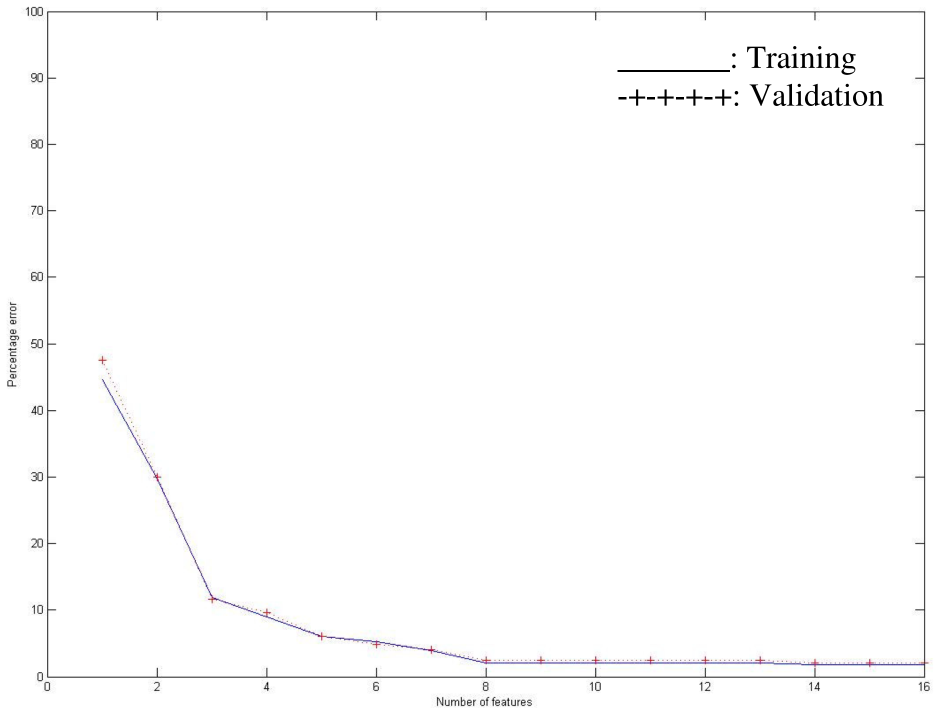

Figure 5.1 Error percentage versus N for grng using autocorrelation method.

The plot in fig 5.1 is for the grng data which has 16 inputs and 4 classes. Grng training and validation data file consists of 400 distinct patterns each. It shows the percentage error for varying number of features. The curve indicates that there is improvement in error percentage as the number of features increases and the error percentage reduces by a significant amount at the end when we use all the features. The training and validation curves are almost identical; this is because the training and testing data are very similar statistically.

