6.1 Introduction

The construction of machines capable of learning from experience/examples has been the subject of several debates both in philosophical and technological pursuits for a long time. The availability of reliable learning systems is of strategic importance, as there are many tasks that cannot be solved by classical programming techniques due to the lack of mathematical models of the problem. The approach of using examples consisting of inputs and desired output to synthesize programs is known as learning methodology. The mapping of inputs to outputs is performed with an underlying function. In the case of classification, this function is sometimes referred to as the decision function. A particular set or class of candidate functions known as hypotheses will be chosen before beginning to learn the correct function. The algorithm which takes the training data as input, and selects a hypothesis from the hypothesis space is referred to as the learning algorithm. A learning problem with binary outputs is referred to as a binary classification problem, and the one with a finite number of categories is multiclass classification. A learning model can be either online or batch type. In batch learning, data are given to the learner at the beginning of learning, whereas online learning accepts data as and when it is submitted to the system. In the latter case, they update their current hypothesis in response to each new example. The quality of learning is assessed by the total number of mistakes made during the process of learning [164].

A learning algorithm is a function which takes as its input a learning sample and outputs a classification or regression model [165]. It performs mapping from a set of input variables $x_1, \ldots, x_n$, to an output variable $t$ representing the class label. In more complex problems there may be several output variables denoted by $t_i$ where $i = 1, 2, \ldots, m$. In general, it will not be possible to determine a suitable form for the required mappings except with the help of a data
set of examples. The mapping is therefore modeled in terms of some mathematical function which contains a number of adjustable parameters. Such functions can be written in the following form,

\[ t_i = t(x;w) \quad (6.1) \]

where \( w \) is the weight vector. The process of determining the values for these parameters on the basis of the data set is called learning or training. Neural network models as well as many conventional approaches to statistical pattern recognition can be viewed as specific choices for the functional forms used to represent the mapping (6.1), together with particular procedures for optimizing the parameters in the mapping [166]. This chapter is a discussion on two learning algorithms for neural networks, ELM and OS-ELM, and a classifier based on statistical learning technique, SVM.

6.2 Neural Networks

The artificial neural network is designed to mimic the decision making ability of the brain, and hence its structure resembles that of the nervous system, featuring a massively interconnected set of neurons (nodes). The nodes (processing elements) are organized into an initial input layer, intermediate hidden layers, and a final output layer. Each node in the input layer has a single input corresponding to one of the features of the input vector, and the output layer contains a node for each class. Each node in a layer is fully connected to every other node in the following layer through a set of weights. Neural networks have the ability to approximate complex nonlinear mappings directly from input samples. For an \( n \)-dimensional feature vector, the node will have \( n+1 \) inputs including the bias term. The output of the node is calculated as the weighted sum of its inputs called the net sum, or simply \( net \), passed through the activation function \( f \). The net sum creates a linear decision boundary which is then modified by the activation function. Popular choice for the activation function \( f \) includes,

\[
\begin{align*}
(i) & \quad f(\text{net}) = \begin{cases} 
1 & \text{if net} \geq 0 \\
0 & \text{otherwise}
\end{cases} \\
(ii) & \quad f(\text{net}) = \begin{cases} 
1 & \text{if net} \geq 0 \\
-1 & \text{otherwise}
\end{cases} 
\end{align*}
\]

(6.2)
When used with this activation function, the node is also known as perceptron.

(ii) The identity (linear) function
\[ f(\text{net}) = \text{net} \]  \hspace{1cm} (6.3)

(iii) The logistic (sigmoid) function
\[ f(\text{net}) = \frac{1}{1 + e^{-\beta \text{net}}} \]  \hspace{1cm} (6.4)

(iv) The hyperbolic tangent (sigmoid) function
\[ f(\text{net}) = \tanh(\beta \text{net}) = \frac{2}{1 + e^{-\beta \text{net}}} - 1 \]  \hspace{1cm} (6.5)

The logistic function takes values between 0 and 1, whereas the range of hyperbolic tangent is between -1 and 1. The parameter \( \beta \) in the sigmoidal functions controls the sharpness of the function transition at zero, and both functions approach thresholding function as \( \beta \) approaches 1, and to a linear function as \( \beta \) approaches 0. Appropriate combination of perceptrons can be quite powerful and approximate an arbitrarily complex nonlinear decision boundary [2].

Different architectures for neural networks have been proposed in the literature. The learning algorithms discussed in this chapter are for Single hidden Layer Feedforward Network (SLFN).

6.2.1 Single hidden layer feedforward network

SLFN is a feedforward neural network, indicating that the information flows in one direction from input layer to output layer through hidden layer (Figure 6.1). Although the number of nodes in the input layer and output layer is fixed, it is a free parameter to hidden layer. With the increase of hidden nodes, the predictive power of SLFN will also increase, and such a network can be used to solve more complicated problems. However, this may lead to overfitting, computational burden, or more memory requirement. Likewise, too few hidden nodes may not be sufficient to produce good results. Therefore, a reasonable number of nodes that provide the desirable performance should be used. Typically, this parameter is determined by a combination of previous expertise, amount of data available, dimensionality, complexity of the problem, trial and error, or validation on an additional dataset [2].
SLFN can approximate any continuous function and implement any classification application. The research on approximation capabilities of SLFN has focused on two aspects: universal approximation on compact input sets and approximation in a set consisting of finite number of training samples [167]. Universal approximation refers to the ability of the networks to approximate any decision boundary of arbitrary dimensionality and arbitrary complexity with arbitrary precision, given adequate amount of data and proper selection of their architectural and training parameters [2]. For function approximation in finite training set, Huang and Babri [168] show that SLFN with at most \( N \) hidden nodes and with almost any nonlinear activation function can exactly learn \( N \) distinct observations.

Over the past three decades, SLFN has been discussed by many researchers. The two main SLFN architectures exist: (1) with additive hidden nodes, and (2) with RBF nodes in the hidden layer. SLFN network functions with \( \tilde{N} \) hidden nodes can be represented by

\[
f_{\tilde{N}}(x) = \sum_{i=1}^{\tilde{N}} \beta_i g_i(x), \quad x \in \mathbb{R}^n, \quad \beta_i \in \mathbb{R}
\]

where \( g_i \) denotes the hidden node output function, and \( \beta_i \) is the weight connecting \( i^{th} \) hidden node to the output node. For additive nodes, \( g_i \) is defined as

\[
g_i = g(w_i x + b_i), \quad w_i \in \mathbb{R}^n, b_i \in \mathbb{R}
\]
where \( \mathbf{w}_i \) is the weight vector connecting the input layer to the \( i^{\text{th}} \) hidden node, and \( g \) is the hidden node activation function, typically a sigmoid function (Figure 6.2). The RBF network is considered a specific SLFN which applies RBF nodes in its hidden layer. Each RBF node has its own centroid \( (\mathbf{w}_i) \) and impact factor \( (b_i) \), and its output is some radially symmetric function of the distance between input and centre. For RBF nodes with activation function \( g \), \( g \) is defined as [167]

\[
g_i = g \left( \frac{\| \mathbf{x} - \mathbf{w}_i \|}{b_i} \right), \quad \mathbf{w}_i \in \mathbb{R}^p, b_i \in \mathbb{R}^+ \quad (6.8)
\]

![Sigmoid function](image)

**Figure 6.2 Sigmoid function**

The BackPropagation (BP) algorithm and its variants have been the backbone for training SLFN with additive hidden nodes. It is basically a batch learning algorithm, and Stochastic Gradient descent Back Propagation (SGBP) is its variant for sequential learning applications. SGBP suffers from slow training error convergence as large number of training data may be required. The network size of SGBP needs to be predefined and fixed. Sequential learning algorithms are also used in feedforward networks with RBF nodes. These include Resource Allocation Network (RAN) and its extensions. The number of RBF hidden nodes in RAN and its variants cannot be predefined. RAN and RAN via Extended Kalman Filter (RANEKF) determine whether to add a new node based on the novelty of incoming data. Besides that, Minimal Resource Allocation Network (MRAN), Growing And Pruning - RBF (GAP-RBF) and GGAP-RBF (Generalized GAP-RBF) prune insignificant nodes from the network. These algorithms require
tuning many control parameters, and in the case of large problems, the learning speed may be slow. They handle data one by one using either additive or RBF nodes [169].

