Acknowledgements

Abstract

Abstract

Chapter 1

Introduction

A major goal of Artificial Intelligence research (AI) continues to be that of inventing systems that are intelligent and capable of dealing with the real world, in a manner similar to human beings [1]. The human brain's decision-making capabilities rely heavily on its pattern recognition abilities in order to make rational decisions. Hence, it is imperative to design systems that are capable of recognizing patterns efficiently. The study of pattern recognition is broad with a wide range of applications including, but not limited to, fingerprint recognition, character recognition, DNA sequence identification and much more. Therefore, the design of such a recognition system requires careful attention to a variety of issues like feature selection, extraction and segmentation, pattern classes, class representation, classifier design, learning techniques used, and design of the training and testing data[2] [9].

The task of learning and recognizing patterns is a challenging process in itself due to the variety of real world problems [10] to which it is relevant. This task becomes particularly difficult when the patterns to be recognized or classified are large in number and belong to

high dimensional spaces that cannot be visualized by humans. Often data of interest are not linearly separable/discernible in their original feature space. The present day advances in technology in the fields of image processing, clustering, computer vision and machine learning has led to a number of techniques to classify this kind of data. Researchers have come up with powerful techniques including neural networks, fuzzy logic, support vector machine and genetic algorithms to process large (or very large) and complex data sets. Many of these techniques typically work by mapping the original data into sufficiently higher dimensions in order to obtain a separating hyperplane.

Support vector machines(SVMs) help in finding hyper planes, which are optimal i.e., separate the patterns in the best possible manner (with the maximum generalization power), as opposed to other techniques like perceptrons where the maximum generalization is not assured. In addition to this, the VC dimension (Vapnik Chervonenkis) [8] of an SVM based system can be calculated and this will give the likelihood of the SVM system performing well on unknown data types. This is not possible in other learning systems such as neural networks. Traditionally, SVM's have been based on kernels. Kernels are functions that perform an implicit non-linear mapping of the data to higher dimensional spaces, in which data are linearly separable, preserving the inner product. The main difficulty of using kernels is that there is no defined rule for constructing a kernel for a given problem and therefore kernel selection is often done on a trial and error basis. Sometimes this also causes overfitting[10].

The research presented in this thesis is an extension of the algorithm proposed in [3], where experiments were done to classify simple linear and non-linear data sets without using

kernels. This work focuses on overcoming the limitations of [3], some of them being, the use of a threshold on the (Euclidean) distance to collect points closer to the class boundary and the inability to apply the technique to closed class boundary data sets. This thesis is organized as follows: The first chapter gives an overview of the entire thesis. The background of this research is explained in the next chapter. The third chapter describes kernels and discusses this work as an alternative to using kernels for the kind of classification problems this thesis deals with. Chapter 4 describes the methods used and the limitations encountered in [3] from which this thesis has evolved. The fifth chapter describes in detail the strategies adopted in this work to overcome the limitations mentioned in Chapter 4. This chapter also explains the different approaches experimented and their results. An algorithm is proposed that meets the goal of this thesis, namely, the classification of data sets with convex polygonal class boundaries and a more efficient method to find the class boundary edge points than that used in [3]. Finally, Chapter 6 gives a summary of this research along with some ideas for the direction of future research.

Chapter 2

Background

2.1 Pattern Recognition

Today, with more and more applications and industries turning towards automation, there arises a need to develop systems that can take over various tasks humans currently perform. As a result, pattern recognition has become a highly researched area. Most of the pattern recognition tasks are represented as classification or categorization tasks and the designed system is made to learn those conditions that would lead the inputs to be categorized into a particular class[2]. This task is split into two stages: a training stage where the learning system is programmed to *learn* how to classify patterns and a testing stage, where the system applies what it has learned on some test patterns and classifies them.

2.2 Supervised and Unsupervised Learning

Supervised learning is the process where a system can be made to learn to recognize patterns by explicitly providing the correct output for every pattern.

Unsupervised learning is the process of learning where a system has no hint about its output but the system learns from the previous inputs and adjusts itself to produce correct results.

The various states are defined as representations of functions and learning is the task of actually recomputing these representations. For example, in supervised learning, a function can be represented, as being in state 0 while its desired state may be state 1. The parameters and their changes that bring about this transition from its present state to the goal state is the information the system has to learn to classify inputs correctly in future [4].

2.3 Approaches for Pattern Recognition

There are several ways a system can be made to recognize the patterns that need to be classified. The approach used in the design of a classifier depends on the domain the classifier is going to be used in, and the type of data sets it has to classify. The four popular approaches according to [2] are a) template matching, b) statistical approach c) syntactic approach and d) neural networks.

In the template matching approach, the pattern to be recognized is matched against a template that is learned from training data. The similarity between the template and the target pattern is calculated and it is used to classify the pattern. The statistical approach

attempts pattern recognition by trying to find the decision boundaries in an n dimensional feature space between the data of the different classes. The important aspect of the statistical approach is the idea of learning my minimization of empirical risk function (a measure of the classification error on a sample of data - the training data set). The syntactical approach considers patterns to be built from sub patterns and hence complex patterns are recognized from having just a basic set of patterns. A neural network system can be represented as a weighted directed graph where each node is called a neuron. Each node is associated with an activation function and the weights associated with the nodes are adjusted during the process of learning and thus adapt themselves to the data to be classified.

2.4 Stages in Classification, Pattern Recognition

A pattern recognition method usually includes training and a testing stage. The training stage is commonly referred to as the learning stage and the testing stage is the classification stage. During the learning phase, the classifier partitions the feature space based on the training samples and therefore good training samples help design a good classifier. Training samples are labeled in the case of supervised learning and unlabeled in unsupervised learning. Some unsupervised classifiers maybe designed with a feedback factor, which helps the system adapt to the training samples.

Overfitting may occur when the classifier is trained too well for the training set or when the training set does not properly represent the kind of data it has to classify. A classifier can be expected to give best performance if it is trained to have maximum generalization

so that the disadvantages of over fitting can be avoided. The Support Vector Machine [7], the subject of this thesis is one such classifier, which learns with maximum generalization ability. Hence this technique is a very powerful classification technique with applications in several areas of science and technology.

2.5 Support Vector Machines

Many existing approaches to classifier design (perceptrons, Bayesian framework, etc.) suffer from the disadvantage of over fitting the training data with poor generalization capability. The ability to provide maximum generalization is an important characteristic of any good classifier and Support Vector Machines exactly do this. SVMs are also appealing because of their mathematical foundations. The separating surfaces generated by SVMs are optimal hyperplanes where the optimality is defined as the maximum distance between the hyper plane and the nearest patterns of the training sets [5]. The maximum distance is called the *margin* and the training patterns for which the margin is attained are called *Support Vectors*.

2.6 Mathematics of Support Vector Machines

The simplest model of a Support Vector Machine is the maximal margin classifier. This classifier works only for linearly separable datasets and separates classes by finding the hyperplane with the maximum margin. This can be expressed as a convex optimization problem. The objective function is to maximize the margin subject to linear inequality

constraints. The following section explains how this strategy is formulated as a quadratic programming problem.

2.7 Hard Margin SVM

2.7.1 Primal Form

Given a set of linearly separable training examples $\{x_i, y_i\}$, $(i = 1, \ldots, M)$ where x_i corresponds to an n-dimensional input and the class labels are given by y_i, $y_i \in \{1, -1\}$, the separating surface for these two classes is given by

$$D(x) = w^t x + b \tag{2.1}$$

where w is an n-dimensional vector that parameterizes all the hyperplanes and b is a constant called the bias. Let us consider the hyperplanes to have a margin of 1 from the support vectors. Additionally, all the points in the training set must satisfy the constraints

$$w^t x_i + b \geq +1, \quad for \;\; y_i = +1 \tag{2.2}$$

$$w^t x_i + b \leq -1, \quad for \;\; y_i = -1 \tag{2.3}$$

The above equations (2.2)and (2.3) can be combined together as

$$y_i(w^t x_i + b) \geq 1, \quad i = 1, \ldots, M \tag{2.4}$$

The data points satisfying the equations (2.2) or (2.3) with equality are the support vectors and they lie on the hyperplanes given by $H_1 : wx_i + b = 1$ and $H_2 : wx_i + b = -1$, respectively. The margin can be determined by the Euclidean distance d_1 from the hyperplane H_1 to the optimal hyperplane H, plus the Euclidean distance d_2 from the hyperplane H_2 to H. Since $d_1 = d_2 = 1/||w||$, the margin given by $d_1 + d_2$ is $2/||w||$. Finally, the optimal separating hyperplane with the maximum margin is obtained by solving

$$Minimize: \ \frac{1}{2}||w||^2 \tag{2.5}$$

with respect to w and b, subject to the constraints (2.4).

The convex optimization problem given by (2.5) and (2.4) is known as the primal form.