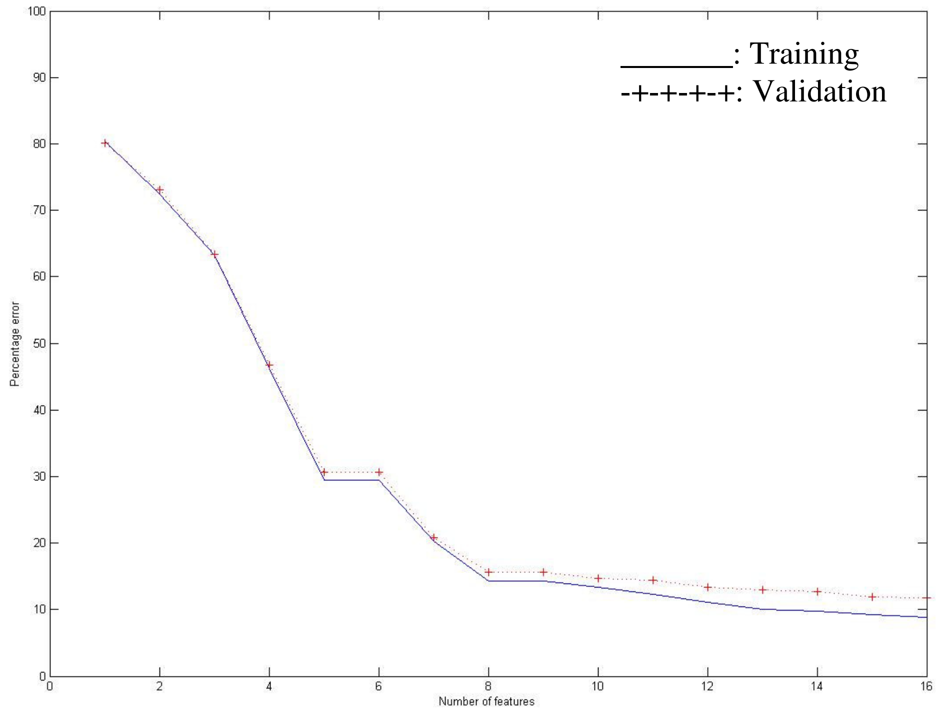

Figure 5.2 Error percentage versus N for gongtrn and autocorrelation method.

The plot of fig 5.2 is for the gongtrn and gongtst data which has 16 inputs and 10 classes. Training and validation data files consist of roughly 3100 patterns each. The plot shows the percentage error for varying numbers of features. The curve indicates that there is improvement in the error percentage as the number of features increases and the error percentage reduces by a significant amount at the end when we use all the features. The training and validation curves are mostly different at all points after the few initial features. Towards the end while using all the features, training does better than the validation.

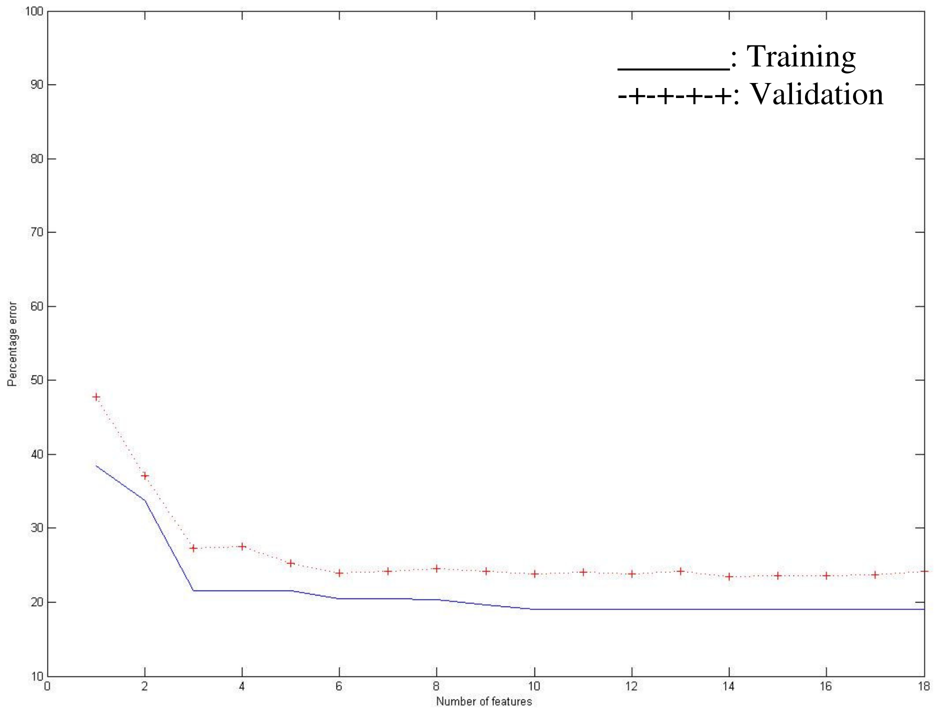

Figure 5.3 Error percentage versus N for comf18 and autocorrelation method.

The plot in fig 5.3 is for the comf18 training data which has 18 inputs and 4 classes. Training and validation data files consist of 1800 patterns each. The plot shows the percentage error for varying numbers of features. The curve indicates that there is improvement in the error percentage as the number of features increases and the error percentage reduces by a significant amount at the end where we use all the features. Here the training is better than testing at all times. Thus we can conclude that the classifier performs well on unseen data but not as good as it performs on the training data.

5.2 Error curves for feature selection using the covariance approach

Here we show the plots of error percentage (% error) versus N for various training data files where the covariance method is used. Iterative improvement is also used with 23 iterations, after each classifier is designed.