The network with fixed architecture and randomly assigned hidden nodes is called ELM where the output parameters are determined by ordinary least square and according to theorem 6.1 \( \lim_{N \to \infty} \| f - f_N \| = 0 \) holds for ELM [167].

**Theorem 6.1:** Let \( e_N \equiv f - f_N \) denotes the residual error function for the current network \( f_N \) with \( \tilde{N} \) hidden nodes, where \( f \in L^2(X) \) is the target function. It can be proved that given any bounded nonconstant piecewise continuous activation function \( g:R \to R \) for additive nodes or integrable piecewise continuous activation function \( g:R \to R \) (and \( \int g(x)dx \neq 0 \)) for RBF nodes, for any continuous target function \( f \) and any randomly generated sequence \( \{ g_N \} \),

\[
\lim_{N \to \infty} \| f - f_N \| = 0 \text{ holds with probability one if } \beta_n = \frac{\langle e_n, g_n \rangle}{\| g_n \|^2}
\]

**6.3 Extreme Learning Machine**

The learning speed of feedforward neural networks is far slower than required, and it has been a major bottleneck in their applications for past decades. Traditionally, all the parameters of feedforward network need to be tuned, and thus there exists dependency among different layers of parameters. The gradient based methods have been used for a long time in various learning algorithms for feedforward networks. However, it is clear that these methods are generally slow and may converge to local minima. Many iterative learning steps may be required to obtain better learning performance. In this section, a learning algorithm called Extreme Learning Machine (ELM) is introduced for SLFN which is suitable in applications requiring fast prediction and response capability. SLFN (having \( N \) hidden neurons) with randomly chosen input weights and hidden layer biases can exactly learn \( N \) distinct observations. One need not adjust these parameters during training. So the learning will be extremely fast and produces good generalization performance. The input weights and hidden layer biases of SLFN can be randomly assigned if the activation functions in the hidden layer are infinitely differentiable. Now, SLFN can be considered a linear system and the output
weights of SLFN can be analytically determined through generalized inverse operation of the hidden layer output matrices. Moore-Penrose generalized inverse and minimum norm least squares solution of a general linear system play an important role in developing ELM algorithm [170].

For \( N \) arbitrary distinct samples \((x_i, t_i)\), where \( x_i = [x_{i1}, x_{i2}, ..., x_{in}]^T \in \mathbb{R}^n \) and \( t_i = [t_{i1}, t_{i2}, ..., t_{in}]^T \in \mathbb{R}^n \), SLFN with \( \hat{N} \) hidden nodes and activation function \( g(x) \) are mathematically modeled as

\[
\sum_{j=1}^{\hat{N}} \beta_j g(x_j) = \sum_{j=1}^{\hat{N}} \beta_j g(w_j \cdot x_j + b_j) = o_j
\]

(6.9)

where \( w_i = [w_{i1}, w_{i2}, ..., w_{in}]^T \) is the weight vector connecting the \( i^{th} \) hidden node and input nodes, and \( [\beta_{i1}, \beta_{i2}, ..., \beta_{in}]^T \) is the weight vector connecting the \( i^{th} \) hidden neuron and output neurons, and \( b_i \) is the threshold of \( i^{th} \) hidden node. SLFN with \( \hat{N} \) hidden nodes and activation function \( g(x) \) can approximate these \( N \) samples with zero error means that \( \sum_{j=1}^{N} \| o_j - t_j \| = 0 \), i.e., there exist \( \beta_j, w_i, \) and \( b_i \) such that

\[
\sum_{j=1}^{\hat{N}} \beta_j g(w_i \cdot x_j + b_i) = t_j, \quad j = 1, 2, ..., N
\]

(6.10)

The above \( N \) equations can be written compactly as

\[
H\beta = T
\]

where

\[
H(w_1, ..., w_{\hat{N}}, b_1, ..., b_{\hat{N}}, x_1, ..., x_N) = \begin{bmatrix}
g(w_1 \cdot x_1 + b_1) & \cdots & g(w_{\hat{N}} \cdot x_1 + b_{\hat{N}}) \\
\vdots & \ddots & \vdots \\
g(w_1 \cdot x_N + b_1) & \cdots & g(w_{\hat{N}} \cdot x_N + b_{\hat{N}})
\end{bmatrix}_{\hat{N} \times N}
\]

\[
\beta = \begin{bmatrix}
\beta_1^T \\
\vdots \\
\beta_{\hat{N}}^T
\end{bmatrix}_{\hat{N} \times 1}, \quad \quad \quad T = \begin{bmatrix}
t_1^T \\
\vdots \\
t_N^T
\end{bmatrix}_{N \times 1}
\]

(6.11)

\( H \) is called hidden layer output matrix, its \( i^{th} \) column is the \( i^{th} \) hidden node output with respect to inputs \( x_1, x_2, ..., x_N \). If the activation function \( g \) is infinitely differentiable, we can prove that the required number of hidden nodes \( \hat{N} \leq N \). Such activation functions include sigmoid, radial basis, sine, cosine, exponential, etc.
Chapter 6

**Theorem 6.2:** Given a standard SLFN with \( N \) hidden nodes and activation function \( g: \mathbb{R} \to \mathbb{R} \) which is infinitely differentiable in any interval, for \( N \) arbitrary distinct samples \((x_i, t_i)\) where \( x_i \in \mathbb{R}^n \) and \( t_i \in \mathbb{R}^m \), for any \( w_i \) and \( b_i \) randomly chosen from any intervals of \( \mathbb{R}^n \) and \( \mathbb{R} \) respectively according to any continuous probability distribution, then with probability one, the hidden layer output matrix \( H \) of SLFN is invertible and \( \|H \beta - T\| = 0 \).

**Theorem 6.3:** Given any small positive value \( \varepsilon > 0 \) and activation function \( g: \mathbb{R} \to \mathbb{R} \) which is infinitely differentiable in any interval, there exist \( \bar{N} \leq N \) such that for \( \bar{N} \) arbitrary distinct samples \((x_i, t_i)\) where \( x_i \in \mathbb{R}^n \) and \( t_i \in \mathbb{R}^m \) for any \( w_i \) and \( b_i \) randomly chosen from any intervals of \( \mathbb{R}^n \) and \( \mathbb{R} \) respectively according to any continuous probability distribution, then with probability one \( \|H_{\bar{N}n} \beta - T_{\bar{N}m}\| < \varepsilon \).

