Chapter 6. Kernel based ANN classifiers, Radial Basis Function Networks and Support Vector Machines

6.1. Introduction:

Unlike multilayer perceptrons trained with the back-propagation algorithm, the design of Radial Basis Function (RBF) and Support Vector Machine (SVM) networks follows a principled approach. In particular, the construction of regularization theory in RBF provides sound mathematical formulation while SVM is based on the principle of Structural Risk Minimization [18].

Another principled approach for the design of RBF and SVM networks is via kernel regression method. This approach involves the use of density estimation, for which the radial basis functions sum to unity exactly. Multivariate Gaussian distribution provides a convenient method for satisfying this requirement. In this chapter, we study the use of kernel based ANN classifiers for the classification of Gujarati symbols. A notion that is central to the construction of the support vector learning algorithm is the inner-product kernel between a "support vector "$\mathbf{x}_i$" and the vector **X** drawn from the input space.

This chapter is divided in to six sections. In the second section that follows this introductory section, we discuss the computational aspects of Radial Basis Function in continuation to the introductory concepts provided in Chapter 1. The learning rules for weights, centers and spreads for the Radial Basis Function architecture are established in this section. The third section demonstrates the application of these rules for the classification of Gujarati symbols. We introduce Support Vector Machines of two-class and multi-class types in the fourth section. A new approach for multi-class Support Vector Machine classifiers which is uniform in its approach for two-class and multi-class problems is presented in the fifth section. The chapter ends with a section providing a summary and discussion of the methods.

6.2. Radial Basis Function Networks (RBFN)

As we have discussed in chapter 1, Radial Basis Function Networks can be characterized into two types, viz. Regularization Radial Basis Function and Generalized Radial Basis Function. As discussed in the section 1.4.2 of chapter 1, the unknown weights of the output layer of regularization Radial Basis Function networks can be computed by taking inverse of the Gaussian matrix **G**. But the drawback of this approach is that it is computationally very costly to find the inverse of **G** in the case of higher dimensional datasets. This problem can be overcome with the help of Generalized Radial Basis Functions, by considering fewer training patterns as centers of hidden layer. The following learning strategies are usually applied for Generalized Radial Basis Functions:

6.2.1 Learning Strategies

The learning process involved in the functioning of a radial-basis function network irrespective of its theoretical background, may be described as follows. The linear weights associated with the output units of the network tend to evolve faster compared to the parameters of the nonlinear activation functions of the hidden units. The hidden layer's activation functions evolve slowly in accordance with some nonlinear optimization strategy, while the weights of the links of output layer evolve faster according to some linear optimization strategy. The task of the hidden layer is to map the input patterns in to a high dimensional feature space while the task of the linear output layer is to perform the classification. Due to this separation of responsibilities, it is reasonable to separate the optimization of the hidden and output layers of the network by using different techniques, and perhaps by operating on different time scales.

There are different learning strategies that we can follow in the design of an RBF network, depending on how the centers of the radial-basis function of the network are specified. In this section we discuss three design strategies specifying design of RBF networks using interpolation theory [18].

a. Fixed Centers Selected at Random

The simplest approach is to assume *fixed* radial-basis functions defining the activation functions of the hidden units. The locations of the centers may be chosen *randomly* from the training data set. For the radial basis functions, we may employ Gaussian functions whose standard deviation is fixed according to the spread of the centers. Specifically, a normalized radial basis function centered at pattern t_i is defined as

$$G\left(\left\|\mathbf{X}-\mathbf{t}_i\right\|^2\right)=\exp\left(-\frac{m_1}{d_{\max}^2}\left\|\mathbf{X}-\mathbf{t}_i\right\|^2\right), \qquad i = 1,2,...,m_1$$

where X is the input vector, m_1 is the number of centers and d_{max} is the maximum of all distance between the pairs of chosen centers.

The standard deviation (i.e. width) of all the Gaussian radial basis functions is fixed at

$$\sigma=\frac{d_{\max}}{\sqrt{2m_1}} \qquad \text{(i)}$$

This formula ensures that the individual radial-basis functions are not too peaked or too flat; both of these two extreme conditions should be avoided. The only parameters that would need to be learned in this approach are the linear weights in the output layer of the network. A straightforward procedure for doing this is to use the *pseudoinverse* method discussed in chapter-1. Specifically, we have

$$\boldsymbol{W}=\mathbf{G}^{+}\boldsymbol{d}$$

where **d** is the vector of desired responses in the training set. The matrix $\boldsymbol{G}^{+}$ is the *pseudoinverse* of the matrix **G**, which is itself defined as

$$\mathbf{G} = \{g_{ji}\}$$

Where $$g_{ji}=\exp\left(-\frac{m_1}{d^2}\left\|\mathbf{X}_j-\mathbf{t}_i\right\|^2\right), \quad j = 1,2,..,N; \; i = 1,2,...,m_1$$

where X_j is the j^{th} input vector of the training sample. The norm in the above expression is usually taken as the Euclidean norm.

For the computation of *pseudoinverse* of a matrix the method of Singular-value decomposition (SVD) can be employed [18].

b. Self-Organized Selection of centers

The main problem with the method of fixed centers described in the earlier sub-section is the fact that it may require a large training set for a satisfactory level of performance. One way of overcoming this limitation is to use a *hybrid learning process*, consisting of two different stages:

- Self-organized learning stage, the purpose of which is to estimate appropriate locations for the centers of the radial basis functions in the hidden layer.
- Supervised learning stage, which completes the design of the network by estimating the linear weights of the output layer.

For the self-organized learning process we need a clustering algorithm that partitions the given set of data points into subgroups. One such algorithm is the k-means clustering algorithm [18], which places the centers of the radial-basis functions in only those regions of the input space χ where significant data are present. Let m_1 denote the number of radial-basis functions. Let $\{\mathbf{t}_k(n)\}_{k=1}^{m_1}$ denote the centers of the radial-basis functions at iteration n of the algorithm. Then, the k-means clustering algorithm proceeds as follows:

1. Initialization: Choose random values for the initial centers $\mathbf{t}_k(0)$; the only restriction is that these initial values be different. It may also be desirable to keep the Euclidean norm of the centers small.
2. Sampling: Draw a sample vector $\boldsymbol{X}$ from the input space χ with a certain probability. The vector $\boldsymbol{X}$ is input into the algorithm at iteration n.
3. Similarity matching: Let $k(\boldsymbol{X})$ denote the index of the best-matching (winning) center for input vector $\boldsymbol{X}$. Find $k(\boldsymbol{X})$ at iteration n by using minimum-distance Euclidean criterion:

$$k(\mathbf{X}) = \arg\min_{K} \|\mathbf{X}(n) - \mathbf{t}_k(n)\|, \quad k = 1,2,\ldots,m_1$$

where $\mathbf{t}_k(n)$ is the center of the k^{th} radial-basis function at iteration n.

4. Updating: Adjust the centers of the radial-basis functions, using the update rule:

$$\mathbf{t}_K(n+1) = \begin{cases} \mathbf{t}_k(n) + \eta[\mathbf{X}(n) - \mathbf{t}_k(n)], & k = k(\mathbf{X}) \\ \mathbf{t}_k(n), & \textit{otherwise} \end{cases}$$

where η is a learning-rate parameter that lies in the range $0 < \eta < 1$.

5. Continuation: Increment n by 1, go back to step 2, and continue the procedure until no noticeable changes are observed in the centers $\mathbf{t}_k$.

The k-means clustering algorithm just described is, in fact, a special case of a competitive (winner-take-all) learning process known as the self-organizing map.

c. Supervised Selection of Centers

In the third approach, the centers of the radial-basis functions and all other free parameters of the network undergo a supervised learning process, i.e., the RBF network takes on its most generalized form. A natural candidate for such a process is error-correction learning, which is most conveniently implemented using a gradient-descent procedure that represents a generalization of the LMS algorithm.

The first step in the development of such a learning procedure is to define the instantaneous value of the cost function.

$$\xi = \frac{1}{2}\sum_{j=1}^{N} \mathbf{e}_j^2 \qquad \text{(ii)}$$

where N is the size of the training sample used to do the learning, and e_j is the error signal defined by

$$e_j = d_j - F^*(X_j)$$

Using equation (1.27) for m_1 number of centers, the equation (ii) takes the form

$$\mathbf{e}_j = \mathbf{d}_j - \sum_{i=1}^{m_1} \mathbf{w}_i G\left(\left\|\mathbf{X}_j - \mathbf{t}_i\right\|_{C_i}\right) \qquad \text{(iii)}$$

$$\mathbf{e}_j = \mathbf{d}_j - \sum_{i=1}^{m_1} \mathbf{w}_i G\left((\mathbf{X}_j - \mathbf{t}_i)^T \Sigma_i^{-1} (\mathbf{X}_j - \mathbf{t}_i)\right)$$

where $\sum_i^{-1}$ is covariance matrix for the center i (as discussed in the chapter 1).

The requirement is to find the free parameters $\mathbf{w}_i$, $\mathbf{t}_i$, and $\sum_i^{-1}$, so as to minimize ξ. The results of this minimization are summarized in the table-6.1. The following points are noteworthy for the adaptation formulas for the linear weights of the output layer, positions and spreads of centers.

- The cost function ξ is convex with respect to the linear parameters $\mathbf{w}_i$, but non-convex with respect to the centers $\mathbf{t}_i$ and matrix $\sum_i^{-1}$.
- The update equations for $\mathbf{w}_i$, $\mathbf{t}_i$, and $\sum_i^{-1}$ are assigned different learning-rate parameters η_1, η_2, and η_3 respectively.
- Unlike the back-propagation algorithm, the gradient-descent procedure described in the table-6.1 for an RBF network does not involve error back-propagation.