2.7.2 Dual Form

In the dual form, the constrained problem represented by (2.5) and (2.4) is transformed to an unconstrained one by introducing non-negative Lagrangian multipliers α_i $(i = 1, \ldots, M)$ for the inequalities in (2.4), to obtain the objective function in (2.6).

$$Q(w, b, \alpha) = \frac{1}{2}w^t w - \sum_{i=1}^{M} \alpha_i \{y_i(w^t x_i + b) - 1\} \tag{2.6}$$

By minimizing $Q(w, b, \alpha)$ with respect to w, b, and maximizing with respect to $\alpha_i (\alpha_i \geq 0)$ we get the optimal solution of (2.6). Therefore, the optimal solution (w, b, α) must satisfy

the following conditions:

$$\frac{\delta Q(w,b,\alpha)}{\delta b} = 0 \tag{2.7}$$

$$\frac{\delta Q(w,b,\alpha)}{\delta w} = 0 \tag{2.8}$$

Using(2.6), (2.7) becomes

$$\sum_{i=1}^{M} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad for \;\; i = 1, \ldots, M \tag{2.9}$$

and (2.8) becomes

$$w = \sum_{i=1}^{M} \alpha_i y_i x_i, \quad \alpha_i \geq 0, \quad for \;\; i = 1, \ldots, M \tag{2.10}$$

(2.9) and (2.10) is substituted in (2.6), to get the dual form as shown in (2.11)

$$Maximize: \;\; Q(\alpha) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j x_i^t x_j \tag{2.11}$$

with respect to α_i subject to the constraints (2.12) and (2.13).

$$\sum_{i=1}^{M} \alpha_i y_i = 0 \tag{2.12}$$

$$\alpha_i \geq 0, \quad for \;\; i = 1, \ldots, M \tag{2.13}$$

The solution to the optimization problem in (2.11) must also verify the Karush-Kuhn-

Tucker (KKT) conditions [10]:

$$\alpha_i\{y_i(w^t x_i + b) - 1\} = 0 \tag{2.14}$$

Solving for α_i in (2.11) - (2.14), we obtain the support vectors. The separating hyperplane is given by

$$D(x) = \sum_{i=1}^{M} \alpha_i y_i x_i^t x + b \tag{2.15}$$

where

$$b = y_i - w^t x_i, \quad for \;\; all \;\; \alpha_i \neq 0 \tag{2.16}$$

A data point x is classified based on the sign of the hyperplane i.e., into $class_1$ if $D(x) > 0$ and into $class_2$ if $D(x) < 0$

2.8 Soft Margin SVM

The SVM problem stated in Section 2.7 is usually referred to the Hard Margin SVM. Hard Margin SVM does not deal with noisy data. Soft Margin SVM is obtained by introducing non-negative slack variables in order to be more flexible and tolerate outliers and noisy data. The soft margin SVM attempts to minimize classification error while trying to maximize the margin.

2.8.1 Primal Form

Introducing the nonnegative slack variables $\xi_i(>0), i = 1, \ldots, M$ in (2.4),

$$y_i(w^t x_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, M \tag{2.17}$$

The primal form of soft margin SVM is obtained by solving

$$Minimize: \ \frac{1}{2}||w||^2 + C\sum_{i=1}^{M}\xi_i \tag{2.18}$$

subject to the constraints (2.17) and

$$\xi_i \geq 0, \quad for \ \ i = 1, \ldots, M \tag{2.19}$$

$C > 0$ is the parameter that controls the tradeoff between the minimization of classification error and maximization of margin. If C is large, the number of misclassification would be small and while C is small, the wider the margin would be.

2.8.2 Dual Form

Introducing Lagrange multipliers in the soft margin Primal Form we get the dual formulation. Two different multipliers, α and β are used here for the constraints (2.17) and (2.19).

Therefore we have

$$Q(w, b, \xi, \alpha, \beta) = \frac{1}{2} w^t w + C \sum_{i=1}^{M} \xi_i - \sum_{i=1}^{M} \alpha_i \{y_i (w^t x_i + b) - 1 + \xi_i\} - \sum_{i=1}^{M} \beta_i \xi_i \tag{2.20}$$

The dual form can be obtained by differentiating w.r.t $(w, b, \xi, \alpha, \beta)$

$$\frac{\delta Q(w, b, \xi, \alpha, \beta)}{\delta b} = 0 \tag{2.21}$$

$$\frac{\delta Q(w, b, \xi, \alpha, \beta)}{\delta w} = 0 \tag{2.22}$$

$$\frac{\delta Q(w, b, \xi, \alpha, \beta)}{\delta \xi} = 0 \tag{2.23}$$

Substituting in (2.20), yields,

$$\sum_{i=1}^{M} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad for \;\; i = 1, \ldots, M \tag{2.24}$$

$$w = \sum_{i=1}^{M} \alpha_i y_i x_i, \quad \alpha_i \geq 0, \quad for \;\; i = 1, \ldots, M \tag{2.25}$$

$$\alpha_i + \beta_i = C, \quad \alpha_i, \beta_i \geq 0, \quad for \;\; i = 1, \ldots, M \tag{2.26}$$

The objective function (2.20) is maximized and the dual formulation is obtained:

$$Maximize: \;\; Q(\alpha) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j x_i^t x_j \tag{2.27}$$

subject to

$$\sum_{i=1}^{M} \alpha_i y_i = 0 \tag{2.28}$$

$$0 \leq \alpha_i \leq C, \quad for \;\; i = 1, \ldots, M \tag{2.29}$$

Further the optimal solution must also satisfy the corresponding KKT conditions:

$$\alpha_i \{ y_i (w^t x_i + b) - 1 + \xi_i \} = 0 \tag{2.30}$$

$$\beta_i \xi_i = (C - \alpha_i) \xi_i = 0 \tag{2.31}$$

The separating hyperplane is again given by (2.15).

Chapter 3

Using Kernels In SVMs

3.1 Kernels

For non-linear data sets, Support Vector Machines make use of kernel functions to perform a mapping to a higher dimensional space where the data set is linearly separable. The kernel function $K(x_i, x)$, these mappings have to be made such that the relation $K(x_i, x_j) = \phi(x_i)^t \phi(x_i) = f(x_i^t x_j)$ holds. For a linear classifier, a kernel is the dot product between an input pattern x and a member of the support set i.e., $K(x_i, x_j) = x_i.x_j$. In other words, the kernel converts the convex non-linear computations to a simple dot product solution in a linear space [6].

There are several common kernel functions including polynomial kernels, sigmoid kernels or Gaussian radial basis kernels. Further appropriate kernels can be engineered for the different data sets such they satisfy the Mercer's theorem [6].

Some of the problems using the kernel based approach is that of selecting the best kernel

function for a given problem or, equivalently, choosing the right mapping to represent the non-linear problem in a linear data space and the properties this function should have to provide the best approximation of the non-linear function. No definite solution has been discovered for this problem and most of the times Kernel functions have to be chosen on a trial and error basis and in some cases lead to overfitting.

Solving the Support Vector Problem is equivalent to solving a quadratic programming problem, the objective function being the maximization of the margins with the KKT conditions as constraints. The size of this quadratic problem depends directly on the size of the training data because the number of variables for the problem is equal to the number of training data. If the training data set is large then solving this as a quadratic problem is very complex. The common solution is to split the training data using some good algorithm and consider each of these subsets as a quadratic problem [3]. This thesis also proposes an algorithm along the lines of this method. This is to avoid the common problems that occur with the use of kernels. This work is an extension of [3] and investigates the possibility of considering the whole non-linear problem as the union of smaller problems that can be solved using linear SVMs. In a way this is almost similar to the fact that a non-linear surface can be represented as a combination of a number of linear surfaces. This algorithm attempts to find these small linear surfaces and thus proves to be an effective substitute for the kernel method.

Solution of the Small Quadratic Programming Problem

The solution to a small quadratic problem as mentioned above can be obtained using the Basic Algorithm of [3]. This algorithm from [3] corresponding to the hard margin SVM is given in Table 3.1.

Table 3.1: The Basic Algorithm [3].

BA1.	Solve the KKT conditions in (2.12) and (2.14) for $\alpha_i, i = 1, \ldots M$ and b.
BA2.	From the solutions obtained in BA1, eliminate those solutions that do not satisfy the constraint given by (2.13).
BA3.	From the remaining solutions, select that that maximize the objective function, $Q(\alpha)$.

Chapter 4

Piecewise Linear Approximation

The Piecewise Approximation method proposed in [3] attempts to find the final boundary of linearly and non-linearly separable data sets without using kernels. Therefore to achieve this the Basic Algorithm discussed in Chapter 3 has to be iteratively applied to small subsets of the training set. These subsets can be chosen based on a number of conditions. The research presented in [3] exploits one such method of using a threshold on the Euclidean distance between the positive and negative class to achieve this. This chapter discusses the limitations of using this approach and investigates several other constraints and their results.

4.1 Iterative SVM Algorithm

The Iterative SVM(ISVM) Algorithm proposed in [3] chooses small subsets of points from the training data to which the Basic Algorithm described in Chapter 3 is applied. When such a subset is linearly separable, the corresponding separating hyperplane is determined.