According to theorem 6.2 and 6.3, the input weights and hidden layer biases can be randomly assigned if the activation function is infinitely differentiable. These parameters need not be adjusted at all and the hidden layer output matrix \( H \) remains unchanged during the learning process. With fixed input weights \( w_i \) and hidden layer biases \( b_i \), training SLFN is equivalent to finding a least square solution \( \hat{\beta} \) of the linear system \( H \beta = T \).

\[
\|H(w_1, \ldots, w_K, b_1, \ldots, b_N) \hat{\beta} - T\| = \min_\beta \|H(w_1, \ldots, w_K, b_1, \ldots, b_N) \beta - T\|
\]  \hfill (6.12)

where \( \|\| \) is a norm in Euclidean space. If the number of hidden nodes \( \bar{N} \) is equal to the number of distinct training samples \( N \), the matrix \( H \) is square and invertible. Thus, SLFN can approximate these training samples with zero error. However, in most cases the number of hidden nodes is much less than the number of distinct training samples, \( \bar{N} \ll N \), \( H \) is a nonsquare matrix and there may not exist \( w_i, b_i, \beta_i \) \((i = 1, 2, \ldots, \bar{N})\) such that \( H \beta = T \). According to theorem 6.4, the smallest norm least squares of the above linear system is

\[
\hat{\beta} = H' T \quad \hfill (6.13)
\]

where \( H' \) is the Moore-Penrose generalized inverse of matrix \( H \), and its calculation consumes most of the learning time of ELM.
**Definition 6.1:** A matrix \( G \) of order \( n \times m \) is the Moore-Penrose generalized inverse of matrix \( A \) (denoted as \( A^+ \)) of order \( m \times n \), if

\[
AGA = A, \ GAG = G, \ (AG)^T = AG, \ (GA)^T = GA
\]

**Definition 6.2:** \( x_0 \in \mathbb{R}^n \) is said to be a minimum norm least squares solution of a linear system \( Ax = y \) if for any \( y \in \mathbb{R}^m \)

\[
\|x_0\| \leq \|x\|, \quad \forall x \in \{x : \|Ax - y\| \leq \|Az - y\|, \forall z \in \mathbb{R}^N\}
\]

i.e., a solution \( x_0 \) is said to be a minimum norm least squares solution of a linear system \( Ax = y \) if it has the smallest norm among all the least square solutions.

**Theorem 6.4:** Let there exist a matrix \( G \) such that \( Gy \) is a minimum norm least squares solution of a linear system \( Ax = y \). Then, it is necessary and sufficient that \( G = A^+ \), the Moore-Penrose generalized inverse of matrix \( A \).

The following properties of ELM are listed in view of this theorem.

1. Minimum training error: The special solution \( \hat{\beta} = H^T \hat{T} \) is one of the least squares solutions of a general linear system \( H\beta = T \), meaning that the smallest training error can be reached by this special solution:

\[
\|H\hat{\beta} - T\| = \|HH^T\hat{T} - T\| = \min_{\beta} \|H\beta - T\|
\]

(6.14)

2. Smallest norm of weights: The special solution \( \hat{\beta} = H^T \hat{T} \) has the smallest norm among all the least squares solutions of \( H\beta = T \):

\[
\|\hat{\beta}\| = \|H^T\hat{T}\| \leq \|\beta\|, \quad \forall \beta \in \{\beta : \|H\beta - T\| \leq \|Hz - T\|, \forall z \in \mathbb{R}^{\hat{N}_N}\}
\]

(6.15)

3. The minimum norm least squares solution of \( H\beta = T \) is unique, which is \( \hat{\beta} = H^T \hat{T} \).

**6.3.1 ELM algorithm**

Given a training set \( \mathcal{X} = \{(x_i, t_i) \mid x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i = 1, 2, \ldots, N\} \), activation function \( g(x) \), and number of hidden nodes \( \hat{N} \).

- **Step 1:** Randomly assign input weight \( w \), and bias \( b_i, i = 1, 2, \ldots, \hat{N} \).
- **Step 2:** Calculate the hidden layer output matrix \( H \).
- **Step 3:** Calculate the output weight \( \beta \),

\[
\beta = H^T \hat{T}
\]

156
where \( \mathbf{T} = [t_1, t_2, ..., t_n]^{T}. \)

According to theorem 6.2, this algorithm works for any infinitely differentiable activation function \( g(x) \). According to theorem 6.3, the upper bound of the number of hidden nodes \( \tilde{N} \) is \( N \), the number of distinct training samples, i.e., \( \tilde{N} \leq N \). SLFN with \( N \) hidden nodes can learn \( N \) distinct samples exactly, and if training error is allowed, the value of \( \tilde{N} \) may be less than \( N \) [170].

ELM has the following features:
1) Learning speed of ELM is extremely fast.
2) ELM has better generalization performance than gradient based learning such as BP.
3) The traditional classic gradient based learning algorithms may face several issues like local minima, improper learning rate and overfitting, etc. ELM tends to reach solutions straight forward without such trivial issues.
4) The gradient based learning algorithms will only work for differentiable activation functions, whereas ELM could be used with many nondifferentiable activation functions.

### 6.3.2 Types of ELM

During the past five years, many variants of ELM have been proposed [9]. The online sequential mode of ELM is given in [169]. This algorithm is referred to as Online Sequential Extreme Learning Machine (OS-ELM), which can learn data one-by-one or chunk-by-chunk with fixed or varying chunk size. Only the number of hidden nodes and the size of blocks are used as control parameters. OS-ELM is faster than other sequential learning algorithms and produces better generalization performance. Usually, ELM may need higher number of hidden neurons due to the random determination of the input weights and hidden biases. In [171], a modified differential evolution algorithm is used to search for the optimal input weights and hidden biases. This approach is known as Evolutionary Extreme Learning Machine (E-ELM). Evolutionary algorithms are widely used as a global searching method for optimization. E-ELM is able to achieve good generalization performance with much more compact networks.

The generalization performance of ELM for sparse data classification problem depends on three parameters – number of hidden neurons, input weights, and bias values. In [172], the
authors proposed two approaches, Real Coded Genetic Algorithm based ELM (RCGA-ELM) and Sparse ELM (S-ELM), to set these parameters. RCGA-ELM used two genetic operators for this purpose. The network based genetic operator controls the number of hidden neurons and the weight based genetic operator evolves the input weight and bias values. S-ELM (KS-ELM [173]), which is computationally less intensive than RCGA-ELM, uses a K-fold Cross-Validation (CV) to select the ELM parameters. CV approach is used to select the input weights and bias values for the given number of hidden nodes such that the estimated generalization error is small. Then, this process is repeated with a random set of hidden nodes and determined the optimal number of hidden nodes for which training and CV efficiencies are high.

A number of algorithms have been proposed to choose the network structure of different applications. They are categorized into pruning method, forward selection method, and incremental learning algorithms. Rong et al. [174] has proposed a Pruned ELM (P-ELM) for pattern classification applications, which starts with a large network and then eliminates the hidden nodes with low relevance to the class label. The statistical criteria such as Chi-square ($\chi^2$) and information gain are utilized to check the relevance. Miche et al. [175] presented Optimally Pruned ELM (OP-ELM), in which hidden nodes are ranked by multiresponse sparse regression algorithm. The final model is selected by leave-one-out CV. A Constructive hidden nodes Selection method for ELM (CS-ELM) is proposed in [176] based on a stepwise forward selection method. At each step, the hidden node with an output that has the highest correlation with the current residual is selected. The hidden nodes selection stops automatically when the unbiased risk estimation criterion reaches its minimum. In [177], Lan et al. has proposed a Two Stage ELM (TS-ELM) consisting of forward selection stage and backward elimination stage. In forward selection, many groups of hidden nodes are randomly generated in each step, and the group of hidden nodes with the highest net contribution is selected and added to the network. The selection process is terminated when the stopping criterion of final prediction error achieves its minimum. In backward elimination, the hidden nodes selected in the previous stage are reviewed one by one, and insignificant hidden nodes are removed from the network.

In [167], Huang et al. developed an Incremental ELM (I-ELM), in which nodes are added one by one to the hidden layer, while keeping the output weights of the existing hidden nodes
unchanged. I-ELM requires only two control parameters – target error and maximum number of hidden nodes. In [178], Barron’s convex optimization learning method [179] is incorporated into I-ELM, and the resulting algorithm is referred to as Convex I-ELM (CI-ELM). Different from I-ELM, CI-ELM recalculates the output weights of the existing hidden nodes after a new hidden node is added. It can achieve faster convergence rate, more compact network architecture, better generalization performance, and more stability than I-ELM.