Adaptation Formulas for the Linear Weights and the Positions and Spreads of Centers for RBF Networks*[18]

1. Learning weights (Output layer)

   In order to update weight vector $\mathbf{w}_i$, the gradient descent rule is applied as follows:

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) - \eta_1 \frac{\partial \xi(n)}{\partial \mathbf{w}_i(n)}, \qquad i = 1,2,\ldots,m_1$$

   where,

$$\frac{\partial \xi(n)}{\partial \mathbf{w}_i(n)} = \frac{\partial \xi(n)}{\partial \mathbf{e}_j(n)} \cdot \frac{\partial \mathbf{e}_j(n)}{\partial \mathbf{w}_i(n)}$$

$$\frac{\partial \xi(n)}{\partial \mathbf{w}_i(n)} = -\sum_{j=1}^{N} \mathbf{e}_j(n) G\left(\left\|\mathbf{X}_j - \mathbf{t}_i(n)\right\|_{C_i}\right) \qquad \text{(using equation iii)}$$

2. Positions of centers (hidden layer)

Positions of centers can be taught with the help of gradient descent rule. The corresponding gradient can be calculated as below:

$$\frac{\partial \xi(n)}{\partial \mathbf{w}_i(n)} = \frac{\partial \xi(n)}{\partial \mathbf{e}_j(n)} \cdot \frac{\partial \mathbf{e}_j(n)}{\partial G(\|\mathbf{X}_j - \mathbf{t}_i\|_{C_i})} \cdot \frac{\partial G(\|\mathbf{X}_j - \mathbf{t}_i\|_{C_i})}{\partial (\|\mathbf{X}_j - \mathbf{t}_i\|_{C_i})} \cdot \frac{\partial (\|\mathbf{X}_j - \mathbf{t}_i\|_{C_i})}{\partial \mathbf{t}_i}$$

$$\frac{\partial \xi(n)}{\partial \mathbf{t}_i(n)} = -\mathbf{w}_i(n) \sum_{j=1}^{N} \mathbf{e}_j(n) G'\left(\|\mathbf{X}_j - \mathbf{t}_i(n)\|_{C_i}\right) \frac{\partial (\|\mathbf{X}_j - \mathbf{t}_i\|_{C_i})}{\partial \mathbf{t}_i} \qquad \text{(iv)}$$

To be precise $\dfrac{\partial (\|\mathbf{X}_j - \mathbf{t}_i\|_{C_i})}{\partial \mathbf{t}_i} = \dfrac{\partial}{\partial t_i} \{[\mathbf{X}_j - \mathbf{t}_i]^T \Sigma_i^{-1} [\mathbf{X}_j - \mathbf{t}_i]\}$

Here, we require the derivative with respect a vector quantity. So we may proceed in a following way:

Vector quantities X_j and t_i will take the form $X_j = \{X_{j1}, X_{j2}, \ldots, X_{jQ}\}_{1xQ}$ and $t_i = \{t_{i1}, t_{i2}, \ldots, t_{iQ}\}_{1xQ}$ respectively, where Q is number of inputs to the network

$$\text{Therefore, } [X_j - t_i]^T = \begin{bmatrix} X_{j1} - t_{i1} \\ X_{j2} - t_{i2} \\ \vdots \\ X_{jQ} - t_{iQ} \end{bmatrix}_{Qx1}$$

Let $f(t_i) = [\mathbf{X}_j - \mathbf{t}_i]^T \Sigma_i^{-1} [\mathbf{X}_j - \mathbf{t}_i]$

$$= \sum_{r=1}^{Q} \sum_{p=1}^{Q} (x_{jr} - t_{ir}) \Sigma_{ir}^{-1} (x_{jp} - t_{ip})$$

$$\text{Therefore, } \frac{\partial}{\partial t_{ir}} (f(t_i)) = \frac{\partial}{\partial t_{ir}} \left\{ \sum_{r=1}^{Q} \sum_{p=1}^{Q} (x_{jr} - t_{ir}) \Sigma_{ir}^{-1} (x_{jp} - t_{ip}) \right\}$$

$$= 2\sum_{p=1}^{Q}\left(-\Sigma_{ir}^{-1}(x_{jp} - t_{ip})\right) \qquad (\text{for } p = r)$$

In a vector form the above equation can be written as

$$\frac{\partial}{\partial \mathbf{t}_i}(f(\mathbf{t}_i)) = -2\Sigma_i^{-1}(\mathbf{X}_j - \mathbf{t}_i)$$

Substituting the above expression in equation (iv), we get

$$\frac{\partial \xi(n)}{\partial \mathbf{t}_i(n)} = -\mathbf{w}_i(n)\sum_{j=1}^{N}\mathbf{e}_j(n)G'\left(\left\|\mathbf{X}_j - \mathbf{t}_i(n)\right\|_{C_i}\right)\left(-2\Sigma_i^{-1}(\mathbf{X}_j - \mathbf{t}_i)\right)$$

$$\frac{\partial \xi(n)}{\partial \mathbf{t}_i(n)} = 2\mathbf{w}_i(n)\sum_{j=1}^{N}\mathbf{e}_j(n)G'\left(\left\|\mathbf{X}_j - \mathbf{t}_i(n)\right\|_{C_i}\right)\Sigma_i^{-1}(\mathbf{X}_j - \mathbf{t}_i)$$

The updating rule for centers can be described as below:

$$\mathbf{t}_i(n+1) = \mathbf{t}_i(n) - \eta_2 \frac{\partial \xi(n)}{\partial \mathbf{t}_i(n)}, \qquad i = 1,2,\ldots,m_1$$

3. Spreads of centers (hidden layer)

In the hidden layer spread of centers can be learned with the similar approach of Gradient descent rule. The corresponding gradient term is computed as follows:

$$\frac{\partial \xi(n)}{\partial \Sigma_i^{-1}(n)} = \frac{\partial \xi(n)}{\partial \mathbf{e}_j(n)} \cdot \frac{\partial \mathbf{e}_j(n)}{\partial G(.)} \cdot \frac{\partial G(.)}{\partial \Sigma_i^{-1}}$$

$$= -\mathbf{w}_i(n)\sum_{j=1}^{N}\mathbf{e}_j(n)G'\left(\left\|\mathbf{X}_j - \mathbf{t}_i(n)\right\|_{C_i}\right)Q_{ji}$$

where, $Q_{ji}(n) = [\mathbf{X}_j - \mathbf{t}_i(n)][\mathbf{X}_j - \mathbf{t}_i(n)]^T$

Hence, the learning rule can be defined as shown below:

$$\sum_i^{-1}(n+1) = \sum_i^{-1}(n) - \eta_3 \frac{\partial \xi(n)}{\partial \sum_i^{-1}(n)}$$

*The term $\mathbf{e}_j(n)$ is the error signal of output unit j at time n. The term $G'(.)$ is the first derivative of the Green's function $G(.)$ with respect to its argument.

In the following section, we demonstrate the classification accuracies for the symbols of Gujarati script. Gujarati numerals (middle zone characters) and lower zone characters are classified using Regularization (number of patterns is equal to the number of centers) and generalized Radial Basis Function networks (number of centers are fewer than that of number of patterns) respectively. Two different networks are constructed, one is for numerals and the other is for lower zone characters. Java is used as a programming language for both the experiments.

6.3.(a) Classification of the symbols of Gujarati numerals:

Along with the paper [3], discussed in the fourth chapter of the thesis, we have made simultaneous attempt of identifying Gujarati numerals with Regularization Radial Basis Function, discussed in the chapter 1. Here also we have taken compressed image of the size 16x16 (256) using Daubechies D4 wavelets (discussed in the chapters 4), as an input to the network. Total 440 patterns of these numerals are considered for our experiment, out of which 200 patterns are taken in training set and the remaining 240 patterns are kept in the testing set. Being the regularization network, each pattern of the training set constitutes a center of the hidden neuron and hence form hidden layer for the network. 10 neurons (one for each numeral) are taken in the output layer of the network.

In order to compute weight vector **w** of equation (1.26) of chapter 1, inverse of the matrix **G** is to be multiplied with vector desired output **d**. But the inverse of $[\mathbf{G}]_{200x200}$ is computationally costly therefore the weights are to be learned by gradient descent (described in the 6.2.1) algorithm as we use in the case of MLP. Results are shown in table-1:

Table-1 Classification of Gujarati numerals

Input of the RBFN	Hidden units (centers)	Performance on testing set (%)
256 Daubechies coefficients	200	93.33

6.3(b) Classification of symbols of lower and middle zone symbols:

We have applied generalized Radial Basis Function networks for the classification of the glyphs of lower and middle zone symbols in the images of the Gujarati script. Two separate networks have been constructed for the symbols of both the zones. Along with the experiments presented using Multilayer Perceptron in the chapter 4, we have made simultaneous attempts of applying generalized Radial Basis Function networks for the identification of lower and middle zone symbols.

In the experiments presented here, we have constructed two networks: the first network is for lower zone symbols and the second network is for the middle zone symbols of Gujarati script (section 2.4). Each of these networks has 256 input neurons and as many output neurons as the number of classes (types) of the respective zone.