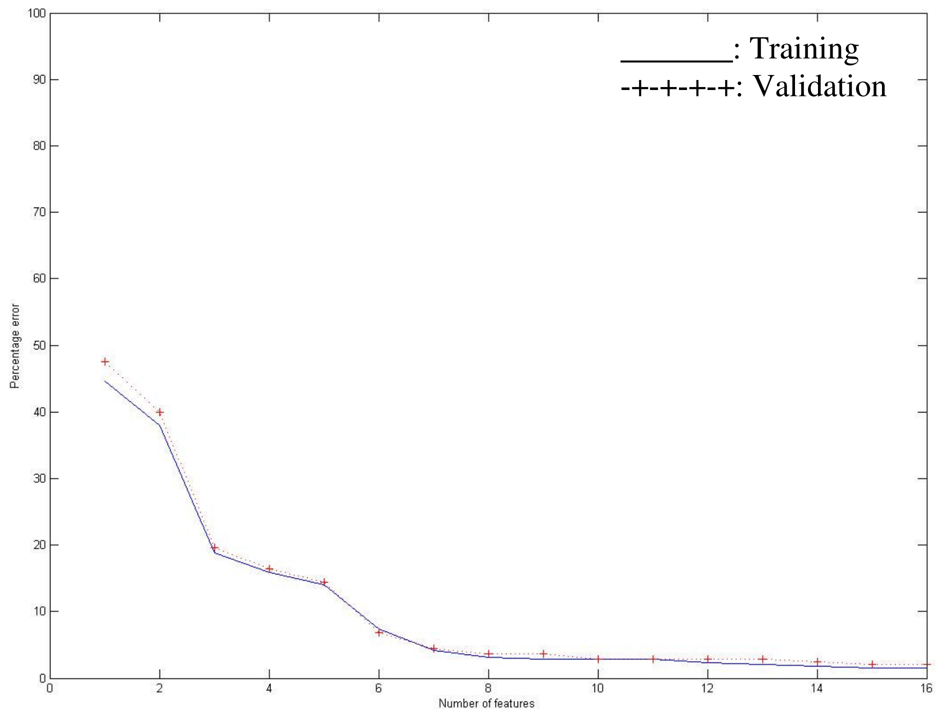

Figure 5.4 Error percentage versus N for grng using covariance method.

The plot in fig 5.4 is for the grng data which has 16 inputs and 4 classes. Training and validation data files consist of roughly 400 patterns each. It shows percentage error for varying number of features. The curve indicates that there is improvement in error percentage as the number of features increases and the error percentage reduces by significant amount at the end when we use all the features. Here the training is almost similar to testing at most of the points but at few points it does give different results than training. Also there is not much improvement in this method as compared to the previous method where we used auto-correlation elements for ordering. The autocorrelation approach performs better while using less number of features.

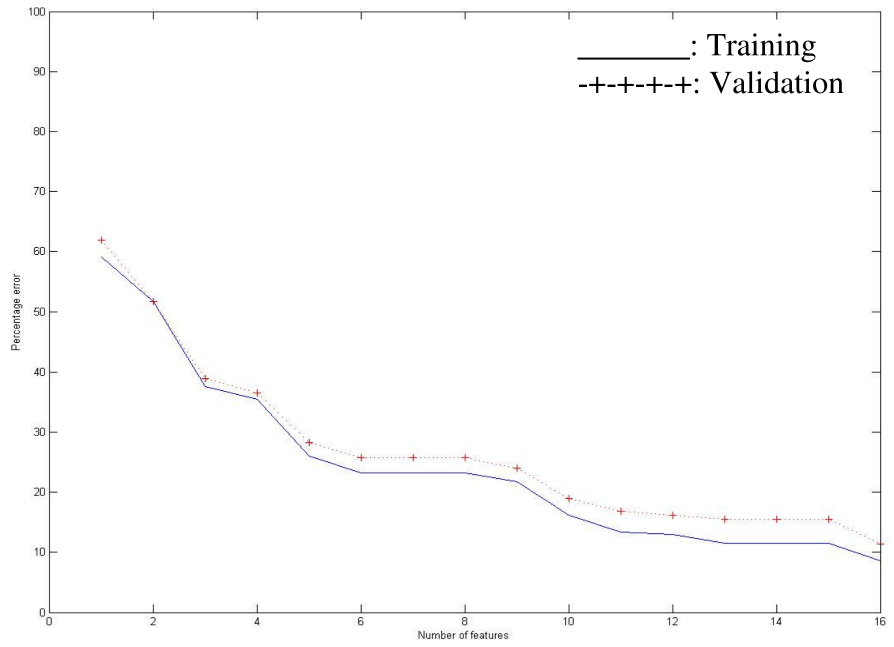

Figure 5.5 Error percentage versus N for gongtrn and gongtst using covariance method.