It is found that some of the hidden nodes in I-ELM may play a very minor role in the network output and this may eventually increase the network complexity. In order to avoid this issue and to obtain more compact network architecture, Huang and Lei [180] proposed an Enhanced method for I-ELM (Ei-ELM). At each learning step, several hidden nodes (say $k$ nodes, where $k$ is a constant) are randomly generated and among them the hidden node leading to the smallest residual error will be added to the existing network. Ei-ELM can achieve faster convergence rate and much more compact network architecture than I-ELM. I-ELM is a specific case of Ei-ELM when $k = 1$. Guorui et al. [181] proposed a method to automatically determine the number of hidden nodes in generalized SLFN. This approach is referred to as Error Minimized ELM (EM-ELM), and can add hidden nodes one by one or group by group. If the network output error is less than the target error, hidden nodes will not be added and the learning terminates. Otherwise, hidden nodes are added and generate new hidden layer output matrix. EM-ELM reduces the computation complexity by updating only the output weights incrementally. It is much faster than other sequential/incremental/growing algorithms, and produce good generalization performance.

6.4 Online Sequential Extreme Learning Machine

The learning based on ELM requires all the data to be present at the beginning of training. However, the training data may arrive one-by-one or block-by-block in real applications. To cope with this situation, a sequential learning variant of ELM called OS-ELM can be used. OS-ELM uses the ideas of batch learning ELM. This algorithm handles both additive and RBF nodes in a unified framework. It is a versatile learning algorithm in the following sense.
1) The training observations are sequentially (one-by-one or block of fixed or varying size) presented to the learning algorithm.
2) Only the newly arrived observations are seen and learned.
3) Training observations are discarded as soon as the learning procedure of that particular block or single observation is completed.
4) The learning algorithm has no prior knowledge about the number of observations to be trained by it.

Consider the case of \( \text{rank}(H) = \hat{N} \), the number of hidden nodes. Under this condition, \( H' \) of (6.13) is given by

\[
H' = (H^T H)^{-1} H^T
\]  

(6.16)

This is called left pseudo inverse of \( H \) from the fact that \( H' H = I_{\hat{N}} \). Substitute (6.16) in (6.13)

\[
\hat{\beta} = (H^T H)^{-1} H^T T
\]  

(6.17)

The above equation is called least squares solution to \( H\beta = T \). Sequential implementation of least squares solution results in OS-ELM.

Given a block of initial training set \( \mathbf{X}_0 = \{(x_i, t_i)\}_{i=1}^{N_0} \) and \( N_0 \geq \hat{N} \), if one considers to use batch ELM, one needs to consider only the problem of minimizing \( \|H_0\beta - T_0\| \) where

\[
H_0 = \begin{bmatrix}
G(w_1, b_1, x_1) & \cdots & G(w_N, b_N, x_1) \\
\vdots & \ddots & \vdots \\
G(w_1, b_1, x_{N_0}) & \cdots & G(w_N, b_N, x_{N_0})
\end{bmatrix}_{N_0 \times \hat{N}} \quad \text{and} \quad T_0 = \begin{bmatrix}
t_1^T \\
\vdots \\
t_{N_0}^T
\end{bmatrix}_{N_0 \times m}
\]  

(6.18)

By theorem 6.2, the solution is given by \( \beta^{(0)} = K_0^{-1} H_0^T T_0 \) where \( K_0 = H_0^T H_0 \). Suppose that we are given another block of data \( \mathbf{X}_1 = \{(x_i, t_i)\}_{i=N_0+1}^{N_0+N_1} \), where \( N_1 \) denotes the number of observations in this block; then the problem becomes minimizing

\[
\left\| \begin{bmatrix}
H_0 \\
H_1
\end{bmatrix}\beta - \begin{bmatrix}
T_0 \\
T_1
\end{bmatrix} \right\|
\]  

(6.19)

160
Chapter 6

\[ H_1 = \begin{bmatrix} G(w_1, b_1, x_{N_0+1}) & \cdots & G(w_{N_0}, b_{N_0}, x_{N_0+1}) \\ \vdots & \ddots & \vdots \\ G(w_1, b_1, x_{N_N+1}) & \cdots & G(w_{N_N}, b_{N_N}, x_{N_N+N_1}) \end{bmatrix} \]

where

\[ T_1 = \begin{bmatrix} t_{N_0+1}^T \\ \vdots \end{bmatrix} \]

and

\[ \beta^{(1)} = K_1 \left[ \begin{array}{c} H_0 \\ H_1 \end{array} \right]^T \left[ \begin{array}{c} T_0 \\ T_1 \end{array} \right] \]

(6.20)

where

\[ K_1 = \begin{bmatrix} H_0^T \\ H_1 \end{bmatrix} \left[ \begin{array}{c} H_0 \\ H_1 \end{array} \right] \]

\[ = \begin{bmatrix} H_0^T & H_1^T \end{bmatrix} \left[ \begin{array}{c} H_0 \\ H_1 \end{array} \right] \]

\[ = K_0 + H_1^T \]

For sequential learning, we have to express \( \beta^{(1)} \) as a function of \( \beta^{(0)} \), \( K_0 \), \( H_1 \) and \( T_1 \).

\[ \left[ \begin{array}{c} H_0 \\ H_1 \end{array} \right]^T \left[ \begin{array}{c} T_0 \\ T_1 \end{array} \right] = H_0^T T_0 + H_1^T T_1 \]

(6.21)

\[ = K_0 K_0^{-1} H_0^T T_0 + H_1^T T_1 \]

\[ = K_0 \beta^{(0)} + H_1^T T_1 \]

\[ = (K_1 - H_1^T H_1) \beta^{(0)} + H_1^T T_1 \]

\[ = K_1 \beta^{(0)} - H_1^T H_1 \beta^{(0)} + H_1^T T_1 \]

\[ \therefore \beta^{(1)} = K_1^{-1} \left[ \begin{array}{c} H_0 \\ H_1 \end{array} \right]^T \left[ \begin{array}{c} T_0 \\ T_1 \end{array} \right] \]

(6.22)

\[ = K_1^{-1} \left( K_1 \beta^{(0)} - H_1^T H_1 \beta^{(0)} + H_1^T T_1 \right) \]

\[ = \beta^{(0)} + K_1^{-1} H_1^T \left( T_1 - H_1 \beta^{(0)} \right) \]

By generalizing the previous arguments, a recursive algorithm for updating least squares solution can be written as follows. When \( (k+1)^{\text{th}} \) block of data set

\[ \mathcal{N}_{k+1} = \{(x_i, t_i)\}_{i=1}^{N_{k+1}} \]

is received, where \( k \geq 0 \) and \( N_{k+1} \) denote the number of observations in the \( (k+1)^{\text{th}} \) block, we have

161
\[ K_{k+1} = K_k + H_k^T H_k \]  
\[ \beta^{(k+1)} = \beta^{(k)} + K_k^{-1} H_k^T (T_k - H_k \beta^{(k)}) \]  

where \( H_k = \begin{bmatrix} 
G(w_1, b, x_{\text{sum}_{j=1}^{N_j}}) & \cdots & G(w_{\tilde{N}}, b, x_{\text{sum}_{j=1}^{N_j}}) \\
\vdots & \ddots & \vdots \\
G(w_1, b, x_{\text{sum}_{j=1}^{N_j}}) & \cdots & G(w_{\tilde{N}}, b, x_{\text{sum}_{j=1}^{N_j}}) 
\end{bmatrix}_{N_{x+l} \times \tilde{N}} \]

and \( T_k = \begin{bmatrix} 
t_{\text{sum}_{i=1}^{N_i}}^T \\
\vdots \\
t_{\text{sum}_{i=1}^{N_i}}^T 
\end{bmatrix}_{N_{x+l} \times m} \)

The update formula for \( K_{k+1}^{-1} \) is derived using Woodbury formula

\[ K_{k+1}^{-1} = (K_k + H_k^T H_k)^{-1} = K_k^{-1} - K_k^{-1} H_k^T (I + H_k^T K_k^{-1} H_k)^{-1} \times H_k K_k^{-1} \]  

Let \( P_{k+1} = K_{k+1}^{-1} \), then

\[ P_{k+1} = P_k - P_k H_k^T (I + H_k^T P_k H_k)^{-1} H_k^T P_k \]

\[ \beta^{(k+1)} = \beta^{(k)} + P_{k+1} H_k^T (T_k - H_k \beta^{(k)}) \]

OS-ELM can be implemented to suit the way data arrives without sacrificing the accuracy. It can be seen that OS-ELM and ELM can achieve same learning performance when \( \text{rank}(H_0) = \tilde{N} \). In order to make \( \text{rank}(H_0) = \tilde{N} \) and \( \text{rank}(K_0) = \tilde{N} \), the number of initialization data \( N_0 \) should not be less than the hidden node number \( \tilde{N} \) [169].