When the subsets chosen are not linearly separable (a fact that can be tested immediately), the points are returned to the training set. The SVM theory guarantees that the hyperplanes generated in this way are locally optimal. These hyperplanes are then combined together to obtain the overall separating surface. This algorithm uses a threshold on the Euclidean Distance to choose subsets of points thus making sure that the selected points lie as close to the boundary as possible. This process of choosing subsets of points satisfying the threshold condition set by the user and finding the separating hyperplane for them goes on until all the training points have been exhausted and no more subsets can be formed. The final separating surface is then inferred as described below.

Inferring the Final Separating Surface

The Final Separating surface for the above Iterative SVM uses a simple logic of starting with the hyperplane having the smallest slope and evaluating it against all the data points, switching to the next best hyperplane when an error is encountered in classification. The algorithm considers the hyperplane being evaluated at any stage against a set of points as the *Current* hyperplane. As this set of points keep changing, errors are encountered and a switch to a better hyperplane takes place and the new hyperplane becomes the *Current* hyperplane.

Though the iterative SVM algorithm gives fairly good results, improvements can be made in the constraints imposed for choosing subsets of points as inputs for the Basic Algorithm as well as in inferring the final surface. This thesis deals with these improvements and

also presents a discussion on the correctness of the method of inferring the final hyperplane used in [3] and the extent to which it preserves the advantages of using Support Vector Machines for such problems. Finally, an alternative to this method is proposed and used in this research and the results are presented.

4.2 Limitations of Iterative SVM Algorithm

4.2.1 Using a Distance Threshold

The Iterative SVM algorithm Method has several limitations, the foremost, being the use of distance as a constraint in choosing the subsets of points for applying the Basic Algorithm. This constraint is very important because it decides the points chosen as input for each run of the Basic Algorithm. The closer these points are to the actual boundary, fewer number of hyperplanes need to be generated, hence the computation involved in inferring the final surface is also reduced and better accuracy can be achieved.

The distance threshold used in Iterative SVM algorithm is supplied by the user as input. Choosing points such that the maximum distance between positive and negative class points is less than the threshold does not guarantee that the points are actually closest to the boundary. Since the user has no idea about the data points, the threshold distance chosen is random and if it happens to be large, then several subsets of points satisfy this condition resulting in many hyperplanes, some of which might not at all be close to the boundary. Also since the maximum Euclidean distance between the points of either class in a subset

need not necessarily be the perpendicular distance, it is possible that the points for training are sometimes not the class boundary points. Figure 4.1 shows the maximum distance between data points that belong to two different classes obtained using Euclidean Distance. The Iterative SVM algorithm selects those subsets of points which have maximum distance between the two classes smaller than the threshold value and discards the others. For example, as shown in Figure 4.1, even if the required distance is smaller than the threshold value, the subset is discarded by the algorithm if the maximum distance as calculated by the Euclidean measure is greater than the threshold.

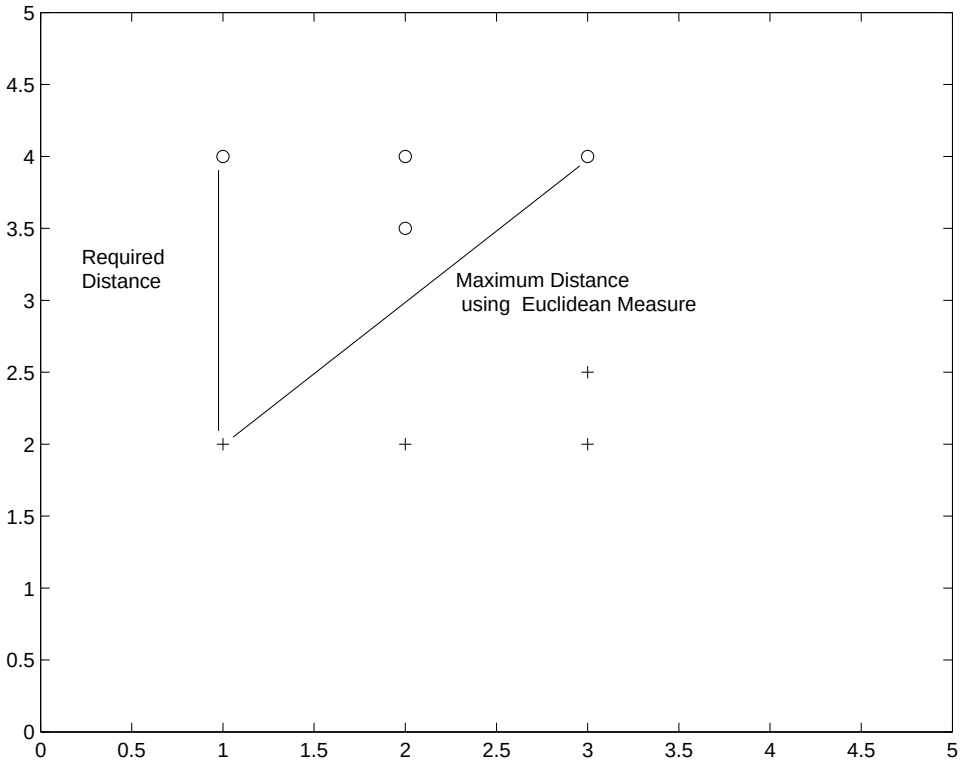

Figure 4.1: Maximum Distances between the two classes.

4.2.2 Inference of the Final Separating Surface

Below are the steps, which detail how the Iterative SVM algorithm infers the final, separating surface.

1. Hyper planes are found by the Basic Algorithm and drawn for a range corresponding to the maximum and minimum of the entire training set.

2. Hyper planes are then arranged in non-decreasing order of their slopes.

3. Each hyperplane is then evaluated against the training data points to see if it classifies the selected group of points correctly. Hyperplanes are switched when a classification error occurs (or, after allowing a small error for each hyperplane).

The Basic Algorithm finds the hyperplanes for a set of points with maximum generalization. However, there is a possibility that the method of inferring the final separating surface as done in [3] - finding the hyperplanes between the maximum and minimum limits of the entire training data set and then evaluating them to infer the final surface - might result in some hyperplane which may actually not be a hyperplane generated for that region, but just an extension of the numerous hyperplanes generated for the different training sets evaluated on that region. Let us examine this scenario for a region $R1$ where the hyperplane generated by the Basic Algorithm is $H1$. In the same way, consider $H2$ to be another hyper plane generated for region $R2$. Let us assume that $H2$ extends into the region $R1$ and here it is called $H21$.

When the separating surface is very non-linear, it is highly doubtful, probably very unlikely, that such a hyper plane as $H21$ might have fewer errors than the actual hyperplane $H1$ for the region $R1$. Hence the algorithm might eventually drop such hyper planes. But if the separating surface is (almost) linear, it is possible that $H21$ and $H1$ will have approximately the same number of errors; the difference being the generalization errors. So there is a possibility that either might be selected. If this happens consistently, then the true purpose of using SVMs (highest generalization power) might not be exploited to the maximum. A different approach has been employed in this research to overcome this problem.

4.2.3 Complex Data Sets

Complex data sets, like closed shapes have difficult boundaries and the Iterative SVM algorithm does not take into account such data sets. Finding the boundaries for such data sets is definitely a challenge and has been dealt with extensively in this research.

Applying ISVM Algorithm on Complex Datasets

As discussed before, the algorithm in [3] chooses portions of hyperplanes that are optimal with respect to the points they are evaluated against and they are combined together by vertical lines to obtain the final separating surface. For complex data sets, the algorithm finds a series of hyperplanes for the upper and lower boundaries and combines them using vertical lines. Further, though the Basic Algorithm generates vertical hyperplanes, they are not selected while the final surface is inferred because, the data points in the training set are

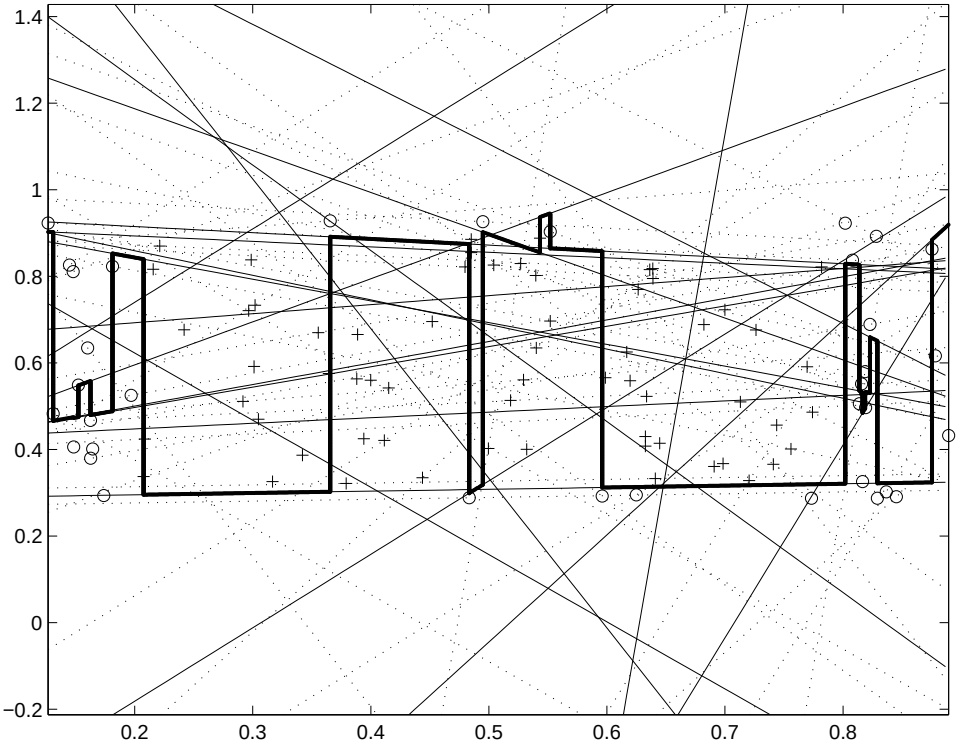

Figure 4.2: Using Iterative SVM on Two Class Rectangle Dataset.

evaluated only along the length of the x-axis.

