Use of Multi-category Proximal SVM for Data Set Reduction

S.V.N. Vishwanathan and M. Narasimha Murty

{vishy, mnm} @csa.iisc.ernet.in
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012
India

Abstract. We present a tutorial introduction to Support Vector Machines (SVM) and try to show, using intuitive arguments, why SVM's tend to perform so well on a variety of challenging problems. We then discuss the quadratic optimization problem that arises as a result of the SVM formulation. We talk about a few computationally cheaper alternative formulations that have been developed recently. We go on to describe the Multi-category Proximal Support Vector Machines (MPSVM) in more detail. We propose a method for data set reduction by effective use of MPSVM. The linear MPSVM formulation is used in an iterative manner to identify the outliers in the data set and eliminate them. A k-Nearest Neighbor (k-NN) classifier is able to classify points using this reduced data set without significant loss of accuracy. We also present geometrically motivated arguments to justify our approach. Experiments on a few publicly available OCR data sets validate our claims.
Keywords: Support Vector Machines, Statistical Learning Theory, VC dimensions, Multi-category Proximal Support Vector Machines, Dataset Reduction.

1 Introduction

k-Nearest Neighbor (k-NN) classifiers are one of the most robust and widely used classifiers in the field of Optical Character Recognition [**?**]. The time required for classification of a test point using a k-NN classifier is linear in the number of points in the training set. One popular method of speeding up the k-NN classifier is to reduce the number of points in the training set by appropriate data selection and *outlier* elimination. It is also well known that the presence of *outliers* tends to decrease the classification accuracy of the k-NN classifier [**?**].

Support Vector Machines (SVM) have recently gained prominence in the field of machine learning and pattern classification [**?**,**?**]. Classification is achieved by realizing a linear or non linear separation surface in the input space. Multi Category Proximal Support Vector Machines (MPSVM), which have been proposed recently, are close in spirit to SVM's but are computationally more attractive [**?**,**?**].

We propose an hybrid classification system, wherein, we first use the MPSVM iteratively to perform data set reduction and outlier elimination. The pre-processed data is now used as the training set for a k-NN classifier. The resultant k-NN classifier is more robust and takes less time to classify test points.

This paper is organized as follows. In section 2 we present a brief introduction to VC theory. We also point out a few shortcomings of traditional machine learning algorithms and show how these lead naturally to the development of SVM's. In section 3 we introduce the linearly separable SVM formulation and discuss its extension to the non linear case in section 4. We try to present a few geometrically motivated arguments to show why SVM's perform very well. We sacrifice some mathematical rigor in order to present more intuition to the reader. While we concentrate our attention entirely on the pattern recognition problem, an excellent tutorial on the use of SVM's for regression can be found in [**?**].

In the second part of the paper we propose a new method for data set reduction using the MPSVM. In section 5 we briefly discuss the MPSVM formulation. We present our algorithm in section 6. In section 7 we discuss the experiments carried out on two widely available OCR data sets. Section 8 concludes with a summary of the present work and gives pointers for future research.

2 VC Theory - A Brief Primer

In this section we introduce the notation and formalize the binary learning problem. We then present the traditional approach to learning and point out some of its shortcomings. We go on to give some intuition behind the concept of VC-dimension and show why capacity is an important factor while designing classifiers. More information can be found in [**?**].

2.1 The Learning Problem

We consider the binary learning problem. Assume that we are given a training set of m labeled patterns

$$X = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_m}, y_m)\} \tag{1}$$

where $\mathbf{x_i} \in R^n$ are the patterns and $y_i \in \{+1, -1\}$ are the corresponding labels. Further, assume that the samples are all drawn i.i.d (Independent and Identically Distributed) from an unknown probability distribution $P(\mathbf{x}, y)$. The goal of building learning machines is to extract some kind of compact abstraction of the data so that we can predict well on unknown samples drawn from the same distribution $P(\mathbf{x}, y)$. In other words we want to learn the mapping $\mathbf{x_i} \rightarrow y_i$ which accurately models $P(\mathbf{x}, y)$. Such a machine may be parameterized by a set of adjustable parameters denoted by α and the

learning function it generates is denoted by $f(\mathbf{x}, \alpha) : R^n \rightarrow \{+1, -1\}$. For example the α's could be the weights on various nodes of a neural network. As is clear different values of α generate different learning functions.