**6.4.1 OS-ELM algorithm**

Select the type of node (additive or RBF), the corresponding activation function \( g \), and the number of hidden nodes \( \tilde{N} \). The data \( X = \{(x_i, t_i) | x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i = 1, 2, \ldots \} \) arrives sequentially. OS-ELM consists of two phases - initialization phase and sequential learning phase. In the initialization phase, the matrix \( H_0 \) is filled up for use in the learning phase. According to theorem 6.2, \( \text{rank}(H_0) = \tilde{N} \) if the first \( \tilde{N} \) training data are distinct. In most cases,
Chapter 6

the number of training data required in the initialization phase can be equal or close to \( \bar{N} \). Following the initialization phase, learning phase commences either on a one-by-one or block-by-block basis as desired. Once the data is used, it is discarded and not used any more.

Step 1) Initialization Phase: Initialize the learning using a block of training data \( \mathbf{X}_0 = \{(\mathbf{x}_i, t_i)\}_{i=1}^{\bar{N}} \) from the given training set \( \mathbf{X} = \{(\mathbf{x}_i, t_i)\mid \mathbf{x}_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i=1, 2, ... \}, \bar{N}_0 \geq \bar{N}.

a) Assign random input weights \( \mathbf{w}_i \) and bias \( b_i \) (for additive hidden nodes) or centre \( \mathbf{w}_i \) and impact factor \( b_i \) (for RBF hidden nodes), \( i = 1, 2, ..., \bar{N} \).
b) Calculate initial hidden layer output matrix \( \mathbf{H}_0 \).
c) Estimate the initial output weight \( \beta^{(0)} \).
d) Set \( k = 0 \).

Step 2) Sequential learning phase: Present \((k+1)^{th}\) block of new observations \( \mathbf{X}_{k+1} \).

a) Calculate partial hidden layer output matrix \( \mathbf{H}_{k+1} \) for the \((k+1)^{th}\) block of data \( \mathbf{X}_{k+1} \).
b) Set \( T_{k+1} \)
c) Calculate the output weight \( \beta^{(k+1)} \).
d) Set \( k = k + 1 \). Repeat step 2.

When the data arrives one-by-one, output weight can be calculated using Sherman-Morrison formula

\[
P_{k+1} = P_k - \frac{P_k \mathbf{h}_{k+1} \mathbf{h}_{k+1}^T P_k}{1 + \mathbf{h}_{k+1}^T P_k \mathbf{h}_{k+1}} \tag{6.28}
\]

\[
\beta^{(k+1)} = \beta^{(k)} + P_k \mathbf{h}_{k+1} (\mathbf{t}_{k+1}^T - \mathbf{h}_{k+1}^T \beta^{(k)}) \tag{6.29}
\]

where \( \mathbf{h}_{k+1} = [G(\mathbf{w}_1, b_1, \mathbf{x}_{k+1})...G(\mathbf{w}_N, b_N, \mathbf{x}_{k+1})] \). If \( N_0 = N \), then OS-ELM becomes the batch learning ELM. Thus, batch ELM can be considered a special case of OS-ELM when all the training observations are present in a single learning iteration.

OS-ELM achieves lowest testing and training RMSE as well as training time compared to other sequential learning algorithms. It also shows better generalization performance. As far as the training time is concerned, one-by-one sequential mode takes the longest while batch learning the shortest time and any block mode operation falls in between [169].
6.5 Kernel Methods for Machine Learning

The kernel methods, introduced in 1990, are used for the representation of nonlinear relations discovered in the data. These new generation learning algorithms utilize techniques from optimization, statistics, and functional analysis to achieve maximal generality, flexibility and performance [165]. A kernel is a function \( K \), such that for all \( x, z \in X \)

\[
K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle
\]  

(6.30)

where \( \phi \) is mapping from \( X \) to a feature space \( F \). The following properties of the function \( K(x, z) \) are necessary to ensure that it is a kernel for some feature space. Clearly, the function must be symmetric,

\[
K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle = \langle \phi(z) \cdot \phi(x) \rangle = K(z, x)
\]  

(6.31)

and satisfy the inequalities,

\[
K(x, z)^2 = \langle \phi(x) \cdot \phi(z) \rangle^2 \leq \|\phi(x)\|^2 \|\phi(z)\|^2
\]

\[
= \langle \phi(x) \cdot \phi(x) \rangle \langle \phi(z) \cdot \phi(z) \rangle = K(x, x)K(z, z)
\]  

(6.32)

One common preprocessing strategy in machine learning involves changing the representation of the data:

\[
x = (x_1, ..., x_n) \mapsto \phi(x) = (\phi_1(x), ..., \phi_n(x))
\]  

(6.33)

This step is equivalent to mapping the input space \( X \) into a new space, \( F = \{\phi(x) \mid x \in X\} \) [164]. Instead of trying to fit a nonlinear model, one can map the problem from the input space to a new space called feature space by doing a nonlinear transformation using suitable chosen basis functions and then use a linear model in the feature space. This is known as kernel trick. The linear model in feature space corresponds to a nonlinear model in input space. The choice of kernel function is crucial for the success of all kernel algorithms because the kernel constitutes prior knowledge about a task.

**Kernel trick:** Mercer’s theorem states that any continuous, symmetric, positive semidefinite kernel function \( K(x, y) \) can be expressed as a dot product in a high dimensional
space. If the arguments to the kernel are as in a measureable space \( X \), and if the kernel is positive semidefinite, i.e.,

\[
\sum_{i,j} K(x_i, x_j)c_i c_j \geq 0
\]

(6.34)

for any finite subset \( \{x_1, \ldots, x_n\} \) of \( X \) and subset \( \{c_1, \ldots, c_n\} \) of objects, then there exists a function \( \phi(x) \) whose range is in an inner product space of possibly high dimension, such that

\[
K(x, y) = \phi(x) \cdot \phi(y)
\]

(6.35)

**Advantages** [182]:

- The kernel defines a similarity measure between two data points and thus allows one to incorporate prior knowledge of the problem domain.

- Most importantly, the kernel contains all of the information about the relative positions of the inputs in the feature space and the actual learning algorithm is based only on the kernel function and can thus be carried out without explicit use of the feature space. The training data only enter the algorithm through their entries in the kernel matrix (a Gram matrix), and never through their individual attributes. Because one never explicitly has to evaluate the feature map in the high dimensional feature space, the kernel function represents a computational shortcut.

- The number of operations required is not necessarily proportional to the number of features.

### 6.6 Support Vector Machine

Support Vector Machine (SVM) is a system for efficiently training the linear learning machines in the kernel induced feature space [164]. It is based on statistical learning theory [183] and quadratic programming optimization. An SVM is basically a binary classifier and multiple SVMs can be combined to form a system for multiclass classification. There may be instances in which two classes are not linearly separable. In such cases, one prefers nonlinear mapping of data into some higher dimensional space called feature space where it is linearly separable. There are several views and corresponding ways of finding the optimal mapping.
The views follow from the concepts like bias-variance trade off, Vapnik-Chervonenkis (VC) dimension and structural risk minimization, minimum description length, etc [165].