Figure 4.2 shows the output generated using the Iterative SVM algorithm for a complex data set where the boundary between the two classes is in the form of a rectangle. It is obvious that a different approach needs to be considered for data sets with difficult class boundaries. This thesis mainly deals with convex polygonal data sets and their class boundaries, while also making use of more efficient methods of finding class boundary edge points and inference of the final surface.

Chapter 5

Iterative SVM for Complex Datasets

Chapter 4 discusses the performance of the Iterative SVM algorithm on closed shapes data sets arriving at the conclusion that this technique needs to be modified in order to be able to apply it to complex data sets. This chapter explains different experiments performed towards the modification of the ISVM for more complex datasets (convex closed boundaries).

5.1 Adapting Iterative SVM technique for complex data sets

In order to use the Iterative SVM algorithm in [3], a simple technique based on axis swapping is adopted here. More precisely, in this algorithm, the entire training data set is split into four subsets based on the medians, Δ_x and Δ_y of x and y axes respectively. The training points above and below the line $y = \Delta_y$ form the *upper* and *lower* sets respectively. The Iterative SVM Algorithm is applied to the *upper* and *lower* sets (eventually with a small

Table 5.1: Swapping of Axes Algorithm.

Input:	The training set T
Output:	The final separating surface inferred for the training set T obtained using the ISVM algorithm
SAA1.	Find the median (Δ_x, Δ_y) along the x and y axes for the training set.
SAA2.	Split the training set into *upper*, below as *lower*: $upper = \{(x, y) \in T, y \geq \Delta_y\}$; $lower = \{(x, y) \in T, y < \Delta_y\}$
SAA3.	Apply the Iterative SVM algorithm to the *upper* and *lower* sets. Collect the final separating surface for these subsets.
SAA4.	Interchange the x and y axes. Repeat Steps SAA2 and SAA3.
SAA5.	Restore axes.

allowance for error, e.g., 10% error of the current chunk of the training set. Next the x and y axes are interchanged and the *left* and *right* sets, determined by $x = \Delta_x$ replace now the *upper* and *lower* sets for which the Iterative SVM algorithm is applied again.

Finally, the axis swapping is necessary because of the way the Iterative SVM algorithm was initially designed: all the processing was restricted along the x-axis alone.

The axes are swapped back to their originals after the above steps have been performed.

The Swapping of Axes algorithm gives fairly good results for classes with boundaries parallel to the axes of coordinates (rectangles, squares) as shown in Figure 5.1. The results were not as good for other polygonal shapes such as triangles, rotated rectangles, etc. For these complex polygon data sets, even after splitting, the resulting subsets were still too complex for the Iterative SVM method as shown in the Figure 5.2. Often, the separating

surfaces found overlap with each other and are not correct.

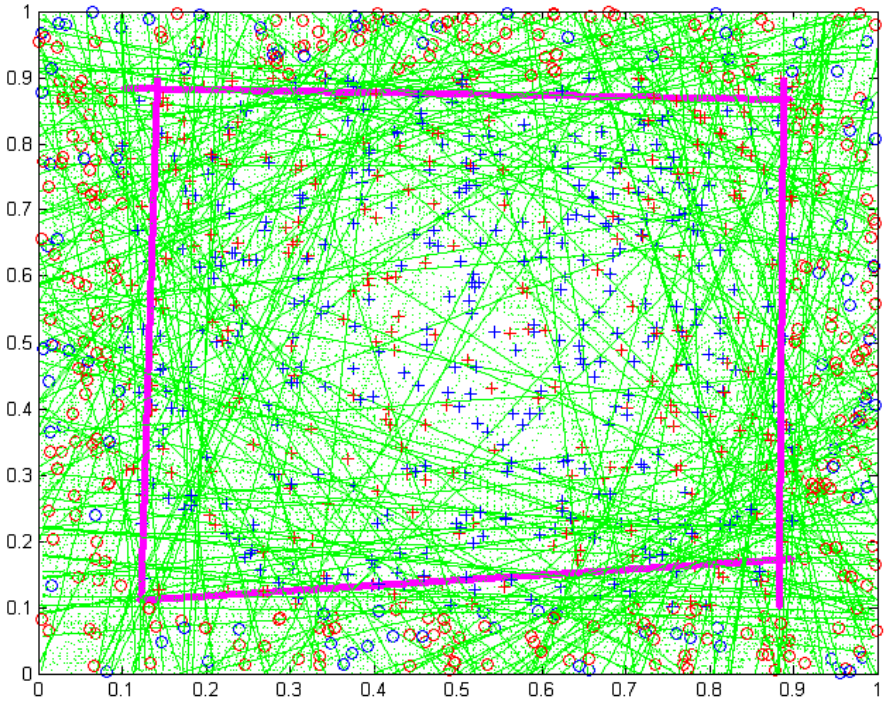

Figure 5.1: Typical result for the Axis Swapping Algorithm for a rectangle data set with sides parallel to the axes of coordinates.

5.2 Iterative Partitioning using Median

Here the idea of partitioning by the median is carried over to subsequent stages of the algorithm. At each stage, the current subsets are split by their respective medians. Eventually, the subsets generated in this fashion are small enough to apply the Basic Algorithm.

The overall separating surface is inferred as a combination of the linear surfaces obtained for these subsets.

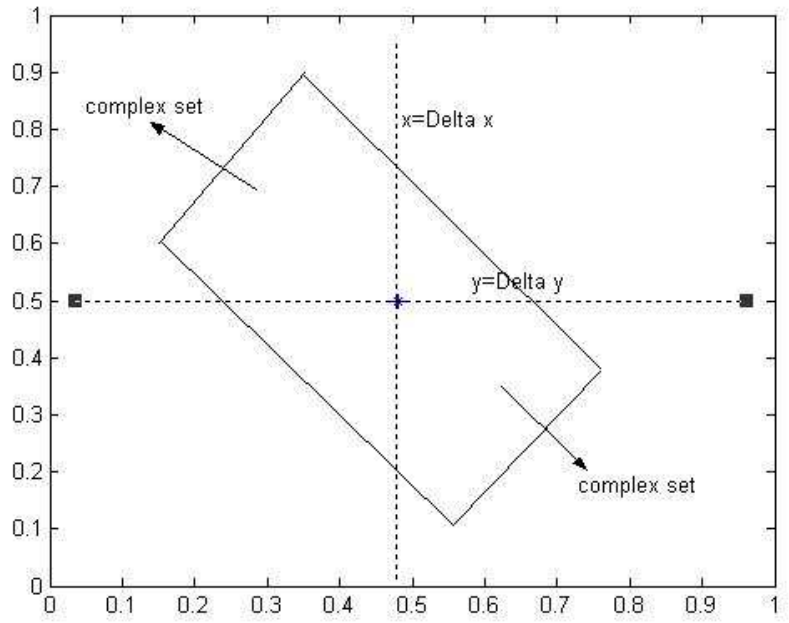

Figure 5.2: Polygonal Shapes too complex for the Swapping Of Axes algorithm.

How does the Iterative Median Partitioning work?

The efficiency of the Iterative Median Partitioning method depends largely on the particular data set to which it is applied. If the training and testing data sets do not have the same underlying data distribution (uniform), then when the subsets are formed, some of the sparse regions near the boundary are grouped into potentially the wrong class.

Since the Basic Algorithm detects subsets having same class points and eliminates them from further training (because such groups are supposed to indicate that they are far away from the class boundary), in some cases (non-uniform distributions), this will lead to holes

Table 5.2: The Iterative Median Partitioning Algorithm.