2.2 Traditional Approach to Learning Algorithms

Traditional learning algorithms like the neural networks concentrated their energy on the task of minimizing the empirical error on the training samples [**?**]. The empirical error is given by

$$E_{emp}(\alpha) = \sum_{i=1}^{m} c(f(\mathbf{x_i}, \alpha), y_i) \tag{2}$$

where $f(\mathbf{x_i}, \alpha)$ is the class label predicted by the algorithm for the i^{th} training sample and $c(.,.)$ is some error function. The hope was that, if the training set was sufficiently representative of the underlying distribution, the algorithm would *learn* the distribution and hence *generalize* to make proper predictions on unknown test samples.

But, researchers soon realized that good training set performance did not always guarantee good test set accuracy. For example consider a naive learning algorithm that *remembers* every training sample presented to it. We call such an algorithm a *memory machine*. The *memory machine* of course has 100% accuracy on the training samples but clearly cannot *generalize* on the test set. In other words, what we are asking for is whether the mean of the empirical error converges to the actual error as the number of training points increases to infinity [**?**].

2.3 VC Bounds

The empirical risk for a learning machine is just the measured mean error rate on the training set. The $0-1$ loss function incurs a unit loss for every misclassified sample and does not penalize correctly classified samples. Using such a loss function the empirical risk can be written as

$$R_{emp}(\alpha) = \frac{1}{2m} \sum_{i=1}^{m} |f(\mathbf{x_i}, \alpha) - y_i| \tag{3}$$

where the scaling factor 0.5 has been included to simplify later calculations. Given a training set and a value of α, $R_{emp}(\alpha)$ is fixed. The actual risk which is the mean of the error rate on the entire distribution $P(\mathbf{x}, y)$ can be found by integrating over the entire distribution as

$$R_{actual}(\alpha) = \int \frac{1}{2} |f(\mathbf{x_i}, \alpha) - y_i| dP(\mathbf{x}, y) \tag{4}$$

Let, $0 \leq \eta \leq 1$ be a number. Then, Vapnik and Chervonenkis proved that, for the $0-1$ loss function, with probability $1-\eta$, the following bound holds [**?**]

$$R_{actual}(\alpha) \leq R_{emp}(\alpha) + \phi\left(\frac{h}{m}, \frac{\log(\eta)}{m}\right) \tag{5}$$

where

$$\phi\left(\frac{h}{m}, \frac{\log(\eta)}{m}\right) = m\sqrt{\frac{h(log(2m/h)+1) - log(\eta/4)}{m}} \tag{6}$$

is called the *confidence* term. Here h is defined to be a non negative integer called the Vapnik Chervonenkis (VC) dimension. The VC-dimension of a machine measures the capacity of the machine to learn complex decision boundaries. In the most abstract sense, a learning machine can be thought of a set of functions that the machine has at its disposal. When we are talking of the VC-dimension of a machine we are talking about the capacity of these functions that the learning machine can implement. In the case of binary classifiers, the VC-dimension is the maximal number of points which can be separated into two classes in all possible 2^h ways by the learning machine.

Consider the *memory machine* that we introduced in Section 2.2. Clearly this machine can drive the empirical risk to zero but still does not generalize well because it has a large capacity. This leads us to the observation that, while minimizing empirical error is important, it is equally important to use a machine with a low capacity. In other words, given two machines with the same empirical risk, we have higher confidence in the machine with the lower VC-dimension.

A word of caution is in order here. It is often very difficult to measure the VC-dimension of a machine practically. As a result it is quite difficult to calculate the VC bounds explicitly. The bounds provided by VC theory are often very loose and may not be of practical use. It must also be borne in mind that only an upper bound on the actual risk is available. This does not mean that a machine with larger capacity will always generalize poorly. What the bound says is that, given the training data, we have more confidence in a machine which has lower capacity. In some sense Equation 5 is an restatement of the principle of *Occam's razor*.