An implementation of the general RBF architecture that can be used to realize these two networks has been developed as a set of java classes. The main class RBF has three attributes which are objects of three classes 1. InputLayer, 2. HiddenLayer and 3. OutputLayer. Each of the three Layer classes contain an arry of objects of an appropriate neuron class, ie, 1. InputNeuron, 2. Radbas and 3. OutputNeuron types. All these neuron classes are sub-classes of a general class Neuron class. The links among all these layers are generated with the help of a Synapse class. The synaptic weighs of the links are updated using this class. The InputLayer class has attributes for an array of input neurons and input patterns etc. and methods like setInputPatterns,

setSynapse etc. The corresponding InputNeuron class has attributes for input patterns, testing patterns etc and methods for setting input and testing patterns and for computing the outputs. HiddenLayer class has attributes for radbas neurons, input synapse weight, output synapse weight, distance, centers etc and methods for computing forward pass, center updation, spread updation etc. The corresponding hidden neuron class has attributes output synapse, distance, spread etc and methods for computing distance, center updation etc. While the OutputLayer class has attributes computed output, desired output, weights etc and the methods for getting computed output for each neuron, updation of weight etc. The corresponding output neuron class has attributes computed output, desired output etc and methods for computing computed output, updation of weights etc.

In the first experiment, the lower zone characters are classified. There are four lower modifiers which are used very frequently in the Gujarati script as shown in the chapter 2. We have constructed a network with 256 input neurons at input to the network and 4 neurons (one for each symbol) in the output layer. The 256 input neurons stand for the extracted features of the images using Daubechieds D4 wavelets as discussed in the chapters 4 and 5. The java program of the experiment is implemented using general network of Radial basis functions. The experimental details can be given as below:

Out of total 336 patterns of all the four symbols of lower modifiers of Gujarati script, we have considered randomly selected 136 patterns for testing and the remaining 200 patterns for the training purpose. There are 40 patterns considered as the centers to the Generalized RBF network. The weights are updated by using the 1st learning rule among those described in the previous section. The distance used during the computation of the updated weights is the Mahalanobis distance which was discussed in the chapter 1(section 1.5). Initially, the spread (width) is chosen using the formula (i) in the previous section 6.2, which gave a starting value of 6 for the spread. Then, by evaluating the classification performance with various values of the spread, the optimal value of 10 was arrived at.

We have updated only synaptic weights using the first updation rule of the previous section. The network is allowed to train until the SSE (Sum squared error) goes under 0.07. The network has taken 2500 iterations to achieve that goal. The experimental results for the classification on test patterns are shown in the table-2:

Table 2. Classification of lower zone characters

Input of the RBFN	Hidden units (centers)	Total number of patterns	Performance on testing set (%)
256 Daubechies coefficients	40	200	95.58

Out of 136 testing patterns 130 are identified correctly and hence achieved good recognition accuracy of 95.58%. This accuracy is slightly less than the accuracy achieved by multilayer perceptron as discussed in the chapter 4.

In the next experiment we have tried to recognize middle zone symbols of the Gujarati script. Middle zone dataset, to be recognized by the network, is made up of images of 10 numerals, 34 consonants, 3 frequently used conjuncts and 5 independent vowels (totaling 52 glyphs). A total of 2986 printed Gujarati characters scanned in various fonts and sizes as described in the second section are collected. Each image is normalized to a matrix of 32 x 32 (1024) binary values representing the black and white pixels. 2011 of these images are taken as the training set and the remaining 975 are used for the testing set. The normalized images are subjected to the D4 wavelet transformation, as described in the section 3. 256 low-low coefficients of each image are used as feature vector for the glyph .

Several experiments with various numbers of hidden units were carried out to determine the optimal number of neurons in the hidden layer of the network. The network has 256 neurons in its input layer and the output layer was made up of 52 neurons (each neuron corresponding to one of the 52 symbols to be recognized).

We have made several experiments by updating weights and centers using the update rules mentioned in the previous section of the above section. But we could manage to get an accuracy of 58.71% only in this case. It is felt that new strategies for center and spread selection may have to be adopted to improve the accuracy.

In the next section we introduce another Artificial Neural Network architecture namely support vector machine.

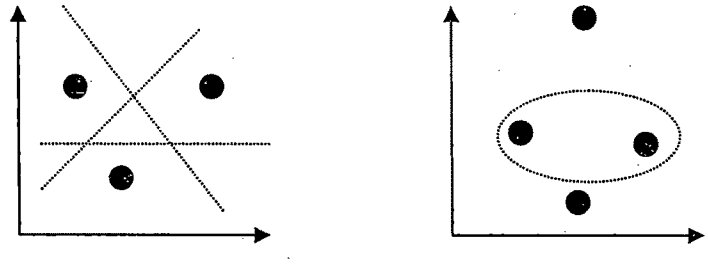
6.4. Introduction to Support Vector Machine

The MLP networks described in chapter 4 and the RBF networks described above are universal approximators in their own ways. In this chapter and the next, we discuss another category of feedforward networks, which are also universal approximators, known as support vector machines (SVM), pioneered by Vapnik (Boser, Guyon, and Vapnik, 1992; Cortes and Vapnik,1995;Vapnik 1995,1998). Like multilayer perceptrons and radial-basis function networks, support vector machines can be used for pattern classification and nonlinear regression. The formulation embodies the Structural Risk Minimization (SRM) principle (defined in the later part of this section), which has been shown to be superior, (Gunn et. al., 1997), to traditional Empirical Risk Minimization (Minimizing the error generated at the time of training in the output layer of the networks). This Structural Risk Minimization can broadly be introduced with the help of Vapnik-Chervonenkis (VC) dimension as below:

Vapnik-Chervonenkis dimension

In computational learning theory, the VC dimension (for Vapnik-Chervonenkis dimension) is a measure of the capacity of a statistical classification algorithm, defined as the cardinality of the largest set of points that the algorithm can shatter. A classification model f with some parameter vector θ is said to shatter a set of data points $(x_1, x_2, \ldots, x_n)$ if, for all assignments of labels to those points, there exists a θ such that the model f makes no errors when evaluating that set of data points.

The VC dimension has utility in statistical learning theory, because it can predict a probabilistic upper bound on the test error of a classification model. The VC dimension is a scalar value that measures the capacity of a set of functions.



[Fig.6.1 VC Dimension illustration]

Figure (6.1) illustrates how three points in the plane can be shattered by the set of linear indicator functions whereas four points cannot be. In this case the VC dimension is equal to the number of free parameters, but in general that is not the case; e.g. the function *Asin(bx)* has an infinite VC dimension (Vapnik, 1995). The set of linear indicator functions in n dimensional space has a VC dimension equal to $n + 1$.

For example, consider a straight line as the classification model: the model used by a perceptron. The line should separate positive data points from negative data points. When there are 3 points that are not collinear, the line can shatter them. However, the line cannot shatter four points. Thus, the VC dimension of this particular classifier is 3. It is important to remember that one can choose the arrangement of points, but then cannot change it as the labels on the points are permuted.

Structural Risk Minimisation

The challenge in solving a supervised learning problems is to realize the best generalization performance by matching the machine capacity to the available amount of training data for the problem at hand. The method of structural risk minimization provides an inductive procedure for achieving this goal by making the VC dimension of the learning machine a control variable (Vapnik, 1992, 1998). To be specific,

consider an ensemble of pattern classifiers $\{ F(\mathbf{X},\mathbf{W}); \mathbf{W} \in \mathcal{W} \}$ and define a nested structure of n such machines

$$\mathcal{F}_k = \{ F(\mathbf{X},\mathbf{W}); \mathbf{W} \in \mathcal{W}_k \}, k=1,2,...,n$$

such that we have

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset ... \subset \mathcal{F}_n$$

where $\subset$ signifies "is contained in". Correspondingly, the VC dimensions of the individual pattern classifiers satisfy the condition

$$h_1 \leq h_2 \leq ... \leq h_n$$

which implies that the VC dimension of each pattern classifier is finite. Then, the method of structural risk minimization may proceed as follows:

- The empirical risk (i.e. training error) for each pattern classifier is minimized.
- The pattern classifier $\mathcal{F}^*$ with the smallest guaranteed risk[18] is identified; this particular machine provides the best compromise between the training error (i.e., quality of approximation of the training data) and the confidence interval, (i.e., complexity of the approximating function) which compete with each other.

Our goal is to find a network structure such that decreasing the VC dimension occurs at the expense of the smallest possible increase in training error.

The principle of structural risk minimization may be implemented in a variety of ways. For example, we may vary the VC dimension h by varying the number of hidden neurons.

Basically, the support vector machine is a linear machine with some very nice properties. To explain how it works, it is perhaps easiest to start with the case of separable patterns that could arise in the context of pattern classification. In this context, the main idea of a support vector machine is to construct a hyperplane as the decision surface (chapter 1 section 1.4.2) in such a way that the margin of separation (discussed in the later part of this section) between positive and negative examples is maximized. The machine achieves this desirable property by following a principled approach rooted in the statistical learning theory that is discussed in Chapter 1. More

precisely, the support vector machine is an approximate implementation of the method of structural risk minimization. This induction principle is based on the fact that the error rate of a learning machine on test data (i.e., the generalization error rate) is bounded by the sum of the training-error rate and a term that depends on the Vapnik-Chervonenkis (VC) dimension; in the case of separable patterns, a support vector machine produces a value of zero for the first term and minimizes the second term. Accordingly, the support vector machine can provide a good generalization performance on pattern classification problems despite the fact that it does not incorporate problem-domain knowledge. This attribute is unique to support vector machines.

6.4.1. Two-class problems

Initially, the theory of support vector machines is developed for two class problems and then the theory of multiclass problem is developed using the same concept of two class problem of support vector machine. Here we discuss the basic theory of two class problems which are considered to be linearly separable.