SVMs try to find the optimal decision boundary by maximizing the margin (using a central hyperplane) between the boundaries (bounding planes) of different classes. To do so, they identify those instances of each class that define the boundary of that class in the feature space. These instances, considered to be the most informative ones, are called support vectors. SVMs really shine in their ability to find decision boundaries in complex and linearly nonseparable high dimensional spaces. In fact, the novelty of SVMs is in their ability to use a kernel function for mapping into a high dimensional space, and find support vectors in that high dimensional space. What makes SVMs even more popular is the fact that they do it through a procedure affectionately known as kernel trick, which does not require any calculation in the high dimensional space [2]. Separating the classes with a large margin minimizes a bound on the expected generalization error. If training vectors are separated without errors by an optimal hyperplane, the expected error rate on the test sample is bounded by the ratio of the expectation of the support vectors to the number of training vectors. Since this ratio is independent of the dimension of the problem, good generalization is guaranteed [184].

SVMs are learning systems that use a hypothesis space of linear functions in a high dimensional feature space trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. An important feature of these systems is that while enforcing the learning biases suggested by the generalization theory, they also produce sparse dual representations of the hypothesis, resulting in extremely efficient algorithms. This is due to Karush-Kuhn-Tucker (KKT) conditions. Another feature of SVM is that due to Mercer’s conditions on the kernels, the corresponding optimization problems are convex and hence have no local minima. The generalization theory gives clear guidance about how to control capacity and hence prevent overfitting by controlling the hyperplane margin measures, while optimization theory provides the mathematical techniques necessary to find hyperplanes optimizing these measures. One can, for example, optimize the maximal margin, the margin distribution, the number of support vectors, etc [164].
Chapter 6

The different SVM formulations such as Cost sensitive-Support Vector Classification (C-SVC), ν-SVC, distribution estimation (one class SVM), ε-support vector regression (ε-SVR), and ν-SVR are available in the literature [185]. Of these, C-SVC is used in the current study. The advantage of using support vector classifier is that it offers a possibility to train generalizable, nonlinear classifiers in high dimensional spaces using a small training set. Moreover, for large training sets, it typically selects a small support set and thereby minimizing the computational requirements during testing [7].

Given the matrix of data points $A = \begin{bmatrix} x_1^T & \cdots & x_n^T \end{bmatrix}$, $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T$, $i = 1, 2, \ldots, N$, and the corresponding class vector $t = [t_1, t_2, \ldots, t_l]^T$, $t_i \in \{1, -1\}$. The aim is to find separating hyperplane of the form $w_1x_1 + w_2x_2 + \ldots + w_nx_n - b = 0$ and two bounding planes of the form $w_1x_1 + w_2x_2 + \ldots + w_nx_n - b \geq 1$ and $w_1x_1 + w_2x_2 + \ldots + w_nx_n - b \leq -1$. The perpendicular distance of the first bounding hyperplane from the origin is $\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}}$ and the distance to the second plane is $\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}}$. Therefore, the distance between the bounding hyperplane is $\frac{2}{\sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}}$. Then, find the weights $w = [w_1, \ldots, w_n]^T$ and bias $b$ that maximizes this distance. Maximization of $\frac{2}{\sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}}$ is same as minimization of $\frac{w_1^2 + w_2^2 + \ldots + w_n^2}{2} = \frac{1}{2}w^Tw$. So the learning problem is

$$\min \frac{1}{w, b} \frac{1}{2}w^Tw$$

(6.36)

Training of SVM consists of finding the suitable values for $w$ and $b$. The decision boundary is $w^Tx - b = 0$, and decision function is $f(x) = \text{sign}(w^Tx - b)$. The problem is easily solved in terms of its Langrangian dual variables.

While training, one must try to maximize the margin between bounding planes and minimize the number of points that contributes error. More margins will cause more points with error. So introduce an error penalty multiplier $C$ through which control the weight given
to these contradicting requirements. When a point $x_i$ in $A_i$ lie either between the bounding hyperplanes or beyond into the region $A_i$ (i.e., region where $w^T x_i - b \leq -1$), add a positive quantity $\xi_i$ to the left of the inequality to satisfy the constraint $w^T x_i - b \geq 1$. It now becomes $w^T x_i - b + \xi_i \geq 1$. Similarly, subtract a positive quantity $\xi_i$ from the left of inequality for points in $A_i$ that falls between bounding hyperplanes and into the region $A_i$. The equation for such a point is $w^T x_i - b - \xi_i \leq -1$. For all other points, add $\xi_i$ with value 0.

Now the objective function is $Ce^T \xi + \frac{1}{2} w^T w$. $e^T \xi$ is the sum of nonnegative errors $\sum_{i=1}^{N} \xi_i$, and $C$ is supplied by the user at the time of training. The basic concept behind SVM is to search for a balance between the regularization term $(1/2)w^T w$ and the training errors. Minimization of $Ce^T \xi + \frac{1}{2} w^T w$ r to $w$ and $\xi$ causes maximum separation between bounding planes with minimum number of points crossing their respective bounding planes. The final formulation is

$$\min_{w, \xi_i, C} Ce^T \xi + \frac{1}{2} w^T w$$

(6.37)

Since the objective function is quadratic, find the solution through quadratic programming. The formulation can also be written as

$$\min_{w, \xi_i, b} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

subject to

$$t_i(w^T \phi(x_i) - b + \xi_i - 1 \geq 0, \quad \xi_i \geq 0, \quad 1 \leq i \leq N)$$

SVM requires three pieces of data $Q, t$ and $C$. $Q$, an $N \times N$ positive semidefinite matrix, is produced from $A * A^T$ and $t * t^T$.

$$AA^T = \begin{bmatrix} x_1^T x_1 & \cdots & x_1^T x_N \\ \vdots & \ddots & \vdots \\ x_N^T x_1 & \cdots & x_N^T x_N \end{bmatrix} = K$$

(6.39)
Note that $(i, j)^{th}$ element of $AA^T$ is $x_i^T x_j$. The matrix is called linear kernel matrix. This implies that all the information needed for training is captured in the form of dot products of the training vectors. If the data is not linearly separable, a function $\phi(.)$ may be used to map each data point $x_i$ into a higher dimensional space, and then try for maximally separating hyperplane in that space as a classifier. If such a separation is possible, SVM algorithm for training requires $\phi(x_i)^T \phi(x_i)$ for all $i$ and $j$. If such explicit mapping is done and the dot product is mapped in the higher dimensional space, then the computational requirement is huge especially when either dimension of the data or the number of data points is high. The kernel trick is used for obtaining dot products without explicitly mapping the data points into higher dimensional space. Then the SVM based nonlinear classifiers have the following advantages:

1) The training algorithm is the same as that of the linear classifier.
2) The problem of curse of dimensionality is tackled in a simple way.

A kernel function $K(x_i, x_j)$ is a function in input space. The basic advantage of using kernel function $K(x_i, x_j)$ is that it helps avoid a mapping $\phi(x)$ at all. Instead, the required scalar products in a feature space $\phi(x_i)^T \phi(x_i)$ are calculated directly by computing kernels $K(x_i, x_j)$ for given training data vectors in an input space. In this way, it is possible to bypass an extremely high dimensionality of a feature space $F$. Thus by using the chosen kernel $K(x_i, x_j)$, one can construct an SVM that operates in an infinite dimensional space. In its simple form, it is just a dot product between the input pattern $x_i$ and a member of the support set: $K(x_i, x_j) = x_i \cdot x_j$. A kernel is a function $K$ such that:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

In the present study, RBF kernel is used,

$$K(x, y) = \exp(-\gamma \|x - y\|^2)$$

where $\gamma$ is a positive parameter controlling the radius.