2.4 Structural Risk Minimization

Although the bounds provided by VC theory are not tight, we can exploit them in order to do model selection. A *structure* is a nested class of functions S_i such that

$$S_1 \subseteq S_2 \subseteq \ldots \subseteq S_n \subseteq \ldots \tag{7}$$

and hence their VC-dimensions h_i satisfies

$$h_1 \leq h_2 \leq \ldots \leq h_n \ldots \tag{8}$$

Now, because of the nested structure of the function classes

- The empirical risk R_{emp} decreases as the complexity, and hence the VC-dimension, of the class of functions increases
- The confidence bound (ϕ) increases as h increases.

The curves shown in Figure 1 depict Equation 5 and the above observations pictorially.

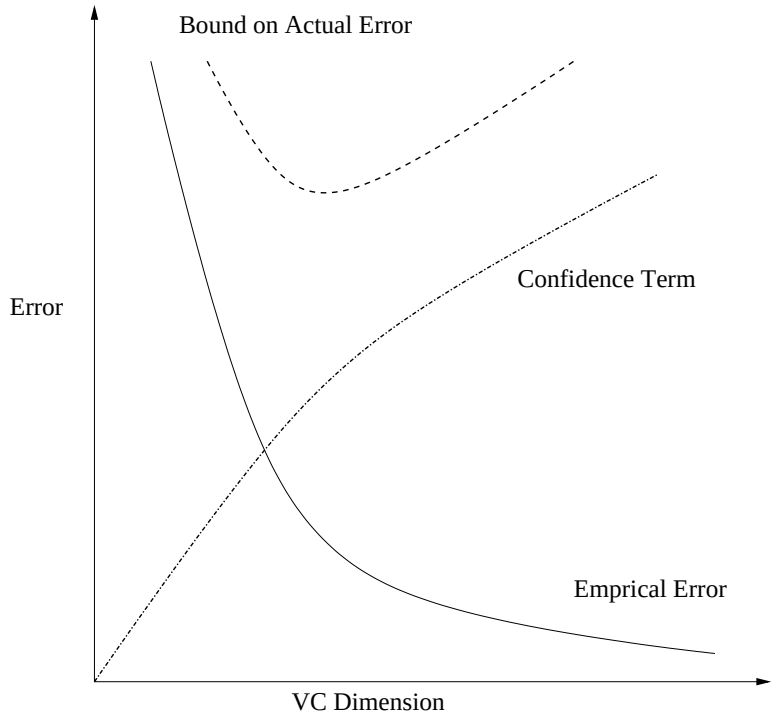

Fig. 1. The heavy dotted line represents the bound on the actual error. It is minimized only when empirical error as well as the confidence term are simultaneously minimized.

These observations suggest a principled way of selecting a class of functions by choosing that class which minimizes the bound on the actual risk over the entire structure. The procedure of selecting the right subset for a given amount of observations is referred to as *capacity control* or *model selection* or *structural risk minimization*.

3 Introduction to Linear SVM's

First, consider a simple linearly separable case where we have two clouds of points belonging to separate classes in 2-dimensions as shown in Figure 2.