6.4.1.1. Optimal hyperplane for linearly separable patterns

Consider the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ is the input pattern for the i^{th} example and d_i is the corresponding desired response (target output). To begin with, we assume that the patterns (class) represented by the subset $d_i = +1$ and the patterns represented by the subset $d_i = -1$ are "linearly separable." The equation of a decision surface in the form of a hyperplane that does the separation is

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{6.1}$$

where $\mathbf{x}$ is an input vector, $\mathbf{w}$ is an adjustable weight vector, and b is a bias. We may thus write

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b &\geq 0 \quad \text{for } d_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b &< 0 \quad \text{for } d_i = -1 \end{aligned} \tag{6.2}$$

The assumption of linearly separable patterns is made here to explain the basis idea behind a support vector machine in a rather simple setting; this assumption will be relaxed in Section 6.3.

For a given weight vector **w** and bias b, the separation between the hyperplane defined in Eq. (6.1) and the closest data point is called the margin of separation, denoted by ρ. The goal of a support vector machine is to find the particular hyperplane for which the margin of separation ρ is maximized. Under this condition, the decision surface is referred to as the optimal hyperplane.

Let $\mathbf{w}_0$ and b_0 denotes the optimum values of the weight vector and bias, respectively. Correspondingly, the optimal hyperplane, representing a multidimensional linear decision surface in the input space, is defined by

$$\mathbf{w}_0^{\mathrm{T}}\mathbf{x}+b_0=0 \tag{6.3}$$

which is a rewrite of Eq.(6.1) .The discriminant function

$$g(\mathbf{x})=\mathbf{w}_o^{\mathrm{T}}\mathbf{x}+b_o \tag{6.4}$$

gives an algebraic measure of the distance from **x** to the optimal hyperplane (Duda and Hart, 1973). Perhaps the easiest way to see this is to express x as

$$\mathbf{x}=\mathbf{x}_p+r\frac{\mathbf{w}_0}{\|\mathbf{w}_0\|}$$

where $\mathbf{x}_p$ is the normal projection of **x** onto the optimal hyperplane , and r is the desired algebraic distance; r is positive if **x** is on the positive side of the optimal hyperplane and negative side. Since, by definition , $g(\mathbf{x}_p)=0$, it follows that

$$g(\mathbf{x})=\mathbf{w}_0^{\mathrm{T}}+b_0=r\|\mathbf{w}_o\| \qquad \text{or}$$

$$\text{Therefore } r=\frac{g(\mathbf{x})}{\|\mathbf{w}_0\|} \tag{6.5}$$

In particular ,the distance from the origin (i.e., **x** = 0) to the optimal hyperplane is given by $b_0/\|\mathbf{w}_0\|$. If $b_0>0$, the origin is on the positive side of the optimal

hyperplane; if $b_0<0$, it is on the negative side. If $b_0=0$, the optimal hyperplane passes through the origin.

The issue at hand is to find the parameters $\mathbf{w}_0$ and b_0 for the optimal hyperplane, given the training set . The pair ($\mathbf{w}_0$, b_0) must satisfy the constraint:

$$\begin{aligned} \mathbf{w}_0^T \mathbf{x}_i + b_0 &\geq 1 \qquad \text{for } d_i = +1 \\ \mathbf{w}_0^T \mathbf{x}_i + b_0 &\leq -1 \qquad \text{for } d_i = -1 \end{aligned} \tag{6.6}$$

Note that if equation (6.2) holds, that is, the patterns are linearly separable, we can always rescale $\mathbf{w}_0$ and b_0 such that equation (6.6) holds; this scaling operation leaves equation (6.3) unaffected.

The particular data points $(\mathbf{x}_i, d_i)$ for which the first or second line of Eq.(6.6) is satisfied with the equality sign are called support vectors, hence the name "support vector for machine". These vectors play a prominent role in the operation of this class of learning machines. In conceptual terms, the support vectors are those data points that lie closest to the decision surface and are therefore the most difficult to classify. As such, they have a direct bearing on the optimum location of the decision surface.

Consider a support vector $\mathbf{x}^{(s)}$ for which $d^{(s)} = +1$. Then by definition, we have

$$\begin{aligned} g(\mathbf{x}^{(s)}) &= \mathbf{w}_0^T \mathbf{x}^{(s)} - b_0 = -1 \qquad \text{for } d^{(s)} = -1 \\ g(\mathbf{x}^{(s)}) &= \mathbf{w}_0^T \mathbf{x}^{(s)} + b_0 = +1 \qquad \text{for } d^{(s)} = +1 \end{aligned} \tag{6.7}$$

From equation (6.5) the algebraic distance from the support vector $\mathbf{x}^{(s)}$ to the optimal hyperplane is

$$r = \frac{g(\mathbf{x}^{(s)})}{\|w_o\|}$$

therefore r, can be written as

$$= \frac{1}{\|\mathbf{w}_0\|} \quad if \quad d^{(s)} = +1$$
$$= -\frac{1}{\|\mathbf{w}_0\|} \quad if \quad d^{(s)} = -1 \tag{6.8}$$

where the plus sign indicates that $\mathbf{x}^{(s)}$ lies on the positives side of the optimal hyperplane and the minus sign indicates that $\mathbf{x}^{(s)}$ lies on the negative side of the optimal hyperplane. Let ρ denote the optimum value of the margin of separation between the two classes that constitute the training set Ψ. Then, from equation (6.8) it follows that

$$\rho = 2r$$
$$= \frac{2}{\|\mathbf{w}_0\|} \tag{6.9}$$

Equation (6.9) states that maximizing the margin of separation between classes is equivalent to minimizing the Euclidean norm of the weight vector **w**.

In summary, the optimal hyperplane defined by equation (6.3) is unique in the sense that the optimum weight vector $\mathbf{w}_0$ provides the maximum possible separation between positive and negative examples. This optimum condition is attained by minimizing the Euclidean norm of the weight vector **w**

Quadratic optimization for Finding the optimal Hyperplane

Our goal is to develop a computationally efficient procedure for using the training sample set $\Psi = \{(\mathbf{x}_i, d_i)\}_{i=1}^N$ to find the optimal hyperplane, subject to the constraint

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \text{for } i = 1,2,\ldots,N \tag{6.10}$$

This constraint combines the two lines of Eq.(6.6) with **w** used in place of $\mathbf{w}_0$. The constrained optimization problem that we have to solve may now be stated as:

Given the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^N$, find the optimum values of the weight vector w and bias b such that they satisfy the constraints

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \text{for } i=1,2,...,N$$

and the weight vector **w** minimizes the cost function: $\varphi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ (*)

The scaling factor $\frac{1}{2}$ is included here for convenience of presentation. This constrained optimization problems is called the *primal problem*. It is characterized as follows:

→ The cost function $\varphi(\mathbf{w})$ is a convex function of **w** .

→ The constraints are linear in **w**.

Accordingly, we may solve the constrained optimization problems using the method of Lagrange multipliers (Bertsekas, 1995).

First, we construct the Lagrangian function:

$$J(\mathbf{w},b,\boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha_i[d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1] \qquad (6.11)$$

where the auxiliary nonnegative variables α_i are called Lagrange multipliers. The solution to the constrained optimization problem is determined by the saddle point of the Lagrangian function $J(\mathbf{w},b,\boldsymbol{\alpha})$, which has to be minimized with respect to **w** and b; it also has to be maximized with respect to $\boldsymbol{\alpha}$. Thus, differentiating $J(\mathbf{w},b,\boldsymbol{\alpha})$ with respect to **w** and b setting the results equal to zero, we get the following two conditions of optimality:

Condition 1: $$\frac{\partial J(\mathbf{w},b,\boldsymbol{\alpha})}{\partial \mathbf{w}} = 0$$

Condition 2: $$\frac{\partial J(\mathbf{w},b,\boldsymbol{\alpha})}{\partial b} = 0$$

Application of optimality condition 1 to the Lagrangian function of equation (6.11) yields (after rearrangement of terms)

$$\mathbf{w} = \sum_{i=1}^{N}\alpha_i d_i \mathbf{x}_i \qquad (6.12)$$

Application of optimality condition 2 to the Lagrangian function of eq.(6.11) yields

$$\sum_{i=1}^{N} \alpha_i d_i = 0 \tag{6.13}$$

The solution vector w is defined in terms of an expansion that involves the N training examples. Note, however, that although this solution is unique by virtue of the convexity of the Lagrangian, the same cannot be said about the Lagrange coefficients, α_i. We may use the inner product kernel to construct the optimal hyperplane in the feature space without having to consider the feature space itself in explicit form. The inner product kernel can be described as below:

Inner Product Kernel:

Let $\mathbf{X}$ be a vector drawn from input space with dimension m_0 and the corresponding set of nonlinear transformations from input space to feature space is $\{\varphi_j(\mathbf{X})\}_{j=0}^{m_1}$. It is assumed that $\varphi_j(\mathbf{X})$ is defined a priori for all j.

We define a hyper plane acting as the decision surface as follows:

$$\sum_{j=1}^{m_1} w_j \varphi_j(\mathbf{X}) + b = 0 \tag{6.14}$$

where $\{w_j\}_{j=0}^{m_1}$ denotes a set of linear weights connecting the feature space to the output space, and b is the bias.

For the sake of convenience, we take $\varphi_0(\mathbf{X}) = 1$ for all $\mathbf{X}$, so that w_0 denotes the bias b. Equation (6.14) will take the form of

$$\sum_{j=0}^{m_1} w_j \varphi_j(\mathbf{X}) = 0 \tag{6.15}$$

Equation (6.15) defines the decision surface computed in the feature space in terms of the linear weights of the machine.

Define the vector point function $\varphi(\mathbf{X}) = \left[\varphi_0(\mathbf{X}) \quad \varphi_1(\mathbf{X}) \quad \ldots \quad \varphi_{m_1}(\mathbf{X})\right]^T$ where due to bias $\varphi_0(\mathbf{X}) = 1$ for all $\mathbf{X}$.