Apply Lagrangian to (6.38),

$$L(w, b, \xi, \mu) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \left[ y_i (w^T \phi(x_i) - b) + \xi_i - 1 \right] - \sum_{i=1}^{N} \mu_i \xi_i$$
Extremely Fast Machine Learning Algorithms

\[
\frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^{N} (C - u_i - \mu_i) \xi_i - \left( \sum_{i=1}^{N} u_i t_i \phi(x_i^T) \right) \mathbf{w} - \left( \sum_{i=1}^{N} u_i t_i \right) b + \sum_{i=1}^{N} u_i
\]

where \( \mathbf{u}, \mu \) are the Lagrange multipliers, and the Wolfe dual problem is:

\[
\max_{\mathbf{u}, \mu} L(\mathbf{w}, b, \xi, \mathbf{u}, \mu)
\]

subject to \( \frac{\partial L}{\partial \mathbf{w}} = 0, \frac{\partial L}{\partial b} = 0, \frac{\partial L}{\partial \xi_i} = 0, 1 \leq i \leq N, \mathbf{u} \geq 0, \mu \geq 0 \)

The constraint \( \frac{\partial L}{\partial \mathbf{w}} = 0 \) implies

\[
\mathbf{w} = \sum_{i=1}^{N} u_i d_i \phi(x_i)
\]

(6.42)

The constraint \( \frac{\partial L}{\partial b} = 0 \) implies

\[
\sum_{i=1}^{N} u_i t_i = 0
\]

(6.43)

The constraint \( \frac{\partial L}{\partial \xi_i} = 0 \) implies

\[
C - u_i - \mu_i = 0, \quad 1 \leq i \leq N
\]

(6.44)

Note that \( \mathbf{u} \geq 0, \mu \geq 0 \), and we have \( 0 \leq u_i \leq C \).

Substituting these results into \( L(\mathbf{w}, b, \xi, \mathbf{u}, \mu) \),

\[
L(\mathbf{w}, b, \xi, \mathbf{u}, \mu) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \mathbf{w}^T \mathbf{w} + \sum_{i=1}^{N} u_i
\]

\[
= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^{N} u_i
\]

\[
= \sum_{i=1}^{N} u_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \phi(x_i^T) \phi(x_j) u_i u_j
\]

The dual problem is

\[
\max_{\mathbf{u}} L(\mathbf{u}) = \sum_{i=1}^{N} u_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \phi(x_i^T) \phi(x_j) u_i u_j
\]

(6.45)

subject to \( \sum_{i=1}^{N} u_i t_i = 0, \quad 0 \leq u_i \leq C, 1 \leq i \leq N \)
This can be converted into a minimization problem by changing the sign of objective function. The matrix form of this formulation is

\[ \text{Minimize } L_r(u) = \frac{1}{2} u^T Q u - e^T u \quad \text{subject to } t^T u = 0, \; 0 \leq u \leq C e \]  

where \( e = [1 \ldots 1]^T \), \( Q = TK^T \) and \( T = \text{diag}(t) \).

After (6.46) is solved using primal dual relationship, the optimal \( w \) satisfies,

\[ w = \sum_{i=1}^{N} t_i u_i \phi(x_i) \]

and the decision function is,

\[ \text{sign}(w^T \phi(x) + b) = \text{sign}\left( \sum_{i=1}^{N} t_i u_i K(x_i, x) + b \right) \]

(6.47)

We store \( t_i u_i, \forall i, b \), label names, support vectors and other information such as kernel parameters in the model for prediction.

KKT conditions have the following three cases for \( u_i \):

(i) If \( u_i = 0 \), then \( \xi_i = 0 \). Thus \( x_i \) is correctly classified.

(ii) If \( 0 < u_i < C \), then \( t_i (w^T x_i - b) + \xi_i = 1 \) and \( \xi_i = 0 \). Thus \( x_i \) is an unbounded support vector. These are points falling on the bounding planes of the class to which the points belong.

(iii) If \( u_i = C \), then \( t_i (w^T x_i - b) + \xi_i = 1 \) and \( \xi_i > 0 \). Thus \( x_i \) is a bounded support vector. These are points falling outside the bounding plane of the respective classes to which the points belong. If \( 0 \leq \xi_i < 1 \), then \( x_i \) is correctly classified, and if \( \xi_i \geq 1 \), it is misclassified.

For computing \( b \), one of those points whose Lagrangian multiplier falls between 0 and \( C \) can be used. For these points

\[ w^T x_i - b = t_i \]

\[ \therefore b = w^T x_i - t_i = \sum_{j \in \text{svindex}} u_j t_j x_j^T x_i - t_i \]

(6.48)

It is customary to get \( b \) from all such points and find the average.

\[ b = \frac{1}{|\text{sv/index}|} \left[ \sum_{j \in \text{svindex}} \left( \sum_{i \in \text{svindex}} u_j t_j x_j^T x_i \right) - t_i \right] \]

(6.49)
where $sv$ is the set of indices of support vectors, and $svm$ is the set of indices of supports vectors that falls on the hyperplanes [165].

### 6.7 Multiclass Classification using SVM

There are two different approaches for multiclass classification by SVM – direct method and indirect method. In indirect method, several binary SVMs are constructed and combine the outputs to find the final class, whereas the direct approach considers all the data in a single optimization formulation. The formulation to solve multiclass SVM problems in one step has variables proportional to the number of classes. This approach constructs $k$ (number of classes) two class rules where the $p^{th}$ function separates training vectors of the class $p$ from other vectors. Hence, there are $k$ decision functions but all are obtained by solving one problem. This method consumes more training time. During testing, the data point is assigned the label of the decision rule that yielded the highest margin. Therefore, for multiclass SVM methods, either several binary classifiers have to be constructed or a larger optimization problem is needed.

Indirect methods are most widely used, and they are classified into three types: one against all (1-v-r), one against one (1-v-1), and DAGSVM. In one against all, a binary SVM classifier is constructed for each class discriminating the data points of that class against the rest. The $i^{th}$ SVM is trained with all the examples in the $i^{th}$ class with positive labels, and all other examples with negative labels. Given $N$ training data $(x_i, y_i), \ldots, (x_N, y_N)$, where $x_i \in \mathbb{R}^n$, $i = 1, \ldots, N$ and $y_i \in \{1, \ldots, k\}$ is the class of $x_i$, the $i^{th}$ SVM solves the following problem:

\[
\begin{align*}
\min_{w', b', \xi'} & \quad \frac{1}{2} (w')^T w' + C \sum_{j=1}^{N} \xi_j'(w')^T \\
\text{subject to} & \quad (w')^T \phi(x_j) + b' \geq 1 - \xi_j', \quad \text{if } y_j = i \\
& \quad (w')^T \phi(x_j) + b' \leq -1 + \xi_j', \quad \text{if } y_j \neq i \\
& \quad \xi_j' \geq 0, \quad j = 1, \ldots, N
\end{align*}
\]

After solving (6.50), there are $k$ decision functions, and $x$ is in the class with the largest value of the decision function

\[
\text{class of } x \equiv \arg \max_{i = 1, \ldots, k} ((w')^T \phi(x) + b')
\]

(6.51)
Practically solve the dual problem of (6.50) whose number of variables is the same as the number of data in (6.50). Hence, $k N$-variable quadratic programming problems are solved. In this method, there is no bound on the generalization error, and the training time also scales linearly with $k$. It cannot compete with 1-v-1 on training time.