The plot in fig 5.5 is for the gongtrn and gongtst data which has 16 inputs and 10 classes. Training and validation data file consists of roughly 3100 patterns each. It shows the percentage error for varying number of features. The curve indicates that there is improvement in error percentage as the number of features increases and the error percentage reduces by a significant amount at the end when we use all the features. Here the training is better than testing at all time. This curve show that the classifier performs better when ordered using cross-correlation elements. Especially for less number of features this method gives better result when compared to autocorrelation approach (Figure 5.2). There is significant amount of improvement in this plot as compared to the one which used autocorrelation elements. For less number of features this method definitely gives better results as compared to the autocorrelation one.

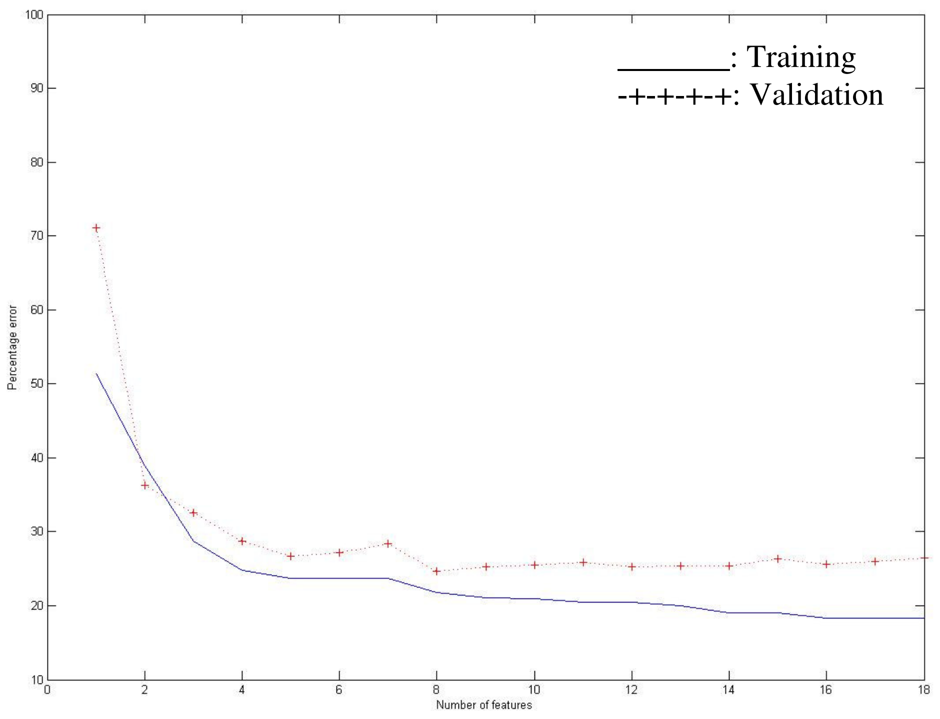

Figure 5.6 Error percentage versus N for comf18 using covariance method.

The plot in fig 5.6 is for the comf18 training data which has 18 inputs and 4 classes. Training and validation data file consists of roughly 1800 patterns each. It shows percentage error for varying number of features. The curve indicates that there is improvement in error percentage as the number of features increases and the error percentage reduces by significant amount at the end when we use all the features. Here the training is better than testing at all time. The training is not as good as it was in the case when we used autocorrelation elements for ordering (Figure 5.3). There is not much difference in the training error but the validation curve is definitely better as compared to the one which used autocorrelation elements.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis we discuss how to improve the BGC's performance by training the constant term in the discriminant function. A back propagation technique is applied to train the constant term such that the classification error is reduced. Also we discuss two methods for developing nested feature sets in the BGC, for the purpose of Structural Risk Minimization [46]. An important feature introduced in this thesis is development of nested feature subset instead of separate or different subsets. As the subsets are nested the error curve for error percentage versus number of features will be monotonically non-increasing which might not be the case for non-nested feature subset. In the first method we used the autocorrelation elements for ordering the features, a monotonically non-increasing curve for error percentage versus the number of features. In the second method, the idea is to use the individual covariance matrices to order the features and to check for feature dependencies in each class individually. This method also gave significant improvement in the error percentage versus N curve. It can be stated that the BGC usually gives faster and more efficient training than the NNC, which usually requires a lot more patterns and weights.