In vector form the above equation can be written as $\mathbf{w}^T\varphi(\mathbf{X}) = 0$. (6.16)

Therefore equation (6.12) can be written as $\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \varphi(\mathbf{x}_i)$ (6.17)

where the feature vector $\varphi(\mathbf{X}_i)$ corresponds to the input X_i in the i^{th} example.

Using equations (6.17) in (6.16), we get

$$\sum_{i=1}^{N} \alpha_i d_i \varphi^T(\mathbf{x}_i)\varphi(\mathbf{x}_i) = 0 \tag{6.18}$$

The term $\varphi^T\varphi$ represents the inner product of two vectors induced in the feature space by the input vector $\mathbf{X}$ and the input pattern $\mathbf{X}_i$ pertaining to the i^{th} example.

The inner product kernel $K(\mathbf{X},\mathbf{X}_i)$ defined by

$$K(\mathbf{X},\mathbf{X}_i) = \varphi^T(\mathbf{X})\varphi(\mathbf{X}_i)$$

$$K(\mathbf{X},\mathbf{X}_i) = \sum_{j=1}^{m_1} \varphi_j(\mathbf{X})\varphi_j(\mathbf{X}_i) \quad \text{for i} = 1,2,\ldots,N \tag{6.19}$$

Using equation (6.19) in equation (6.18), we get

$$\sum_{i=1}^{N} \alpha_i d_i K(\mathbf{X},\mathbf{X}_i) = 0.$$

Consider the following 2-class problem in which our aim is to find the equation of the optimal hyper plane which discriminates the given pattern. The optimal hyper plane is obtained by using two different kernels.

XOR problem:

Input Vector (X)	Desired response (d)
(-1,-1)	-1
(-1,1)	+1
(1,-1)	+1
(1,1)	-1

[a] Polynomial kernel:

Considering the kernel $K(\mathbf{X},\mathbf{X}_i)=(1+\mathbf{X}^T\mathbf{X}_i)^2$

$$=\left(1+\begin{bmatrix}x_1 & x_2\end{bmatrix}\begin{bmatrix}x_{i1}\\ x_{i2}\end{bmatrix}\right)^2$$

$$=(1+x_1x_{i1}+x_2x_{i2})^2$$

Therefore, $K(\mathbf{X},\mathbf{X}_i)=1+x_1^2x_{i1}^2+x_2^2x_{i2}^2+2x_1x_{i1}+2x_2x_{i2}+2x_1x_2x_{i1}x_{i2}$

The image of the input vector **X** induced in feature space is therefore deduced to be

$$\varphi(X)=\left[1,\ x_1^2,\ \sqrt{2}x_1x_2,\ x_2^2,\ \sqrt{2}x_1,\ \sqrt{2}x_2\right]^T$$

Similarly $\varphi(X_i)=\left[1,\ x_{i1}^2,\ \sqrt{2}x_{i1}x_{i2},\ x_{i2}^2,\ \sqrt{2}x_{i1},\ \sqrt{2}x_{i2}\right]^T$ for $i=1,2,3,4$

Now, kernel $K=\{K(\mathbf{X}_i,\mathbf{X}_j)\}_{j=1}^{N=4}$

$$K=\begin{bmatrix}9 & 1 & 1 & 1\\ 1 & 9 & 1 & 1\\ 1 & 1 & 9 & 1\\ 1 & 1 & 1 & 9\end{bmatrix}$$

The objective function

$$Q(\alpha)=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j K(\mathbf{X}_i,\mathbf{X}_j) \qquad (6.20)$$

$$Q(\alpha)=\alpha_1+\alpha_2+\alpha_3+\alpha_4$$
$$-\frac{1}{2}\left(9\alpha_1^2-2\alpha_1\alpha_2-2\alpha_1\alpha_3+2\alpha_1\alpha_4+9\alpha_2^2+2\alpha_2\alpha_3-2\alpha_2\alpha_4+9\alpha_3^2-2\alpha_3\alpha_4+9\alpha_4^2\right)$$

Differentiating above objective function with respect to α_1, α_2, α_3 and α_4 respectively, we get following simultaneous equations:

$$\begin{aligned}
9\alpha_1-\alpha_2-\alpha_3+\alpha_4&=1\\
-\alpha_1+9\alpha_2+\alpha_3-\alpha_4&=1\\
-\alpha_1+\alpha_2+9\alpha_3-\alpha_4&=1\\
\alpha_1-\alpha_2-\alpha_3+9\alpha_4&=1
\end{aligned}$$

Solving the above simultaneous linear equations, we get $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 1/8$

In order to obtain the optimum value of the objective function $Q(\alpha)$, we denote the above obtained solution as

$$\alpha_{o1} = \alpha_{o2} = \alpha_{o3} = \alpha_{o4} = 1/8$$

All α_{oi}'s are equal that indicates that in this example all four input vectors $\{\mathbf{X}_i\}_{i=1}^4$ are support vectors.

Therefore, from equation (6.20) the optimal value of the objective function $Q(\alpha)$ is $Q_o(\alpha) = 1/4$.

From equation (6.17), the optimal weight vector can be written as

$$w_0 = \sum_{i=1}^{N} \alpha_{oi} d_i \varphi(\mathbf{X}_i)$$

$$w_o = \frac{1}{8}[-\varphi(\mathbf{X}_1) + \varphi(\mathbf{X}_2) + \varphi(\mathbf{X}_3) - \varphi(\mathbf{X}_4)]$$

$$\mathbf{w}_o = \frac{1}{8}\left[-\begin{bmatrix} 1 \\ 1 \\ \sqrt{2} \\ 1 \\ -\sqrt{2} \\ -\sqrt{2} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -\sqrt{2} \\ 1 \\ -\sqrt{2} \\ \sqrt{2} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -\sqrt{2} \\ 1 \\ \sqrt{2} \\ -\sqrt{2} \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ \sqrt{2} \\ 1 \\ \sqrt{2} \\ \sqrt{2} \end{bmatrix} \right]$$

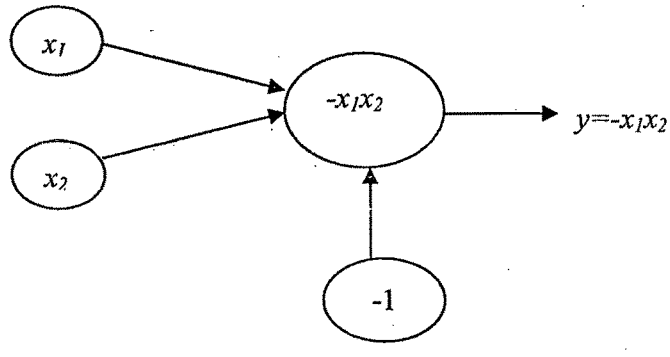
$$\mathbf{w}_o = \begin{bmatrix} 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \end{bmatrix}^T$$

From equation (6.16), the Support Vector Machine can be constructed as follows:

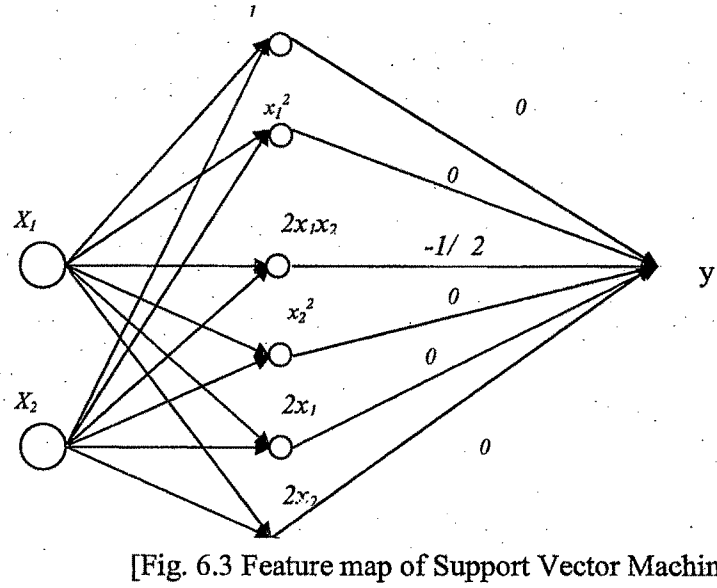
$$\begin{bmatrix} 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & x_1^2 & \sqrt{2}x_1x_2 & x_2^2 & \sqrt{2}x_1 & \sqrt{2}x_2 \end{bmatrix}^T = 0$$

$$\Rightarrow \quad -x_1x_2 = 0 \tag{6.21}$$

Using equation (6.21), the computed output for each input pattern becomes identical as the desired output.



[Fig. 6.2 Support Vector Machine]



[Fig. 6.3 Feature map of Support Vector Machine]

Figures 6.2 and 6.3 demonstrate the support vector machine and feature map of support vector machine respectively of the XOR problem.

Now we try to construct Support Vector Machine of the same problem by applying Gaussian kernel and observe the difference.

[b] Gaussian kernel:

Consider $K(\mathbf{X}, \mathbf{X}_i) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{X}-\mathbf{X}_i\|^2\right)$ with

$$\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2]^T \text{ and } X_i = [x_{i1}, x_{i2}]^T$$

The image of the input vector X induced in feature space is therefore deduced to be

$$\varphi(\mathbf{X}) = [G(\mathbf{X}, \mathbf{X}_1), G(\mathbf{X}, \mathbf{X}_2), G(\mathbf{X}, \mathbf{X}_3), G(\mathbf{X}, \mathbf{X}_4)]^T$$

Taking spread, $\sigma = 1$, we get the Gaussian matrix as follows:

$$K = \begin{bmatrix} 1 & 0.0183 & 0.0183 & 0.0183 \\ 0.0183 & 1 & 0.0183 & 0.0183 \\ 0.0183 & 0.0183 & 1 & 0.0183 \\ 0.0183 & 0.0183 & 0.0183 & 1 \end{bmatrix}$$

Solving equation (6.20) for $N = 4$ patterns using Gaussian kernel, we get the optimal value of Lagrangian coefficients as

$$\alpha_{o1} = \alpha_{o2} = \alpha_{o3} = \alpha_{o4} = 1.038$$

All the four points are support vectors.