There are many linear boundaries that can separate these points but the

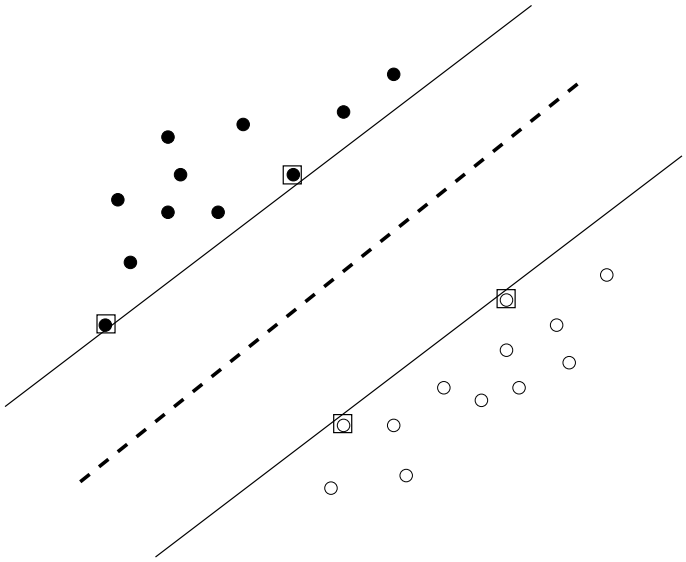

Fig. 2. The empty circles and filled circles belong to two different classes. The dashed line represents the maximally separating linear boundary. The boxed points are Support Vectors.

one that is intuitively appealing is the one that maximally separates the points belonging to two different classes. In some sense we are making the best guess given the limited data that is available to us. It turns out that such a separating hyper plane comes with guarantees on its generalization performance. We show in section 3.3 that selecting such a hyper plane is equivalent to doing structural risk minimization.

3.1 Formulating the Separable Linear SVM Problem

We first formulate the SVM problem for a linearly separable, 2-class problem. As before, $X = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_m}, y_m)\}$ be a set of labeled data points with $\mathbf{x_i} \in R^n$ and $y_i \in \{+1, -1\}$. Further, assume that there is a linear hyper plane parameterized by $(\mathbf{w}, b)$ which separates the points belonging to two different classes. We can write the equation for the hyper plane as

$$\mathbf{w}.\mathbf{x} + b = 0 \tag{9}$$

where $\mathbf{w} \in R^n$ is the normal to the hyper plane and $|b|/||\mathbf{w}||$ is the perpendicular distance of the hyper plane from the origin ($||\mathbf{w}||$ is the Euclidean

norm of $\mathbf{w}$). Let d_+ be the distance of the point closest to the hyper plane with a class label $+1$ and d_- be the corresponding distance for a point with class label -1. We define the margin to be $d_+ + d_-$.

We first show how the problem of maximizing the margin can be expressed as a quadratic optimization problem involving only dot products. Then, we show in Section 4 how we can use the kernel trick to solve this problem in higher dimensions. We require a separating hyper plane that satisfies

$$\mathbf{w}.\mathbf{x_i} + b \geq 1 \tag{10}$$

for all points with $y_i = +1$ and

$$\mathbf{w}.\mathbf{x_i} + b \leq -1 \tag{11}$$

for all points with $y_i = -1$. This can be expressed compactly as

$$y_i(\mathbf{w}.\mathbf{x_i} + b) \geq 1 \quad \forall i = 1, 2, \ldots m \tag{12}$$

Clearly, points which satisfy the above constraint with equality are the closest points to the hyper plane. If $y_i = +1$ then such points lie on the plane

$$H_1 : \mathbf{w}.\mathbf{x_i} + b = 1 \tag{13}$$

which is at a distance of $|1 - b|/||\mathbf{w}||$ from the origin and hence $d_+ = 1/||\mathbf{w}||$. Similarly, if $y_i = -1$ then such points lie on the plane

$$H_2 : \mathbf{w}.\mathbf{x_i} + b = -1 \tag{14}$$

which is at a distance of $|-1-b|/||\mathbf{w}||$ from the origin and hence $d_- = 1/||\mathbf{w}||$. In the simple 2-dimensional case depicted in Figure 2 the boxed points are the closest points to the maximal hyper plane. We can now express our problem of maximizing the margin as

$$\min \ \frac{1}{2}||\mathbf{w}||^2 \quad st \quad y_i(\mathbf{w}.\mathbf{x_i} + b) \geq 1 \quad \forall i = 1, 2, \ldots m \tag{15}$$

where the factor $\frac{1}{2}$ has been introduced to simplify further calculations. A standard technique for solving such problems is to formulate the Lagrangian and solve for the dual problem. To formulate the Lagrangian for this problem we introduce positive Lagrange multipliers $\alpha_i (\geq 0)$ for $i = 1, 2, \ldots m$ and write

$$L(\alpha, \mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i y_i (\mathbf{w}.\mathbf{x_i} + b) + \sum_{i=1}^{m} \alpha_i \tag{16}$$