One against one method constructs $k(k-1)/2$ classifiers where each one is trained on data from two classes. To train data from the $i^{th}$ and $j^{th}$ classes, the following binary classification problem can be solved:

$$
\min_{w_i^\parallel, b_i^\parallel, \xi_i^\parallel} \frac{1}{2} (w_i^\parallel)^T w_i^\parallel + C \sum_i \xi_i^\parallel (w_i^\parallel)^T
$$

$$(w_i^\parallel)^T \phi(x_i) + b_i^\parallel \geq 1 - \xi_i^\parallel, \quad \text{if } y_i = i
$$

$$(w_i^\parallel)^T \phi(x_i) + b_i^\parallel \leq -1 + \xi_i^\parallel, \quad \text{if } y_i = j
$$

$$\xi_i^\parallel \geq 0 \tag{6.52}$$

Practically solve the dual of (6.52) whose number of variables is the same as the number of data in two classes. If each class has $N/k$ data points, $k(k-1)/2$ quadratic programming problems are to be solved where each of them has about $2N/m$ variables [186]. There are different methods to conduct testing on the given data. Knerr [187] suggested combining these two class classifiers with an AND gate. Friedman [188] suggested a Max-Wins algorithm: each 1-v-1 classifier casts one vote for its preferred class, and the final result is the class with the most votes.

The disadvantage of 1-v-1 approach is that unless the individual classifiers are carefully regularized, the overall $m$ class classifier system will tend to overfit. The AND combination method and the Max-Wins combination method do not have bounds on the generalization error. Also, the size of 1-v-1 classifier may grow super linearly with $m$, and hence, may be slow to evaluate on large problems [184].

In the case of DAGSVM, training phase is same as one against one method, and testing phase uses Directed Acyclic Graph (DAG). Each node in a DAG is a binary classifier. For the three classes $H$, $E$, and $C$, the prediction structure is shown in Fig. 6.3. Given a test data, starting at the root node and the binary decision function of the classifier $H/E$ is evaluated. The node is then exited at the left edge if it does not belong to $H$, or right edge if it does not
belong to $E$. The next node’s decision function is then evaluated. Finally, the input reaches a leaf node that indicates the class label of the input data [165]. This method is applied in our pattern recognition problem, and it is described in the next section.

![Diagram of DAG SVM]

**Figure 6.3 Prediction using DAG SVM**

### 6.7.1 Decision directed acyclic graph

A multiclass learning architecture called Decision Directed Acyclic Graph (DDAG) is introduced in this section. VC analysis of DDAGs shows that the margins achieved at the decision nodes and the size of the graph both affect their performance, while the dimensionality of the input space does not. It indicates that building large margin DDAGs in high dimensional feature spaces can yield good generalization performance.

DDAG is a graph whose edges have an orientation and no cycles. A rooted DAG has a unique node such that it is the only node which has no arcs pointing into it. A rooted binary DAG has nodes which have either 0 or 2 arcs leaving them. We will use rooted binary DDAGs in order to define a class of functions to be used in classification tasks.

**Definition 6.3:** Given a space $X$ and a set of Boolean functions $\mathcal{F} = \{f: X \to \{0, 1\}\}$, the class $\text{DDAG}(\mathcal{F})$ of decision DDAGs on $k$ classes over $\mathcal{F}$ are functions which can be implemented using a rooted binary DAG with $k$ leaves labeled by the classes where each of the $k(k-1)/2$ internal nodes is labeled with an element of $\mathcal{F}$. The nodes are arranged in a triangle with the
single root node at the top, two nodes in the second layer and so on until the final layer of \( k \) leaves. The \( j^{th} \) node in layer \( j < k \) is connected to the \( j^{th} \) and \((i+1)^{th}\) node in the \((j+1)^{th}\) layer.

To evaluate a particular DDAG \( G \) on input \( x \in X \), starting at the root node and evaluate the binary function in each node. The value of the decision function is the value associated with the leaf node. The path taken through the DAG is known as the evaluation path. The input \( x \) reaches the node of the graph if that node is on the evaluation path of \( x \). The decision node distinguishing classes \( i \) and \( j \) are referred to as the \( ij \) node. Assuming that the number of a leaf is its class, this node is the \( i^{th} \) node in the \((k-j+1)^{th}\) layer provided \( i < j \). Similarly, the \( j \) nodes are those nodes involving class \( j \), i.e., the internal nodes on the two diagonals containing the leaf labeled by \( j \).

DDAG is equivalent to operating on a list of classes where each node eliminates one class from the list. A test point is evaluated against the decision node that corresponds to the first and last elements of the list. If the node prefers to one of the two classes, the other class is eliminated from the list, and the DDAG proceeds to test the first and the last elements of the new list. DDAG terminates when only one class remains in the list. For a problem with \( k \) classes, \( k-1 \) decision nodes will be evaluated in order to derive an answer.

The current state of the list is the total state of the system. Since a list state is reachable in more than one possible path through the system, the decision graph that the algorithm traverses is a DAG. DDAG naturally generalizes the class of decision trees allowing for a more efficient representation of redundancies and repetitions that can occur in different branches of the tree by allowing the merging of different decision paths [184].

### 6.7.2 DAGSVM algorithm

DAGSVM algorithm places 1-v-1 SVMs on the DDAG nodes and combines their results to find the class. Let \( w \) be the weight vector correctly splitting \( i \) and \( j \) classes at the \( ij \) node with threshold \( \theta \). The margin of \( ij \) node is defined as \( \delta = \min_{c(x)=i,j} \|w^T x - \theta\| \), where \( c(x) \) is the class associated to training example \( x \). Maximizing the margin of all nodes in a DDAG will minimize a bound on the generalization error. Therefore, a DDAG has to be created whose nodes are maximum margin classifiers over a kernel induced feature space. Such a DDAG is
obtained by training each $ij$ node only on the subset of training points labeled by $i$ or $j$. The DDAG partitions the input space into polytopic regions, each of which is mapped to a leaf node and assigned to a specific class.

DAGSVM operates the individual classes with large margin. It is safe to discard the losing class at each 1-v-1 decision because all of the examples of the losing class are far away from the decision surface. The choice of the class order in the list or DDAG is arbitrary. One possible heuristic is to place the most frequent classes at the centre of the list. If the DDAG decisions are random coin flips, then the distribution of class outputs would be binomial.

DAGSVM is superior to existing multiclass SVM algorithms in both training and evaluation time. Empirically, SVM training is observed to scale super linearly with the training set size $N$, according to a power law, $T_{\text{single}} = cN^\delta$, for $\delta = 2$ for algorithms that are based on the decomposition method, with some proportionality constant $c$. For 1-v-r multiclass SVM training algorithm, the entire training set is used to create all $k$ classifiers, hence the training time is $T_{1-v-r} = ckN^\delta$. Assuming that the classes have the same number of examples, training each 1-v-1 SVM only requires $2N/k$ training examples. Thus, training $k(k-1)/2$ 1-v-1 SVMs would require

$$T_{1-v-1} = crac{k(k-1)}{2} \left( \frac{2N}{k} \right)^\delta \approx 2^{\delta-1}ck^{2-\delta}N^\delta$$

(6.53)

For a typical case, where $\delta = 2$, the amount of time required to train all of the 1-v-1 SVMs is independent of $k$, and is only twice that of training a single 1-v-r SVM. Using 1-v-1 SVMs with a combination algorithm is thus preferred for training time. The number of kernel evaluations is a good indication of evaluation time, and it is the number of unique support vectors averaged over the evaluation paths through the DDAG taken by the test set [184]. To save testing time, $K(x, x)$ is calculated initially, and then used in several places of decision function.
Chapter 6

6.8 Conclusion

The different learning algorithms, ELM, OS-ELM and SVM, used for the present study are discussed in this chapter. They are based on two approaches in classification – neural networks and statistical learning theory. The general characteristic of these algorithms is that they are extremely fast either in training or in testing stage. While designing the classifier, neural network parameters need to be tuned iteratively. Here, two algorithms, ELM and OS-ELM, are presented for SLFN which use randomly assigned parameters. Hence, they are many times faster than conventional algorithms like BP. They are also able to provide better generalization performance. On the other hand, SVM is even faster than these algorithms in the training phase. For multiclass classification, our choice is DAGSVM which has shown good generalization performance in the recent past on various problems.