Now using equation (6.17), we get the weight vector as follows:

$$\mathbf{w}_o = [-1 \quad 1 \quad 1 \quad -1]^T$$

Using equation (6.16), the Support Vector Machine can be constructed as follows:

$$(-1)G(\mathbf{X},\mathbf{X}_1) + (1)G(\mathbf{X},\mathbf{X}_1) + (1)G(\mathbf{X},\mathbf{X}_1) + (-1)G(\mathbf{X},\mathbf{X}_1) = 0$$

Using equation (6.21), the computed output for each input pattern can be expressed as follows:

Input Vector (X)	Desired response (d)	Computed response
(-1,-1)	-1	-0.9634
(-1,1)	+1	0.9634
(1,-1)	+1	0.9634
(1,1)	-1	-0.9634

By taking appropriate value of σ, we may get more accuracy.

6.4.2. Multi-class SVM problem

There is more than one way to device multi-class problem classifiers employing linear discriminant function. For example, we might reduce the problem to c two-class problems, where the i^{th} problem is solved by a linear discriminant function that separates points assigned to $\mathbf{w}_i$ from those not assigned to $\mathbf{w}_i$. This approach is called one to one approach. A more extravagant approach would be to use $c(c-1)/2$ linear discriminants, one for every pair of class. This technique is known as *one to all* approach. But both of these approaches can lead to regions in which the classification is undefined [39].

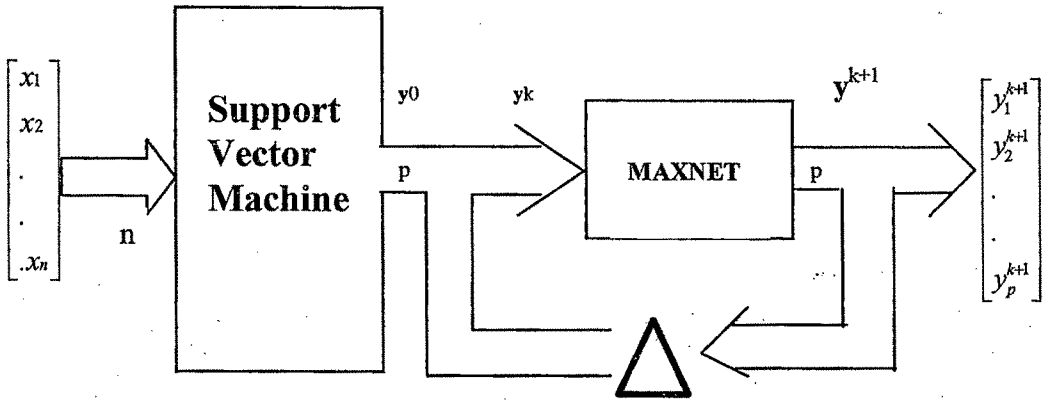
The new approach of obtaining the hyper planes of multiclass problems can be introduced using Maxnet network as shown below:

6.5. Multiclass Support Vector Machine using MAXNET

The new approach of classifying multiclass problems is divided in to two layers viz. Support Vector Machine network and second is MAXNET network. Both the layers can be characterized as below:

The Support Vector Machine is of feed forward type and constitutes the 1st layer of the classifier. The p-class SVM network has p output neurons. The strongest response of a neuron is indicative of the distance value between the input and the category this neuron represents.

The second layer of the classifier is called MAXNET and it operates as a recurrent recall network in an auxiliary mode. Its only function is to suppress value at MAXNET output nodes other than the initially maximum output node of the first layer. The block diagram of the MAXNET SVM classifier is depicted below:



[Fig. 6.4. Block diagram of the MAXNET SVM classifier]

Consider the training sample $\{(\mathbf{x}_i, \mathbf{Z}_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ is the input pattern for the i^{th} example out of total N patterns and $\mathbf{Z}_i = \{z_{ji}\}_{j=1}^{K}$ is the corresponding desired response (target output) out of total K classes.

To begin with *'one verses all'* approach, we assume that if the j^{th} output of the i^{th} pattern (class) is represented by the $z_{ji} = +1$ then the i^{th} pattern is said to be correctly identified and all other patterns for which $z_{ji} = -1$ are said to be incorrectly identified. We assume that the correct pattern is "linearly separable" than the incorrect patterns. The equation of a decision surface for k^{th} class in the form of a hyperplane that does the separation is

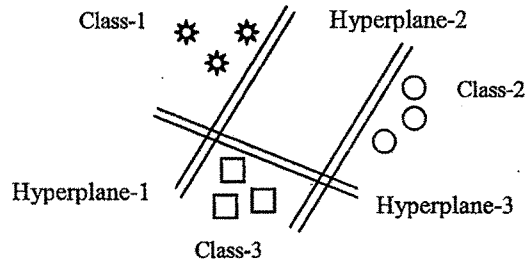
$$\mathbf{w}_k^T \varphi(\mathbf{x}) + b_k = 0 \tag{6.22}$$

where $\mathbf{x}$ is an input vector, $\mathbf{w}_k$ is an adjustable weight vector, and b_k is a bias for k^{th} class. We may thus write

$$\left.\begin{array}{lll} \mathbf{w}_k^T \varphi(\mathbf{x}_i) + b_k \geq 0 & \text{for} & z_{ki} = +1 \\ \mathbf{w}_k^T \varphi(\mathbf{x}_i) + b_k \prec 0 & \text{for} & z_{ki} = -1 \end{array}\right\} \tag{6.23}$$

The discriminant function, as defined in the equation (6.4) for a two class problem, can be written for a multiclass problem as follows:

$$g(\mathbf{x}) = \mathbf{w}_{ok}^T + b_{ok} \tag{6.24}$$



[Fig. 6.4a Separation of the three classes by three hyperplanes]

As shown in the Fig. 6.4a, for each class one hyperplane may be obtained which discriminate one class from the other classes.

Perhaps the easiest way to see this is to express **x** as

$$\mathbf{x} = \mathbf{x}_p + r \frac{w_{ok}}{\|w_{ok}\|}$$

where $\mathbf{x}_p$ is a normal projection of x on to the optimal hyperplane, and r is the algebraic distance as defined for the 2-class problem.

The issue at hand is to find the parameters w_{0k} and b_{0k} for the optimal hyperplane. The pair $(\mathbf{w}_{0k}, b_{0k})$ must satisfy the constraint:

$$\begin{aligned} \mathbf{w}_{ok}^T \varphi(\mathbf{x}_i) + b_{ok} &\geq 1 \qquad \text{for} \quad z_{ki} = +1 \\ \mathbf{w}_{ok}^T \varphi(\mathbf{x}_i) + b_{ok} &< -1 \qquad \text{for} \quad z_{ki} = -1 \end{aligned} \tag{6.25}$$

Note that if Eq.(6.2) holds, that is, the patterns ate linearly separable, we can always rescale $\mathbf{w}_0$ and b_0 such that Eq. (6.6) holds; this scaling operation leaves equation (6.3) unaffected.

Quadratic optimization for Finding the optimal Hyperplane for kth class

The constrained optimization problem, as discussed in the subsection (6.3.1), that we have to solve may now be stated as:

Given the training sample $\{(\mathbf{x}_i, z_{ki})\}_{i=1}^N$, find the optimum values of the weight vector $\mathbf{w}_k$ and bias b_k such that they satisfy the constraints

$$z_{ki}(\mathbf{w}_k^T \varphi(\mathbf{x}_i) + b_k) \geq 1 \qquad \text{for } i=1,2,\ldots,N$$

and the weight vector w minimizes the cost function: $\phi(\mathbf{w}_k) = \frac{1}{2}\mathbf{w}_k^T\mathbf{w}_k$ (6.26)

The scaling factor $\frac{1}{2}$ is included here for convenience of presentation. This constrained optimization problems is called the *primal problem*. It is characterized as follows:

→ The cost function $\phi(\mathbf{w}_k)$ is a convex function of $\mathbf{w}_k$.

→ The constraints are linear in $\mathbf{w}_k$.

Accordingly, we may solve the constrained optimization problems using the method of Lagrange multipliers (Bertsekas, 1995).