Input:	The training set T; n a threshold on the training set size.
Output:	The final separating surface inferred for the training set T
IMPA1.	While the the size of the current training set is greater than n.
	(a) Find the median (Δ_x, Δ_y) along the x and y axes for the training set. **(b)** Split the training set into *upper*, *lower*, *left*,*right*: $upper = \{(x,y) \in T, y \geq \Delta_y\}$; $lower = \{(x,y) \in T, y < \Delta_y\}$ $left = \{(x,y) \in T, x \leq \Delta_x\}$; $right = \{(x,y) \in T, x > \Delta_x\}$ **(c)** Set the training set to each of these subsets.
IMPA2.	Apply the Basic SVM algorithm to all the current training subsets: **(a)** For each subset containing points from different classes the corresponding hyperplane is generated for the extent of the subset only. **(b)** Each subset containing points from one class only is discarded.
IMPA3.	The final surface is the combination of all the surfaces obtained.

in the training set that are propagated into the final separating surface.

5.3 Edge Points

In the following, training points which lie close (in a sense to be made precise) to the class boundary are informally referred to as *edge points*. Many of these points may end up to be the support vectors locally for the separating surface. However, they need not be so.

The Iterative Median Partitioning algorithm, as well as practically all other implementations of the Support Vector Machine approach, illustrate the important role of the edge points: they define the separating surface in a data set. Therefore, a common approach to

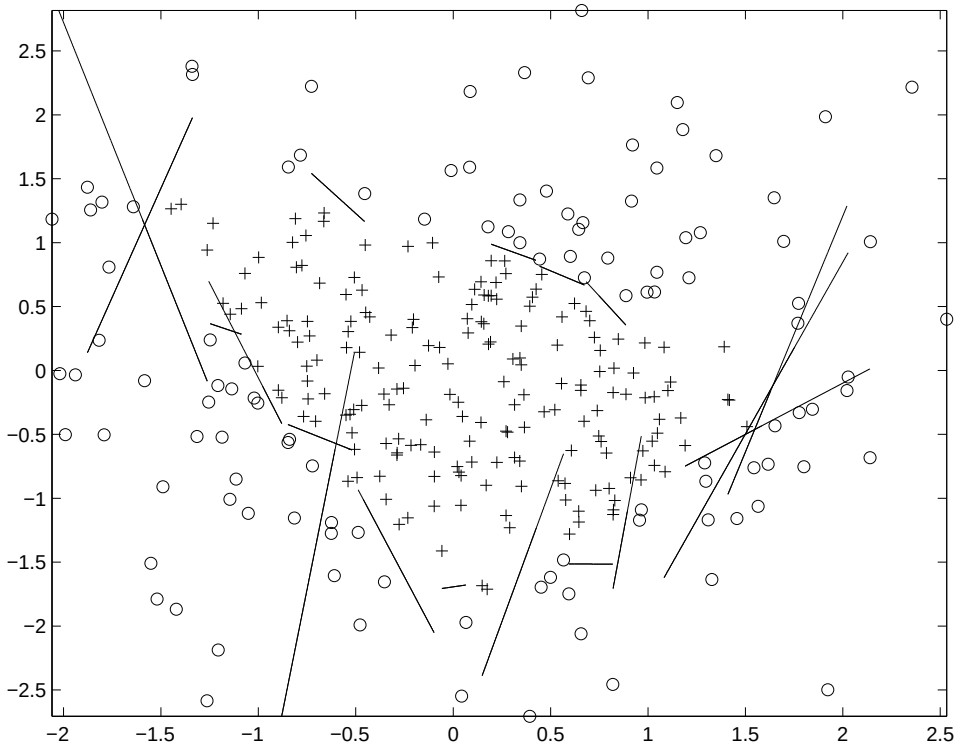

Figure 5.3: Hyperplanes obtained from the Median Partitioning Algorithm.

implementations of the SVM paradigm, is to first find some information on the edge points and then to use only these points to find the actual separating surface between classes. Indeed, the approach in [3] using a threshold on the distance between classes, is one way in which edge points were determined. It is however obvious that the actual value of the threshold is very important and setting a good threshold value even for the same training set (that is, one which identifies edge points in different regions of the training set, when the data is not uniformly distributed) may be difficult.

In any case, once a *good* set of edge points is determined, the hyperplanes generated based on them will be more effective in separating the two classes. Good edge points also

greatly reduce the complexity of training.

5.4 Finding Good Edge Points

Various approaches to find good edge points were experimented with as part of the work for this thesis.

5.4.1 Using the K-Nearest Neighbors (K-NN)

The ISVM algorithm was used first. For each of the support vectors generated a small subset of its $K = 10$ neighbors was generated. The Basic Algorithm was then applied to subsets containing points from the two classes.

As already mentioned in Chapter 4, since all the support vectors obtained by the Iterative SVM are not necessarily near the boundary, some method was needed to choose from these support vectors, the class boundary edge points. Therefore, for each of the Support Vector obtained, its 10 closest neighbors were collected using K-NN approach [11], and subsets were formed. The assumption is that if such subsets have different class points then they must lie close to the class boundary and so the separating surface can be determined using the Basic Algorithm.

Experiments carried out with this approach showed that it too was dependent on the distribution of the points in the training data. Though most of the edge points near the class boundary were determined with some accuracy, they could not be exploited well: If the distance between the two classes was slightly higher in any region, then the closest

points chosen by K-NN belong to the same class, and so the Basic Algorithm never gets called leading to gaps in the final surface. If there are many large gaps, combining the few hyperplanes obtained does not give a final surface with good generalization.

5.4.2 Using other distance measures

Collecting edge points using a threshold on the Euclidean distance between points in different classes does not always capture the topology of the class boundary. Other distance measures can be used. For example, experiments were done with the $Dist_{min}$ distance defined in (5.1).

$$Dist_{min}((x1, y1), (x2, y2)) = min(|x1 - x2|, |y1 - y2|) \tag{5.1}$$

The idea behind using $Dist_{min}$ is to capture edge points which are close along one of the components, even if they are far away along the other.

However, in general, except for very particular cases (for example, some cases where class boundaries are parallel to the axes of coordinates), the results were very poor.

The reason for this failure is that sometimes points very far from the class boundary are selected because they share a closeness on one of their coordinate axis. For such points the separating surface obtained from the Basic Algorithm is totally away from the true separating surface. Further, the problem of selecting a good threshold value remains.

5.4.3 Context dependent selection of edge points: nearest two points

The next experiment evolved in an effort to remove the problems faced in the various methods worked with so far. The most important feature of this experiment is that it eliminates the need for using a threshold parameter. The idea is to determine closest points in a way which depends on the data set, and for the same data set, on the particular region in which the points lie (this cannot be accomplished by using one threshold). Informally this closeness property can be viewed as context dependence.

In this method, every point in the training set is projected into the nearest point in the training set belonging to the other class. The procedure is repeated until it converges.

More precisely, point A belonging to $Class_1$ is projected into its closest neighbor, point B in $Class_2$. Subsequently, B is projected into its closest neighbor A' in $Class_1$. This process is repeated until $A \equiv A'$.

The separating hyperplane is the perpendicular bisector between the two points. The hyperplanes generated this way also act as support vector hyperplanes since the margin from the points on either side is maximized.

However, the perpendicular bisectors (based on two training points only) do not always provide the correct separating surface for a group of points (even when these include the two training points). Figure 5.4.3 shows the output obtained for a two-class data set where the separating surface is a rotated rectangle.

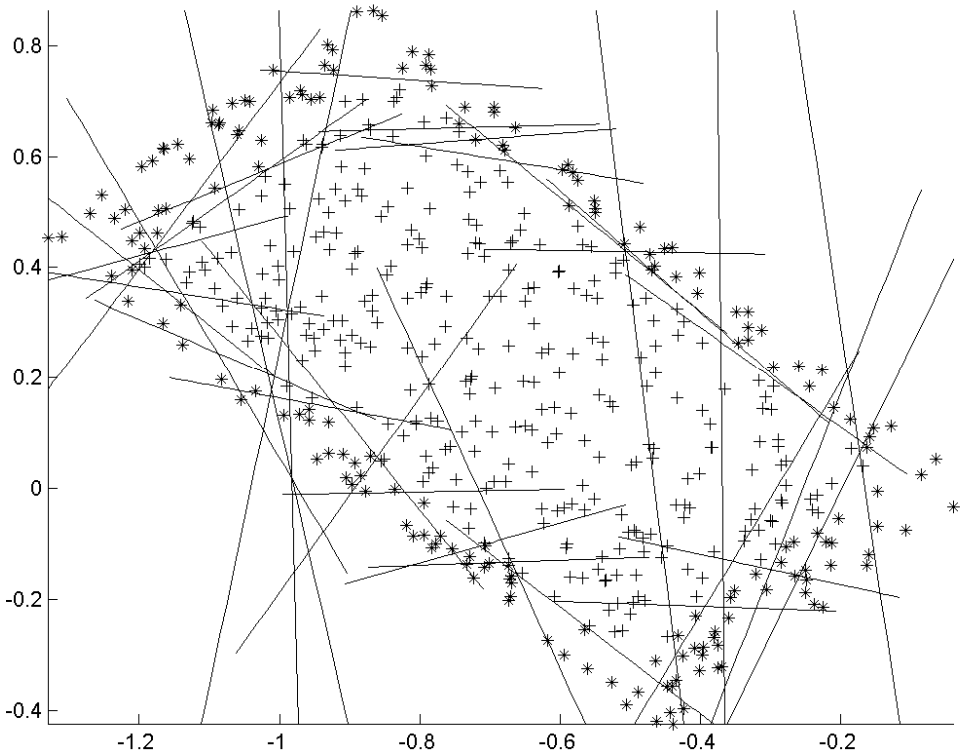

Figure 5.4: Perpendicular bisectors of the closest boundary edge points.