6.2 Future Work

Much work remains for improving the BGC.

(1) We can train other classifier coefficients to improve the classifier's performance.

(2) We can investigate methods for eliminating small elements in the inverse covariance matrices. We haven't included this part in our thesis because it is a heuristic task and there is not an obvious optimal method to do this.

(3) We could apply BRUTE FORCE method to reduce size of inverse covariance matrix.

    (a) Generate set of inverse covariance matrix elements to be candidates for removal.

    (b) Set them to zero.

    (c) Measure probability of error.

    (d) Save the classifier if it is best for its size (number of coefficients).

    This method is optimal but impractical due to combinatorial explosion.

APPENDIX A

GRAM SCHMIDT ALGORITHM

The Gram-Schmidt procedure transforms a set of linearly dependent vectors into an orthonormal basis vector. It is commonly used for various applications in the field of neural networks such as (a) To find optimum choice of Radial centers for RBF [32], (b) Fast computation of weights of RBF network [33], (c) Pruning of MLP [8], and (4) Feature selection in piecewise classifier [9]. A less know feature of Gram-Schmidt procedure is that it can order the basis functions in the order of their contribution to minimize the MSE. Such a network can be represented as a monotonically non-increasing function of sum of basis functions and achieve a faster rate of convergence. This method of orthonormalizing basis functions can be used to prune the less important basis functions. These desirable properties along with the effective representation, system re-transformation and a fast and distributed iterative solution make it a better candidate over other learning algorithms.

Normal Schmidt process requires one pass through training file to obtain each new basis function. Here we discuss a more useful form of Schmidt process, which will let us express the orthonormal system in terms of autocorrelation and covariance elements.

Consider a vector **X** (X_0, X_1, X_2… X_{N-1}) whose elements are basis functions and its corresponding orthonormal mapping is **X**' (X'_0, X'_1, X'_2… X'_{N-1}) where (**X** and **X**') $\in \Re^N$. By the definition of orthonormality, vector **X**' is orthonormal only if it satisfies the following condition:

$$< X'(i)\ X'(j) > \; = 0 \qquad \text{for} \quad i \neq j \tag{A.1}$$

$$= 1 \qquad \text{for} \quad i = j \qquad i,j \in (0, N-1) \tag{A.2}$$

where, X'(i) is the ith element of the vector **X**' and < X'(i) X'(j) > is defined as the inner product of X'(i) with X'(j).

$$<X'(i)\,X'(j)> = \frac{1}{N_v}\sum_{p=1}^{N_v} X'_p(i)\,X'_p(j) \qquad \text{(A.3)}$$

Here $X'_p(i)$ refers to the pth value of ith orthonormal function X'(i).

Also, $\left\| X(i) \right\|$ is defined as:

$$\| X(i) \| = \left[\frac{1}{N_v}\sum_{p=1}^{N_v} X_p(i)^2\right]^{\frac{1}{2}} \qquad \text{(A.4)}$$

Then by the standard Gram-Schmidt procedure, X'(k) is calculated as:

$$X'(0) = \frac{X(0)}{\| X(0) \|} \qquad \text{(A.5)}$$

$$X'(1) = \frac{X(1) - <X(1)\,X'(0)> X'(0)}{\| X(1) - <X(1)\,X'(0)> X'(0) \|} \qquad \text{(A.6)}$$

…

$$X'(k) = \frac{X(k) - \sum_{i=0}^{k-1} <X(k)\,X'(i)> X'(i)}{\| X(k) - \sum_{i=0}^{k-1} <X(k)\,X'(i)> X'(i) \|} \qquad \text{for } 1 \le k \le N-1 \qquad \text{(A.7)}$$

<u>A General approach towards Gram-Schmidt Procedure</u>

Due to round off errors in computer, Gram-Schmidt procedure is not always numerically stable. An iterative solution to it is given here. **A** represents a lower triangular N x N orthonormal transformation matrix such that:

X' = A • X (A.8)

Thus the m^th orthonormal function can be obtained from **X** and **A** by

$$X'(m) = \sum_{i=0}^{m} a(m,i)\, X(i) \qquad \text{for } 0 \le m \le N-1. \qquad (A.9)$$

Let **R** be the auto-correlation matrix, where its elements r(i,j) is defined as:

$$r(i,j) = \frac{1}{N_v} \sum_{p=1}^{N_v} X_p(i) \cdot X_p(j) \qquad \text{for } 0 \le i,j \le N-1 \qquad (A.10)$$

Then from (A.8 - A.11),

$$a(0,0) = \frac{1}{\left\| X(0) \right\|} = \frac{1}{r(0,0)^{\frac{1}{2}}} \qquad (A.11)$$

X'(0) = a(0,0)·X(0) (A.12)

$$X'(1) = a(1,0) \cdot X(1) + a(1,1) \cdot X(1) \qquad (A.13)$$

Let X'(1) be equal to

$$X'(1) = Z \frac{1}{\| Z(1) \|} \qquad (A.14)$$

Then Numerator of X'(1) is

Z(1) = X(1) – b(0)·X'(0) = X(1) – b(0)·a(0,0)·X(0) (A.15)

where b(0) = <X'(0)·X(1)> = a(0,0)·r(0,0) (A.16)

Writing Z(1) as

$$Z(1) = b(0)\, X(0) + b(1)\, X(1) \qquad (A.17)$$

Here, $b(0) = -\, c(0)\, a(0,0)$ (A.18)

$b(1) = 1$ (A.19)

Also, $\| Z(1) \| = \| < X(1) - c(0)\, X'(0)\, ,\, X(1) - c(0)\, X'(0) > \|$ (A.20)

$$\therefore \|Z(1)\| = \left[r(1,1) - c(0)^2\right]^{\frac{1}{2}} \tag{A.21}$$

Equating (A.14) and (A.20)

$$X'(1) = \left(a(1,0)\,X(1) + a(1,1)\cdot X(1)\right) = \frac{b(0)\cdot X(0) + b(1)\cdot X(1)}{\left[r(1,1) - c(0)^2\right]^{\frac{1}{2}}} \tag{A.22}$$

$$a(1,0) = \frac{-c(0)\,a(0,0)}{\left[r(1,1) - c(0)^2\right]^{\frac{1}{2}}} \tag{A.23}$$

$$a(1,1) = \frac{1}{\left[r(1,1) - c(0)^2\right]^{\frac{1}{2}}} \tag{A.24}$$

REFERENCES

[1] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall PTR, 1998.

[2] Saurabh Sureka, Orthonormal Functional Link Net , Thesis presentation, UTA, 2007.

[3] Christopher M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[4] Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley Pub, 1989.

[5] Michael T. Manry, Steven J. Apollo, and Qiang Yu, "Minimum Mean Square Estimation and Neural Networks," *Neurocomputing*, vol. 13, September 1996, pp. 59-74.

[6] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill Book Company, New York, 1965.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986, vol. 323, pp. 533-536.

[8] F. J. Maldonado, M. T. Manry, T. Kim, "Finding optimal neural network basis function subsets using the Schmidt procedure", *Proc. of IJCNN*, Vol. 1, 2003, pp. 444 - 449.

[9] J. Li, M. T. Manry, P. Narasimha, C. Yu, "Feature Selection Using a Piecewise Linear Network", *IEEE Trans. Neural Networks*, Vol 17, No. 5, 2006, pp.1101-1115.

[10] R. O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification*, 2nd Ed,Wiley Interscience, 2000.

[11] K. Fununaga, *Statistical Pattern Recognition*, 2nd Ed., Academic Press, NY, 1990.

[12] C.K.I. Williams and D. Barber, “Bayesian Classification with Gaussian Processes” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, 1998, pp. 1342-1351.

[13] D.W. Ruck, S. Rogers, M. Kabrisky, M. Oxley, B. Suter, "The multilayer perceptron as an approximation to a bayes optimal discriminant function", *IEEE Trans. Neural Networks*, 1990, pp. 296-298.