First, we construct the Lagrangian function:

$$J(\mathbf{w}_k, b_k, \alpha_k) = \frac{1}{2}\mathbf{w}_k^T\mathbf{w}_k - \sum_{i=1}^{N}\alpha_{ki}[z_{ki}(\mathbf{w}_k^T\varphi(\mathbf{x}_i) + b_k) - 1] \quad (6.27)$$

where the auxiliary nonnegative variables α_{ki} are called Lagrange multipliers for the k^{th} class. Thus, differentiating equation (6.27) with respect to $\mathbf{w}_k$ and b_k setting the results equal to zero, we get the following two conditions of optimality:

Condition 1: $$\frac{\partial J(\mathbf{w}_k, b_k, \alpha_k)}{\partial \mathbf{w}_k} = 0$$

Condition 2: $$\frac{\partial J(\mathbf{w}_k, b_k, \alpha_k)}{\partial b_k} = 0$$

Application of optimality condition-1 to the Lagrangian function of eq.(6.27) yields (after rearrangement of terms)

$$\mathbf{w}_k = \sum_{i=1}^{N}\alpha_{ki}z_{ki}\varphi(\mathbf{x}_i) \quad (6.28)$$

Applicant of optimality condition-2 to the Lagrangian function of eq.(6.28) yields

$$\sum_{i=1}^{N}\alpha_{ki}z_{ki} = 0 \quad (6.29)$$

As discussed in the subsection (6.3.1), the dual problem of equation (6.27) can be obtained by expanding it term by term

From equation(6.24),

$$J(\mathbf{w}_{ok}, b_k, \alpha_k) = \frac{1}{2}\mathbf{w}_k^T\mathbf{w}_k - \sum_{i=1}^{N}\alpha_{ik}z_{ik}\mathbf{w}_k^T\varphi(\mathbf{x}_i) - b_k\sum_{i=1}^{N}\alpha_{ik}z_{ik} + \sum_{i=1}^{N}\alpha_{ik}$$

Using the constraints 1 and 2 of this section, we get

Therefore, $J(\mathbf{w}_{ok}, b_k, \boldsymbol{\alpha}_k) = \sum_{i=1}^{N} \alpha_{ik} - \frac{1}{2}\sum_i \sum_j \alpha_{ki}\alpha_{kj} z_{ki} z_{kj} \varphi(\mathbf{x}_j)\varphi(\mathbf{x}_{k_1})$

$$J(\mathbf{w}_{ok}, b_k, \boldsymbol{\alpha}_k) = \sum_{i=1}^{N} \alpha_{ik} - \frac{1}{2}\sum_{i=j=1}^{N} \alpha_{ki}\alpha_{kj} z_{kj} z_{kj} K(\mathbf{x}_j, \mathbf{x}_k)$$

We may now state the dual problem as follows:

$$J(\mathbf{w}_{ok}, b_k, \boldsymbol{\alpha}_k) = \sum_{i=1}^{N} \alpha_{ik} - \frac{1}{2}\sum_{i=j=1}^{N} \alpha_{ki}\alpha_{kj} z_{kj} z_{kj} K(\mathbf{x}_j, \mathbf{x}_k) \tag{6.30}$$

subject to the constraints

(i) $\sum_{i=1}^{N} \alpha_{ki} z_{ki} = 0$

(ii) $\alpha_{ki} \geq 0$

Having determined the optimum Lagrange multipliers, denoted by α_{oki}, we may compute the optimum weight vector w_{ok} using the formula (6.28), we get

$$\mathbf{w}_{ok} = \sum_{i=1}^{N} \alpha_{oki} z_{ki} \varphi(\mathbf{x}_i)$$

For instance, suppose we have a k class problems for instance k=3.

Then the dual problem may be written from equation (6.30) as

$$L(\boldsymbol{\alpha}_{k_1}) = \sum_{i=1}^{n} \alpha_{k_1 i} - \frac{1}{2}\sum_{k,j}^{n} \alpha_{k_1 k}\alpha_{k_1 j} z_{k_1 k} z_{k_1 j} K(\mathbf{X}_k, \mathbf{X}_j)$$

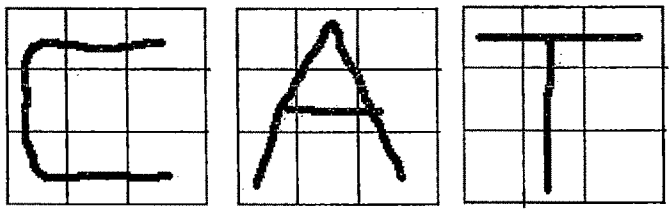
Subject to $\sum_{k=1}^{n} Z_{k_1 k}\alpha_{k_1 k} = 0$, $\alpha_{k_1 k} \geq 0$

Where n = number of patterns = 3

$k = 1, 2, \ldots, n$

k_1 = output neuron =9

$z_{k_1 k} = k_1^{th}$ output neuron of k^{th} pattern



Consider input vectors of three characters ***C***, ***A*** and ***T***

$\boldsymbol{C} = [1,1,1,1,-1,-1,1,1,1]^T$

$\boldsymbol{A} = [-1,1,-1,1,-1,-1,1,-1,1]^T$

$\boldsymbol{T} = [1,1,1,-1,1,-1,-1,1,-1]^T$

The corresponding output vectors are

$\boldsymbol{C_o} = [-1,-1,1]^T$, $\boldsymbol{A_o}=[-1,1,-1]^T$, $\boldsymbol{T_o}=[1,-1,-1]^T$ respectively.

In this example $k_I=\{1,2,3\}$

Take $k_I=1$

$$L(\alpha_1) = \sum_{i=1}^{n} \alpha_{1i} - \frac{1}{2}\sum_{k,j}^{n} \alpha_{1k}\alpha_{1j} z_{1k} z_{1j} K(\mathbf{X}_k, \mathbf{X}_j)$$

Subject to $\sum_{k=1}^{n} z_{1k}\alpha_{1k} = 0$, $\alpha_{1k} \geq 0$

Let $K(\mathbf{X},\mathbf{Y}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{X}-\mathbf{Y}\|^2\right)$ (6.31)

For $\sigma = 1$, the Gaussian matrix K is given by

$$K(\mathbf{X},\mathbf{Y}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$L(\alpha_1) = \alpha_{11} + \alpha_{12} + \alpha_{13}$$

$$-\frac{1}{2}\left[\alpha_{11}^2(-1)(-1)1 \;+\; \alpha_{11}\alpha_{12}(-1)(-1)0 \;+0+0+\; \alpha_{12}^2(-1)(-1) \;+0+0+0+0+\alpha_{13}^2\right]$$

$$\frac{\partial L}{\partial \alpha_{11}} = 1 - \alpha_{11} = 0 \qquad \Rightarrow \alpha_{11} = 1$$

$$\frac{\partial L}{\partial \alpha_{12}} = 1 - \alpha_{12} = 0 \qquad \Rightarrow \alpha_{12} = 1$$

$$\frac{\partial L}{\partial \alpha_{13}} = 1 - \alpha_{13} = 0 \qquad \Rightarrow \alpha_{13} = 1$$

for the class $k_1=1$ all the three patterns are Support Vectors. Proceeding in the same way for the remaining classes $k_1=2$ & $k_1=3$, we get

$$\alpha_{21} = \alpha_{22} = \alpha_{23} = 1$$

$$\alpha_{31} = \alpha_{32} = \alpha_{33} = 1$$

Thus for all the classes all the three patterns are support vectors. Using these support vectors for each class we calculate optimized weight vector as follows:

$$\mathbf{w}_{01} = \sum_{i=1}^{N=3} \alpha_{1i} z_{1i} \varphi(\mathbf{X}_i)$$

$$= 1\left[-\begin{bmatrix}1\\0\\0\end{bmatrix} - \begin{bmatrix}0\\1\\0\end{bmatrix} + \begin{bmatrix}0\\0\\1\end{bmatrix}\right]$$

$$= \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

Proceeding in the same way for the rest of the patterns, we get the optimized weight vectors for second the third class as follows:

$$\mathbf{w}_{02} = \sum_{i=1}^{N=3} \alpha_{2i} z_{2i} \varphi(\mathbf{X}_i)$$

$$= 1\left[-\begin{bmatrix}1\\0\\0\end{bmatrix} + \begin{bmatrix}0\\1\\0\end{bmatrix} - \begin{bmatrix}0\\0\\1\end{bmatrix}\right]$$

$$= \begin{bmatrix}-1\\1\\-1\end{bmatrix}$$

$$\mathbf{w}_{03} = \sum_{i=1}^{N=3} \alpha_{3i} z_{3i} \varphi(\mathbf{X}_i)$$

$$= 1\left[\begin{bmatrix}1\\0\\0\end{bmatrix} - \begin{bmatrix}0\\1\\0\end{bmatrix} - \begin{bmatrix}0\\0\\1\end{bmatrix}\right]$$

$$= \begin{bmatrix}1\\-1\\-1\end{bmatrix}$$

Now for class $k_1=1$, the support vector machine can be given by using the equation (6.16)

$$\mathbf{w}_{01}^{T} \varphi(\mathbf{X}) = 0$$

From the definition of $\varphi(\mathbf{X})$, discussed in the section 6.4, we can write

$$(-1)G(\mathbf{X},\mathbf{X}_1) + (-1)G(\mathbf{X},\mathbf{X}_1) + (1)G(\mathbf{X},\mathbf{X}_1) = 0 \qquad (6.32)$$

Proceeding in the same for other two classes $k_1=2$ and $k_1=3$, we get the support vector machines given as follows:

$$(-1)G(\mathbf{X},\mathbf{X}_1) + (1)G(\mathbf{X},\mathbf{X}_1) + (-1)G(\mathbf{X},\mathbf{X}_1) = 0 \qquad (6.33)$$

$$(1)G(\mathbf{X},\mathbf{X}_1) + (-1)G(\mathbf{X},\mathbf{X}_1) + (-1)G(\mathbf{X},\mathbf{X}_1) = 0 \qquad (6.34)$$

For the validation purpose, we take a new pattern as shown below:

$$\boldsymbol{C}_1 = [1,1,-1,1,-1,-1,1,1,1]^T$$

The corresponding values of the Gaussian kernel defined in the equation (6.31) are as under:

$$G(\boldsymbol{C}_1,\boldsymbol{X}_1) = 0.0183,\ G(\boldsymbol{C}_1,\boldsymbol{X}_2) = 0.0,\ G(\boldsymbol{C}_1,\boldsymbol{X}_3) = 0.0$$