5.4.4 Context dependent selection of edge points: nearest three points

Finally, the edge points are found as the subsets of nearest three points from the two classes.

The triangles formed in this way are called the Delaunay triangles[12]. The vertices of a Delaunay triangle are now the closest points to each other (there are no other points from the training set in the interior of the circle circumscribing the triangle).

If these points belong to different classes then we can conclude that these are the edge points for the two different classes in the dataset. For the classification problem, if these points are all of the same class, we know that they do not lie near the boundary of the two

classes. Conversely, subsets of three points near the class boundary will form vertices for Delaunay triangles.

The collection of the vertices for the Delaunay triangles can be stored in a matrix of edge points. Figure 5.4.4 shows the edge points as vertices of the Delaunay Triangles. Only these edge points are taken for further training.

This method proved to be very effective in finding good edge points without an explicit threshold value.

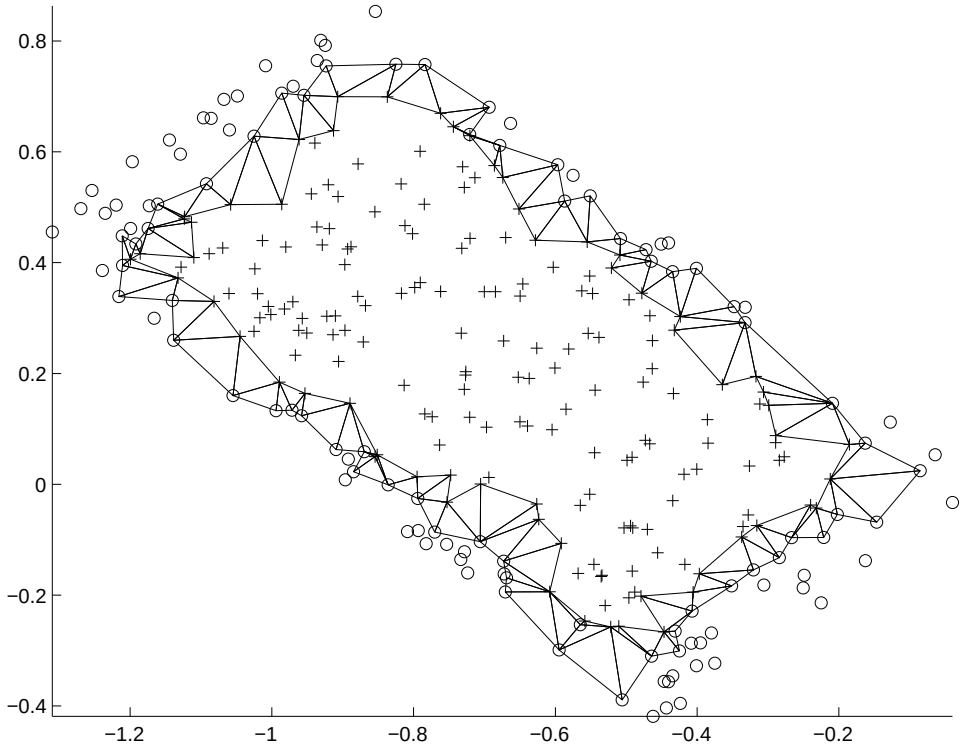

Figure 5.5: Edge points displayed as vertices of triangles.

5.5 Training using Edge Points

Once the edge points have been obtained, subsets are taken one at a time for further processing. A subset here refers to a set of three edge points as described in section 5.4.4. The separating surface for the first subset is found using the Basic Algorithm. Once this hyperplane is obtained, points in the neighboring subset are evaluated against this hyperplane. If they are correctly classified, the next neighboring subset (containing common points as the current subset) is evaluated against this hyperplane and so on. Once some subset that this hyperplane cannot classify correctly is reached, the Basic Algorithm is called to obtain a new hyperplane that classifies this subset correctly and the above process continues until all the edge points are processed and trained. The mid points of the hyperlanes so obtained are found and the final boundary is their convex hull.

5.6 Simulation Results and Computational Aspects

The Nearest Three Points Algorithm was run for 100 runs for several closed convex shape data sets. In each data set 60 percent of the points were taken at random for training and 40 percent for testing. Table 5.4 shows the average results for 100 runs of each closed convex boundary data set. As it can be seen from this table, the overall results are very good: indeed they all exceed 94.5% accuracy, and for some figures the accuracy reaches way into the 97% accuracy. The average time taken for a dataset of 300 points is less than 140 seconds and for a dataset of about 640 points, time taken is around a 304 seconds. For a dataset of 1550 points, the average time taken is 541 seconds. Thus the algorithm is fast, with good

Table 5.3: Nearest Three Points Algorithm: Union of three point sets.

Input:	Training set T of points separable by a convex boundary.
Output:	A convex separating surface for T.
NTPA1.	Find all the boundary subsets of nearest three points.
NTPA2.	Eliminate from the subsets obtained in Step 1, those containing points from the same class. Store the remaining subsets as rows in a matrix of edge points, the Delaunay matrix.
NTPA3.	Starting with the first subset (first row in the Delaunay matrix), the Basic Algorithm is applied to obtain the corresponding hyperplane: h.
NTPA4.	Test h for subsequent subsets and as soon as such a subset is not correctly classified do: **(a)** Group all the subsets correctly classified by h and apply the Basic Algorithm to generate the optimal separating surface for these points; (it is known that they are linearly separable by h but not that h is optimal). **(b)** For the first subset misclassified by h and a new hyperplane is generated and for this the procedure described at steps NTPA3 and NTPA4**(a)** is applied.
NTPA5.	Repeat the above until all edge points are correctly classified.
NTPA6.	Find the mid points of all the hyperplanes generated and the final separating surface as their convex hull.

classification accuracy.

The number of hyperplanes varies between 28 and 33 for training sets of size between 109 and 170. This number increases with the size of data sets. It depends on the dataset and also on the number of linearly separable points that are combined together to form a subset for which the Basic Algorithm generates a final hyperplane. This is the most expensive step of the algorithm (exponential in the number of points).

Data set	Training points	Testing points	Testing points misclassified	Accuracy %	Hyperplanes (no. of)
Rectangle	180	120	3	97.94	32
Rectangle	386	258	7	97.13	49
Rotated Rectangle	300	200	4	97.80	32
Circle	180	120	3	97.00	26
Triangle	180	120	6	94.68	31
Parallelogram	180	120	4	96.88	28

Table 5.4: Simulation Results for various convex closed boundary data sets.

When the union of edge points collected as denoted in step NTPA4**(a)** of the Nearest Three Points Algorithm becomes large, applying the Basic Algorithm leads to an overall slowdown of the entire algorithm. This step ensures that, according to the SVM theory, the hyperplanes generated are all locally optimal. Alternatively, if such an optimality is not a big issue - the hyperplanes generated **are separating the training data anyway** - NTPA4**(a)** can be omitted. The overall time now taken for the algorithm is around 33 seconds for datasets having 300 points and approximately 55 seconds for datasets with 640 points. For a dataset of 1550 points, the average time is 72 seconds. Therefore a trade off is required between the processing speed of the algorithm and the classification accuracy. On the other hand, experiments done by skipping step NTPA4**(a)** of the Nearest Three Points Algorithm did not show a difference in the average classification accuracy for the datasets used in this thesis.

To further asses the behavior of the algorithm a learning curve is obtained. Figure 5.6 shows the learning curve - classification accuracy - when the size of the training set ranges from 30% to 80% of the total dataset. As seen in this figure, the learning rate increases

Training Points	% of Error	Hyperplanes
30% of the DataSet	4.5	20
40% of the DataSet	4.1	23
50% of the DataSet	3.7	27
60% of the DataSet	3.3	30
70% of the DataSet	3.3	31
80% of the DataSet	3.5	35

Table 5.5: Simulation Results showing the Learning Rate.

with the size of the training set when the latter does not exceed 60% of the data set, it is stationary for sizes in between 60% and 70%, after which it decreases. This suggests that when the training set exceeds 70% of the whole data set, overfitting takes places: even though more hyperplanes are generated, accuracy on the test data deteriorates.

5.7 Conclusion

The union of three edge points strategy used in this approach for classification of points eliminates the need for using parameters (e.g. a distance threshold) that has to be given as input in order to collect the most relevant points for training (referred to as edge points in this thesis).