[14] R. G. Gore, J. Li, M. T. Manry, L. M. Liu, C. Yu and J. Wei, "Iterative Design of Neural Network Classifiers Through Regression", *Int. Journal Artificial Intelligence Tools*, Vol 14, Issues 1&2, 2005.

[15] H. C. Yau, M. T. Manry, "Iterative Improvement of a Nearest Neighbor Classifier", *Neural Networks*, Vol. 4, 1991, pp. 517-524.

[16] W. H. Delashmit and M. T. Manry, "Recent Developments in Multilayer Perceptron Neural Networks", *Proceedings of the 7th annual Memphis Area Engineering and Science Conference (MAESC)*, 2005.

[17] Nello Cristianini and John Shawe-Taylor, A*n Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, 2000.

[18] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers", *5th Annual ACM Workshop on COLT*, pp. 144-152, Pittsburgh, PA, ACM Press, 1992.

[19] T. Kohonen, *Self-Organization and Associative Memory*, 2nd ed., Springer-Verlag, 1987.

[20] Mohammed Kolahdouzan and Cyrus Shahabi, "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases," *Proceedings of the 30th Very Large Data Bases (VLDB) Conference*, Toronto, Canada, 2004.

[21] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE 78 (9)*, pp. 1484-1487, 1990.

[22] Thomas Bayes, "An Essay towards solving a Problem in the Doctrine of Chances", *Philosophical Transactions*, 1763

[23] Y.Hirose, K.Yamashita, and S.Hijiya, "Back-propagation algorithm which varies the number of hidden units," *Neural Networks*, vol. 4, no. 1, pp. 61-66, 1991.

[24] Jan A. Snyman, *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer Publishing, 2005.

[25] L. Fausett, *Fundamentals of Neural Networks : architectures, algorithms,and applications*, Prentice-Hall, 1994.

[26] L. Prechelt, Automatic Early Stopping Using Cross Validation: Quantifying the criteria, *Neural Networks 11*, pp. 761-767, 1998.

[27] J. Platt, “Fast Training of Support Vector Machines using Sequential Minimal Optimization,” *Advances in Kernel Methods - Support Vector Learning*, MIT Press, 1998.

[28] J. Platt, “Using Sparseness and Analytic QP to Speed Training of Support Vector Machines,” *Advances in Neural Information Processing Systems 11*, MIT Press, 1999.

[29] M. T. Manry, S. J. Apollo, L. S. Allen, W. D. Lyle, W. Gong, M.S. Dawson, and A. K. Fung," Fast Training of Neural Networks for Remote Sensing," *Remote Sensing Reviews*, vol. 9, pp. 77-96, 1994.

[30] Gilbert Strang, *Introduction To Linear Algebra*, Wesley-Cambridge Press, 1993

[31] S. Chen, C. F. N. Cowan, and P. M. Grant, “Orthogonal least squares learning algorithm for radial basis function networks,” *IEEE Trans. Neural Networks*, vol. 2, 1991, pp. 302–309.

[32] W. Kaminski and P. Strumillo, "Kernel Orthonormalization in Radial Basis Function Neural Networks," *IEEE Trans. Neural Networks*, Vol. 8, No. 5, 1997, pp. 1177-1183.

[33] H.C.Yau and M. T. Manry, “ Iterative Improvement of a Nearest Neighbor Classifier,” Neural Networks, Vol.4, Number 4, pp.517-524,1991.

[34] Abdul A. Abdurrab, Michael T. Manry, Jiang Li, Sanjeev S. Malalur and Robert G. Gore, “A Piecewise Linear Network Classifier”, Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007.

[35] Bors, A.G., Pitas, I., (1996) “Median radial basis functions neural networks,” *IEEE Transaction on Neural Networks,* vol. 7, no. 6, pp. 1351-1364. Casdagli, M. (1989) “Nonlinear prediction of chaotic time series,” *Physica* D, vol. 35, pp. 335-356.

[36] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998.

[37] W. Gong, H. C. Yau, and M. T. Manry, "Non-Gaussian Feature Analyses Using a Neural Network," Progress in Neural Networks, vol. 2, 1994, pp. 253-269.

[38] R.R. Bailey, E. J. Pettit, R. T. Borochoff, M. T. Manry, and X. Jiang, "Automatic Recognition of USGS Land Use/Cover Categories Using Statistical and Neural Network Classifiers," Proceedings of SPIE OE/Aerospace and Remote Sensing, April 12-16, 1993, Orlando Florida.