Using the values mentioned above the support vector machines (6.32), (6.33) and (6.34) yield the following output vector:

$$\begin{bmatrix}-0.0183\\-0.0183\\0.0183\end{bmatrix}$$

Applying MAXNET network, we can get the solution. The MAXNET network can be introduced as shown below [19]:

MAXNET network:

MAXNET needs to be employed as a second layer only for cases in which an enhancement of the initial dominant response of the m^{th} node responds positively, as opposed to all remaining nodes whose responses should have decayed to zero. As shown in figure (below), MAXNET is a recurrent network involving both excitatory and inhibitory connections. The excitatory connection within the network is implemented in the form of a single positive self-feedback loop with a weighting coefficient of 1. All the remaining connections of this fully coupled feedback network are inhibitory. They are represented as M-1 cross-feedback synapses with coefficients -ε from each output. The second layer weight matrix $\mathbf{W}_M$ of size p x p is thus of the form

$$\mathbf{W}_M = \begin{bmatrix} 1 & -\varepsilon & -\varepsilon & \cdots & -\varepsilon \\ -\varepsilon & 1 & -\varepsilon & \cdots & -\varepsilon \\ \vdots & & & & \vdots \\ -\varepsilon & -\varepsilon & -\varepsilon & \cdots & 1 \end{bmatrix}$$

where ε must be bounded $0 < \varepsilon < 1/p$. The quantity ε can be called the lateral interaction coefficient. Input to the network should fulfil the initializing inputs condition

$$0 \le y_i^o \le 1, \text{ for i =1,2, ..., p}$$

The MAXNET network gradually suppresses all but the largest initial network excitation. When initialized with the input vector $\mathbf{y}_o$, the network starts processing it by adding positive self-feedback and negative cross-feedback. As a result of a number of recurrences, the only unsuppressed node will be the one with the largest initializing entry y_m^o. This means that the only nonzero output response node is the node closest to input vector argument. The recurrent processing by MAXNET leading to this response is

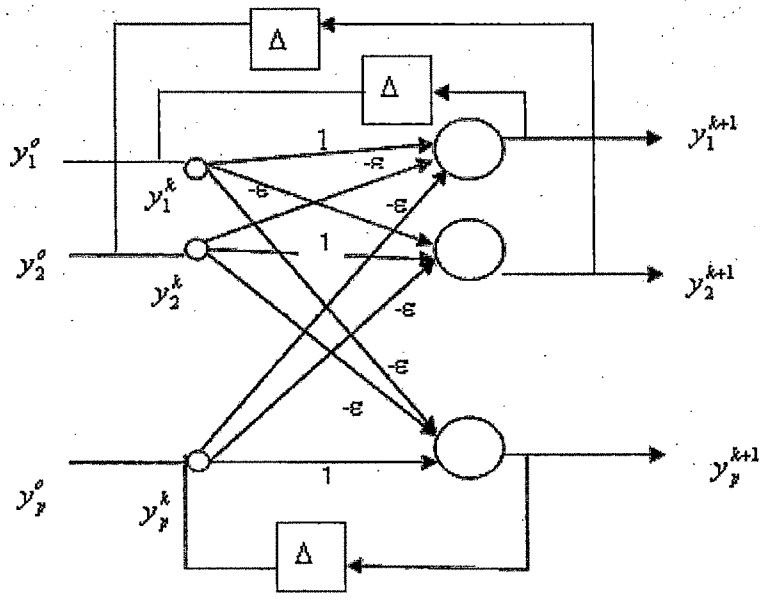
$$\mathbf{y}^{k+1} = T[\mathbf{W}_m \mathbf{y}^k] \tag{6.35}$$

where T is a nonlinear diagonal matrix operator with entries $f(.)$ given below:

$$f(net) = \begin{cases} -1 & net < 0 \\ net & net \geq 0 \end{cases} \tag{6.37}$$

Each entry of the updated vector y^{k+1} decreases at the k^{th} recursion step of equation (6.35) under the MAXNET update algorithm, with the largest entry decreasing slowest. This is due to the conditions on the entries of matrix $\mathbf{W}_M$, specifically, due to the condition $0 < \varepsilon < 1/p$.

Assume that $y_m^o > y_i^o$, i = 1,2,3...,p and i ≠ m. During the first recurrence, all entries of $\mathbf{y}^1$ are computed on the linear portion of f(net). The smallest of all $\mathbf{y}^0$ entries will first reach the level f(net) = 0, assumed at the k^{th} step. The clipping of one output entry slows down the decrease of y_m^{k+1} in all forthcoming steps. Then, the second smallest entry of $\mathbf{y}^0$ reaches f(net) = 0.The process, repeats itself until all values except for one, at the output of the m^{th} node, remain at nonzero values.



[Fig. 6.5. MAXNET architecture for p class]

Consequently, our problem is of 3-class.Therefore the synaptic weight matrix can be given by

$$\mathbf{W}_m = \begin{bmatrix} 1 & -\varepsilon & -\varepsilon \\ -\varepsilon & 1 & -\varepsilon \\ -\varepsilon & -\varepsilon & 1 \end{bmatrix}$$

where ε is the delay which is normally taken as a smaller value less than 1.

By considering $\varepsilon = 1/3$, the above matrix becomes

$$\mathbf{W}_m = \begin{bmatrix} 1 & -1/3 & -1/3 \\ -1/3 & 1 & -1/3 \\ -1/3 & -1/3 & 1 \end{bmatrix}$$

and the initial output vector obtained from the first layer, i.e. support vector machine is

$$\mathbf{y}^0 = \begin{bmatrix} y_1^0 \\ y_2^k \\ y_3^k \end{bmatrix} = \begin{bmatrix} -0.0183 \\ -0.0183 \\ 0.0183 \end{bmatrix}$$

Now using the formula (6.35), the weighted input can be given by

$$net^k = \begin{bmatrix} 1 & -1/3 & -1/3 \\ -1/3 & 1 & -1/3 \\ -1/3 & -1/3 & 1 \end{bmatrix} \begin{bmatrix} y_1^k \\ y_2^k \\ y_3^k \end{bmatrix} \tag{6.36}$$

for iteration k=0, applying the formula (6.36), we get

$$net^0 = \begin{bmatrix} 1 & -1/3 & -1/3 \\ -1/3 & 1 & -1/3 \\ -1/3 & -1/3 & 1 \end{bmatrix} \begin{bmatrix} -0.0183 \\ -0.0183 \\ 0.0183 \end{bmatrix} = \begin{bmatrix} -0.0183 \\ -0.0183 \\ 0.02562 \end{bmatrix}$$

The corresponding computed output can be given by the transfer function defined in the equation (6.37).

Using above transfer function, we get the first updated output as shown below:

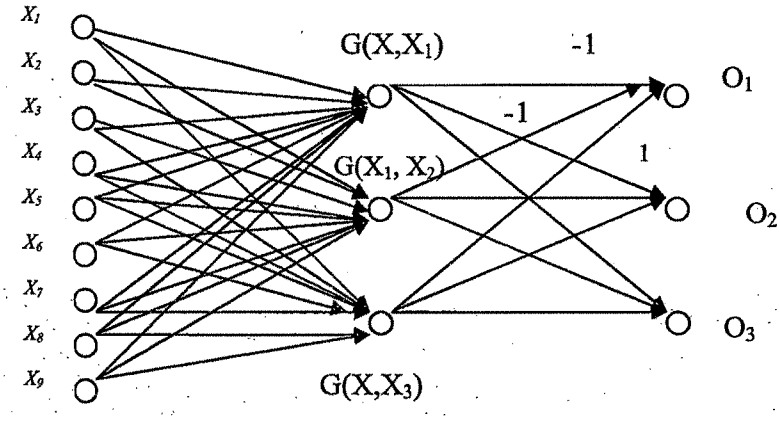
$$y^1 = \begin{bmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 0.02562 \end{bmatrix}$$

Applying the formulas (6.36) and (6.37) recursively 5 times, wet get the desired output as follows:

$$y^5 = \begin{bmatrix} y_1^5 \\ y_2^5 \\ y_3^5 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

Hence the above output response concludes that the new pattern given out here is of is of class 3 i.e. **C**.

Figure 6.6 shows the feature map of the support vector machines for multi-class problem.



[Fig. 6.6 Feature map of Support Vector Machines for 3-class problem]

Advantages for the method:

- Easy to implement
- Same technique is used to classify 2-class and multi-class problems
- For each class separate decision surface will be obtained

6.6. Summary and Discussion

This chapter deals with the various learning strategies and the experimental details for the classification of Gujarati glyphs using Radial Basis functions. The first attempt of recognition of the symbols for Gujarati numerals with two ANN architectures viz. MLP and RBF is presented in [3]. As discussed in section 3 of this chapter, the regularized RBF provides significantly good recognition accuracy of 93%. This can further be improved by selecting proper spread using learning strategies discussed earlier.

Some times Support Vector Machine can be characterized as the procedure of interpolation rather than the architecture of ANN because the feature map generated from the training set can be straight away used for the classification of the new pattern without any kind of training. Section 6.5 of this chapter introduces the new approach of classifying multiclass problems using MAXNET network. All the approaches quoted in the literatures like one-against-all, one-against-one, and DAGSVM are lacking uniformity of classifying two class problems and multiclass problems. The approach presented here is capable of handling multiclass problems as well as two class problems. We have solved the dual problem mentioned in the equation (6.30) for each category. Each solution of the dual problem yields the unique optimal hyperplanes for each category which separates the current class with the remaining classes and thus adopts the one against all approach. This new approach provides same number of optimal hyperplanes as the number of classes. Therefore this approach remains uniform for two class as well as multiclass problems.

These kernel based ANN architectures are based on sound mathematical background and can be used for complex classification problems like speech recognition, character recognition etc.