Further, although not analytically proved, it is apparent from limited experiments, that the optimality of the final separation surface, here the convex hull, reflects the optimality of the hyperplanes generated from which it is obtained. More precisely, if step NTPA4**(a)** is applied then the final separating surface is piecewise (locally) optimal. Even when step NTPA4**(a)** is omitted generalization optimality is not compromised.

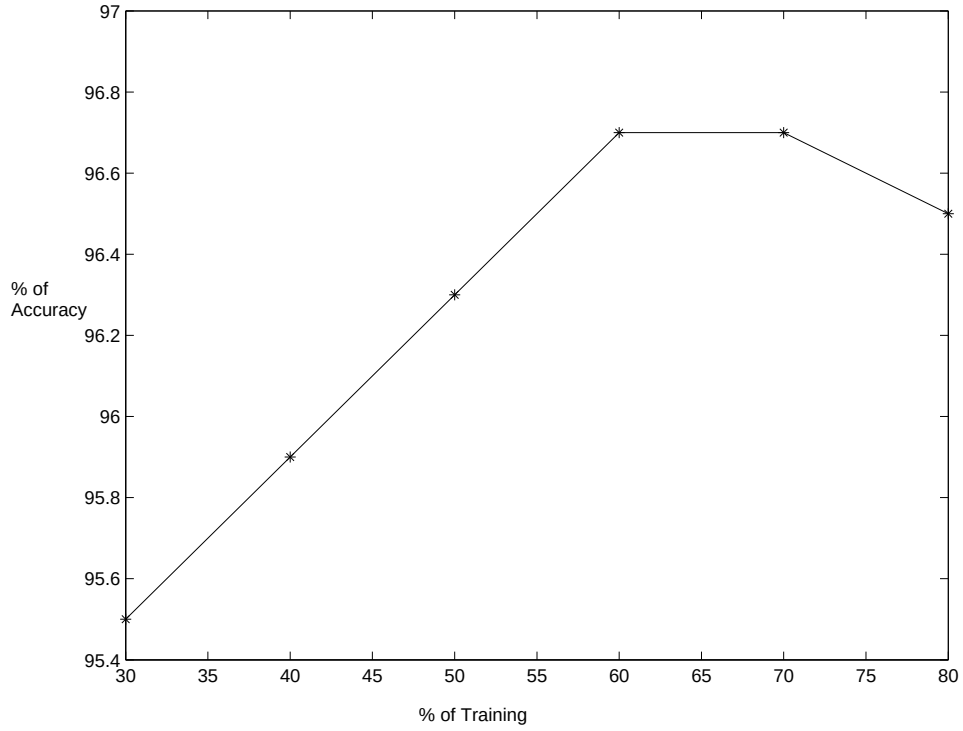

Figure 5.6: Percentage of Training Points used Vs Classification Accuracy.

As shown by experiments for various closed convex boundary classes, the classification accuracy is very high, exceeding in all experiments 94% and in many cases reaching 97%.

It can therefore be concluded that the proposed algorithm overcomes some of limitations of the Basic Algorithm and its iterative version the Iterative SVM algorithm proposed in [3]: eliminate the need for distance threshold, and extending the types of data sets to which it can be applied.

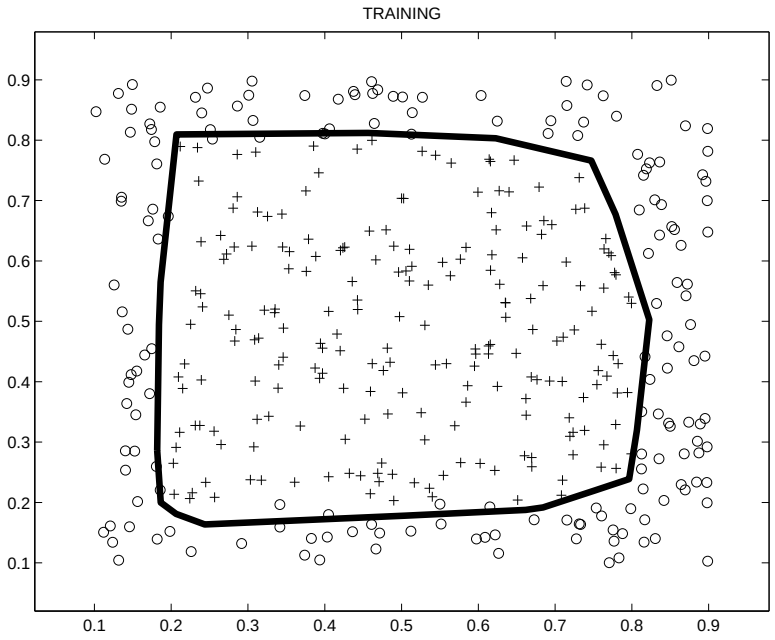

Figure 5.7: Union edge point on a two class rectangle data set.

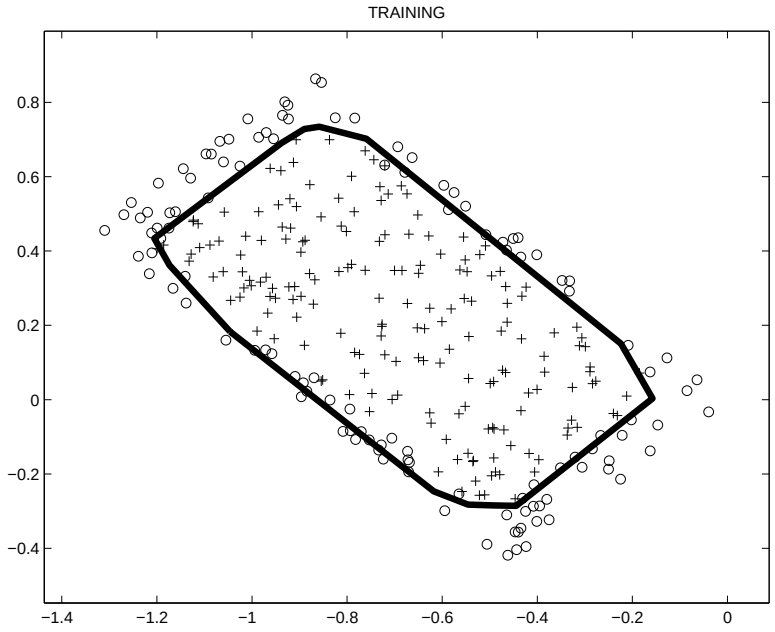

Figure 5.8: Union edge point on a two class rotated rectangle data set.

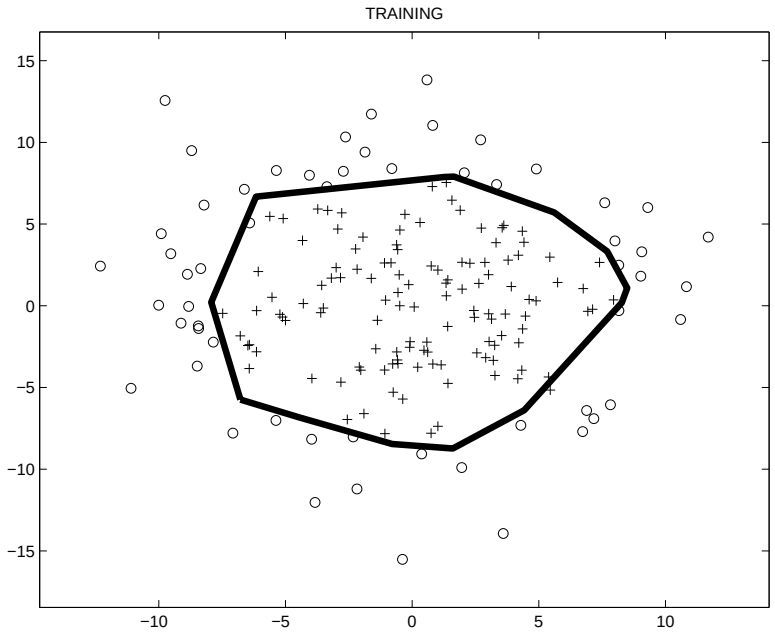

Figure 5.9: Union edge point on a two class circle data set.