[39] W. H. Delashmit and M. T. Manry, "Recent Developments in Multilayer Perceptron Neural Networks", Proceedings of the 7th annual Memphis Area Engineering and Science Conference (MAESC), 2005

[40] Changhua Yu, M. T. Manry, “ A Modified Hidden Weight Optimization Algorithm for Feed-forward Neural Networks,” the 36th Asilomar Conference on Signals, Systems, & Computers ’02, pp. 1034 – 1038

[41] Rohani, K. ; Manry, M.T.; “ Nonlinear Neural Network Filters for Image Processing” the Acoustics, Speech, and Signal Processing, 1992. ICASSP-92.,1992 IEEE International Conference, vol 2, 23-26 March 1992, pp. 373-376

[42] Valdimir N. Vapnik, " An Overview of Statistical Learning Theory," *IEEE Trans. On Neural Networks*, vol. 10, no. 5, September 1999, pp. 988-999.

[43] Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, Mark E. Oxley and Bruce W. Suter, "The Multilayer Perceptron as an approximation to a Bayes optimal discriminant function," *IEEE Trans Neural Networks,* TNN-1(4):296-298, 1990.

[44] Abdul Aziz, A Piecewise Linear Classifier , Thesis presentation, UTA, 2007.

[45] Jonathan Richard Shewchuk, "An Introduction to Conjugate Gradient without the Agonizing Pain" Edition 1 ¼ 1994, Carnegie Mellon University, Pittsburg, PA.

[46] V.Vapnik and A. Chervonenkis, "On the Uniform Convergence of Relative Frequencies of events to their probabilities," *Theory of Probability and its Applications, 16(2):264-280, 1971.*

[47] P.E. Gill, W. Murray, and M.H. Wright, "Practical Optimization," *Academic Press, New York* 1981.

[48] Athanasios Papoulis, "Probability*, Random Variables, and Stochastic Processes",* second edition. New York: McGraw- Hill.

[49] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, Vol. 2, No. 5, pp. 359-366, 1989.

[50] K. Hornik, M. Stinchcombe, and H. White, "Universal Approximation of an Unknown Mapping and its Derivatives Using Multilayer Feedforward Networks," *Neural Networks,* vol. 3, pp. 551-560, 1990.

[51] Michael D. Richard and Richard P. Lippman, "Neural Network Classifiers estimate Bayesian a-posteriori probabilities," *Neural Computation*, vol. 3, no. 4, pp. 461-483, 1991.

[52] Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, Mark E. Oxley and Bruce W. Suter, “The Multilayer Perceptron as an approximation to a Bayes optimal discriminant function,” *IEEE Trans Neural Networks*, TNN-1(4):296-298, 1990.

[53] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[54] J. H. Friedman, “Multivariate adaptive regression splines,” *Annals of Statistics*, vol. 19, no. 1, pp. 1-141, 1991.

[55] D.R. Hush and B. Horne, “Efficient algorithms for function approximation with piecewise linear sigmoidal networks,” *IEEE Trans. Neural Networks*, Vol. 9, No. 6, pp. 1129-1141, 1998.

[56] E.F. Gad, A.F. Atiya, S. Shaheen, A. El-Dessouki, “A new algorithm for learning in piecewise-linear neural networks,” *Neural Networks 13*, pp. 485–505, 2000.

[57] Strang G (1998). "Introduction to Linear Algebra". Section 6.7. 3rd ed., Wellesley-Cambridge Press. ISBN 0-9614088-5-5.

[58] Strang, Gilbert (2003). Introduction to Linear Algebra, 3rd edition, Wellesley, Massachusetts: Wellesley-Cambridge Press, 74-76.

BIOGRAPHICAL INFORMATION

Jimy Shah was born in Vadodara, India in 1983. He received the Bachelor of Engineering in Electronics and Communication from Gujarat University in 2005 and Master of Science in Electrical Engineering from University of Texas at Arlington in 2007.

He has been involved in research activities in Image Processing and Neural Networks Laboratory (IPNNL) since 2006. His main area of research has been Neural Networks and Pattern Recognition. He has served as a Graduate Teaching Assistant for the course of Digital Signal Processing in the Electrical Engineering department of University of Texas at Arlington (2006-2007).