We want to minimize the above Lagrangian with respect to $\mathbf{w}$ and b which requires that the derivative of $L(\alpha, \mathbf{w}, b)$ with respect to $\mathbf{w}$, and b vanish. Therefore we get

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x_i} \tag{17}$$

and

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{18}$$

Substituting the above results into the Lagrangian gives us

$$L(\alpha, \mathbf{w}, b) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x_i}.\mathbf{x_j} \tag{19}$$

As can be seen, the problem is a convex quadratic optimization problem which involves only the dot products of vectors $\mathbf{x_i}$ and $\mathbf{x_j}$. This is a key observation which will be useful to extend these arguments to the non linear case. Another interesting observation is that, α_i's are non zero for only those data points which lie on either H_1 or H_2. These points are called *Support Vectors* to denote the fact that the solution would have been different if these points had not been present. So in some sense they are *supporting* the current solution. The Support Vectors for our simple 2-dimensional case are shown in Figure 2.

An interesting property of the solution is that it can be expressed completely in terms of the Support Vectors. What it means is that the solution can be specified by only a small fraction of the data points which are closest to the boundary. In some sense the SVM's assign maximum weightage to boundary patterns which intuitively are the most important patterns for discriminating between the two classes.

3.2 Formulating the Non-Separable Linear SVM Problem

Practically observed data is frequently corrupted by noise. It is also well known that the noisy patterns tend to occur near the boundaries [**?**]. In such a case the data points may not be separable by a linear hyper plane. Furthermore we would like to ignore the noisy points in order to improve generalization performance. Consider for example the scenario depicted in Figure 3. If the outlier is also taken into account then the margin of separation decreases and intuitively the solution does not generalize well. In this section we use the ideas of the previous section to extend the formulation to the non separable case. This makes the designed classifier robust to the presence of noisy points. We account for outliers by introducing slack variables $\xi_i \geq 0$ for $i = 1, 2, \ldots, m$ which penalize the outliers [**?**]. We then require

$$\mathbf{w}.\mathbf{x_i} + b \geq 1 - \xi_i \tag{20}$$

for all points with $y_i = +1$ and

$$\mathbf{w}.\mathbf{x_i} + b \leq -1 + \xi_i \tag{21}$$

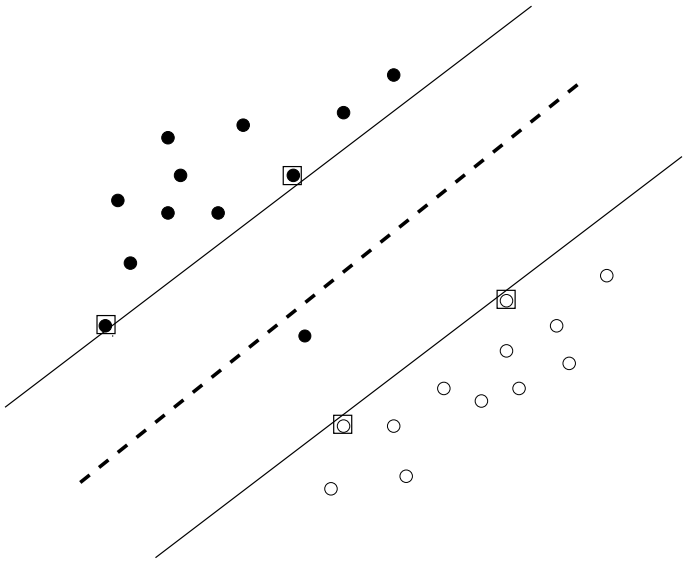

Fig. 3. Ignoring the single outlier increases the margin of separation and leads to better generalization.

for all points with $y_i = -1$. We now minimize the objective function

$$\min \ \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i \tag{22}$$

where C is a penalty factor which controls the penalty incurred by each misclassified point in the training set. On solving the above problem using the same methods as outlined in the previous section we get

$$L(\alpha, \mathbf{w}, b) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \mathbf{x_i}.\mathbf{x_j} \tag{23}$$

with the additional constraint that

$$0 \leq \alpha_i \leq C \quad \forall i = 1, 2, \ldots m \tag{24}$$

which again is a convex quadratic optimization problem. In this case, if $\alpha_i = C$ then we call such a pattern an *error vector*.