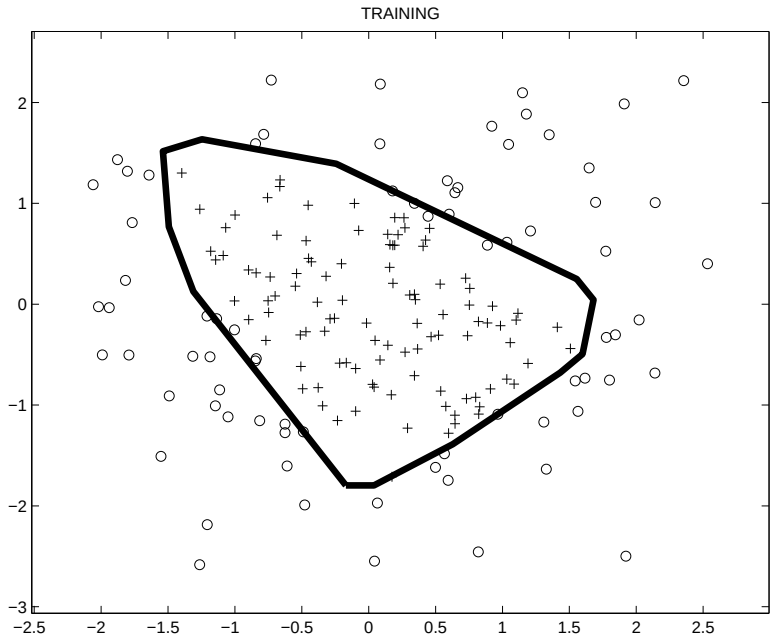

Figure 5.10: Union edge point on a two class triangle data set.

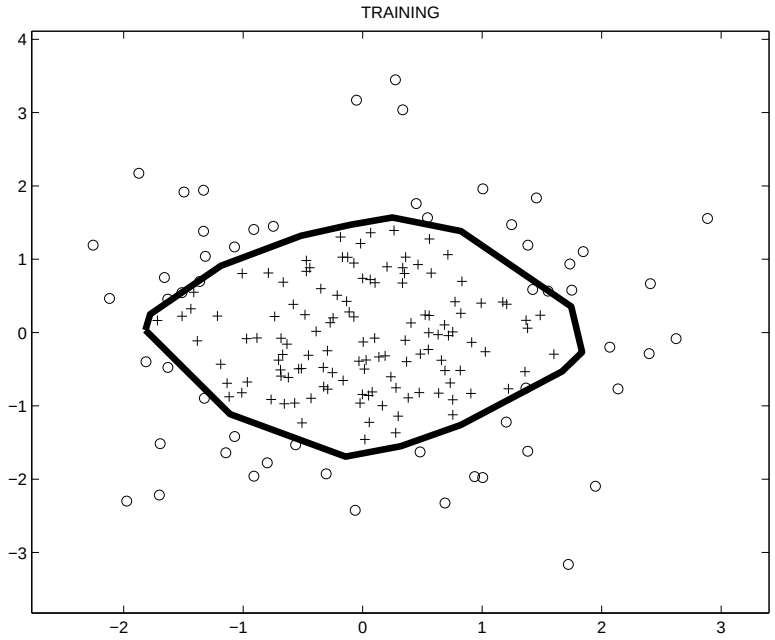

Figure 5.11: Union edge point on a two class parallelogram data set.

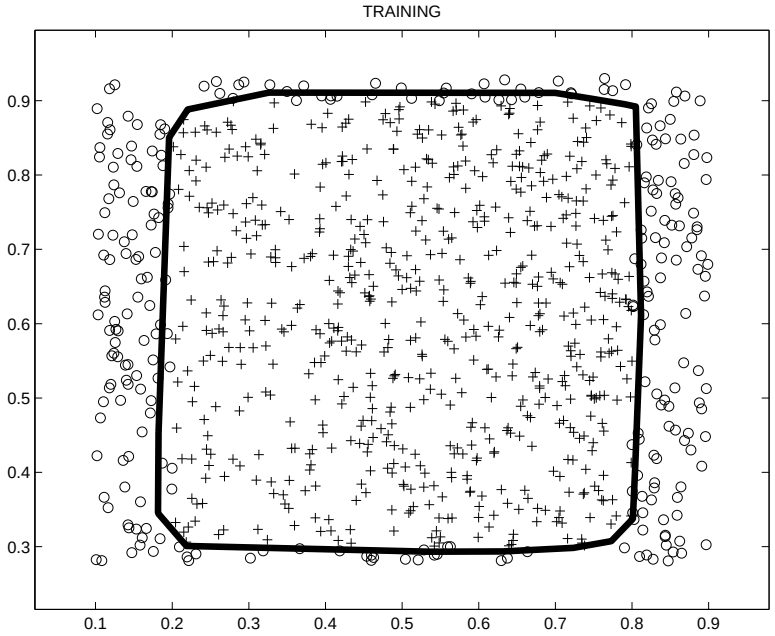

Figure 5.12: Union edge point on a two class rectangular data set of size 1550.

Chapter 6

Summary and Future Work

Support Vector Machines is an actively researched area today. SVMs are nowadays being adapted as a standard tool in several pattern recognition and machine learning applications. This thesis is an attempt to contribute several experiments on classification of closed convex boundary classes, by adapting the hard margin classifier, in the original feature space (without use of kernels).

This research is an extension of the work done in [3] where Piecewise Linear Approximations are applied to simple non-linear datasets and their separating surfaces are found using the hard margin SVM. The Iterative SVM algorithm designed in [3] however could not be applied to complex data sets like closed shapes. The ISVM algorithm was designed such that all data points were processed only along the x-axis. Though the ISVM gave good results for simple non-linear datasets, it needed to be modified in order to apply it to complex closed shape datasets. Further, the ISVM used distance as a threshold to collect the boundary edge points of two class datasets and one of the goals of this thesis was to find more efficient

methods of collecting the points close to the class boundaries.

The use of kernel functions is equivalent to mapping non-linearly separable data sets into higher dimension feature space, in which data are expected to be separable. However, precise criteria for kernel selection are not known. In addition, mapping data into a higher dimension makes it difficult to visualize the data points in the new feature space. By contrast, this research uses Piecewise linear approximations and the separating surfaces are found in the **original feature space**.

Several experiments have been done on two class datasets in order to find a good method to collect the boundary edge points. The Context Dependent Selection of Edge Points : nearest three points approach of collecting edge points used here is a foolproof method. The results of each of these experiments and their suitability or non-suitability for the task at hand has also been described. The Nearest Three Points algorithm uses the edge points collected by this method for training and the simulation results are documented. This algorithm achieves the goal of classifying convex polygonal data sets with a classification accuracy more than 94% for all the datasets.

6.1 Future Work

The following are some areas that require improvements and future work in order enhance the experiments and algorithms proposed in this thesis for a wider variety of problems.

6.1.1 Finding Edge Points for Higher Dimension Data

The method for finding edge points proposed in this research works well for two dimensional and three dimensional data. However this method is not applicable directly for data sets in higher dimensions. For these an extension/generalization of the Delaunay triangulation may be needed.

6.1.2 Classification for any kind of Data set

The algorithm proposed in this research works well for convex polygonal data sets. This is because the final surface inferred is the convex hull of the hyperplanes generated. Future work could be focussed on using a different approach for finding the final surface that enables the algorithm to classify all kinds of data sets.

6.1.3 Enhancing the Basic Algorithm to Process Symbolic Solutions

The Basic Algorithm used in [3] as well as in this work does not take into account the symbolic solutions obtained as a result of solving the system corresponding to 2.12 and 2.14 for $\alpha_i, i = 1, \ldots M$ and b. However, limited experiments show that the true SVM solution is indeed obtained form the symbolic solutions of this system.

Moreover, it would be worthwhile to investigate what is the significance of the solutions to this system corresponding to other values of the objective function.

Bibliography

[1] Donald Tveter, The Pattern Recognition Basis of Artificial Intelligence, IEEE, August 1997.

[2] Anil K. Jain, Robert P.W. Duin, and Jianchang Mao, Statistical Pattern Recognition: A Review, IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 22, No. 1, January 2000.

[3] Rinako Kamei, Experiments in Piecewise Approximation of Class Boundary using Support Vector Machines, University Of Cincinnati, August 2003.

[4] Stuart Russell, Peter Norvig, Artificial Intelligence A Modern Approach, Prentice Hall, January 1995.

[5] Lothar Hermes and Joachim M. Buhmann,Feature Selection for Support Vector Machines, International Conference on Pattern Recognition (ICPR'00), Vol 2, 2712-2715,September 2000.

[6] Bernhard Scholkopf, Alexander.J.Smola, Learning With Kernels, SVMs, Regularization, and Optimization and Beyond, MIT Press, February 2002.

[7] V. Vapnik, The Nature of Statistical Learning Theory. Springer-Verlag, 1995.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, Learnability and the Vapnik-Chervonenkis dimension, Journal of the ACM, Vol 36, No. 4:929–865, October 1989.

[9] Richard O. Duda, Peter E. Hart, David G. Stork, Pattern Classification,John Wiley & Sons Inc, November 2000.

[10] Nello Cristianini, John Shawe-Taylor, An Introduction to Support Vector Machines and other kernel-based learning methods,Cambridge University Press, March 2000.

[11] Rajesh Parekh, Jihoon Yang and Vasant Honavar, Constructive neural network learning algorithms for multi-category classification. Technical Report ISU-CS-TR:95-15a, Department of Computer Science, Iowa State University, 1995.

[12] Dave.F.Watson,Computing the n-dimensional Delaunay tessellation with application to Voronoi Polytopes, The Computer Journal, Vol. 24, No 2, pp. 167- 172, 1981.