Another interesting formulation penalizes the error points quadratically rather than linearly [**?**]. In other words we minimize the objective function

$$\min \ \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i^2 \tag{25}$$

This formulation has been shown to be equivalent to the separable linear formulation in a space that has more dimensions than the kernel space [**?**].

3.3 SVM and SRM

We now try to provide some intuition on how SVM's implement the SRM principle discussed in section 2.4. A ball of radius R around a point $\mathbf{x} \in R^n$ is defined as

$$B_R(\mathbf{x}) = \{\mathbf{y} \in R^n : \| \mathbf{y} - \mathbf{x} \| < R\}. \tag{26}$$

The following lemma from [**?**] relates the margin of separation to the VC-dimension of a class of hyper planes.

Lemma 1. *Let, R be the radius of the smallest ball containing all the training samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$. The decision function defined by a hyper plane with parameters $\mathbf{w}$ and b can be written as*

$$f_{\mathbf{w},b} = \text{sgn}(\mathbf{w}.\mathbf{x} + b) \tag{27}$$

The set $\{f_{\text{w},b} : \| \mathbf{w} \| \leq A, A \in R\}$ has VC-dimension h satisfying

$$h < R^2 A^2 + 1 \tag{28}$$

Recall that the margin of separation between hyper planes is given by $2/ \| \mathbf{w} \|$. A large margin implies a small value for $\| \mathbf{w} \|$ and hence a small value of A, thus ensuring that the VC-dimension of the class $f_{\mathbf{w},b}$ is small. This can be understood geometrically as follows. As the margin increases the number of planes, with the given margin, which can separate the points into two classes decreases and thus the capacity of the class decreases. On the other hand a smaller margin implies a richer class of admissible hyper planes which translates to higher capacity. SVM's also penalize misclassified points in order to ensure that R_{emp} is minimized. Thus, SVM's select the optimal hyper plane which minimizes the bound on the actual risk R_{actual}. This also justifies our intuition that the hyper plane with the largest margin of separation is in some sense the optimal hyper plane.

4 The Kernel Trick for Non Linear Extensions

As we noted in Section 3.1, the final quadratic optimization problem is expressed in terms of dot products between individual samples. Assume that we have a non linear mapping

$$\phi : R^n \rightarrow R^d \tag{29}$$

for some $d >> n$ such that

$$k(\mathbf{x_i}, \mathbf{x_j}) = \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_j}) \rangle \tag{30}$$

and

$$k(\mathbf{x_i}, \mathbf{x_j}) = k(\mathbf{x_j}, \mathbf{x_i}). \tag{31}$$

Then we can write Equation 23 as

$$L(\alpha, \mathbf{w}, b) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x_i}, \mathbf{x_j}) \tag{32}$$

which allows us to work in a $d(>> n)$ dimensional space where the data points are possibly linearly separable. The function k is called as a kernel function.

Now consider the case the data points are not vectors from R^n but are drawn from an arbitrary input domain χ. Suppose, we are able to find a mapping

$$\phi : \chi \rightarrow \mathcal{H} \tag{33}$$

where $\mathcal{H}$ is a Hilbert space. Furthermore, suppose the mapping ϕ satisfies Equations 30 and 31. We can apply our SVM algorithm for classifying such data also. Thus the advantage of using a kernel function is that we can work with non vectorial data as long as the corresponding kernel function is known.

The natural question to ask is whether we can always find a Hilbert space $\mathcal{H}$ and a mapping ϕ such that we can express dot products in $\mathcal{H}$ in the form of Equation 30. It turns out that certain class of functions which satisfy the Mercer's conditions are admissible as kernels. The rather technical condition is expressed as the following two lemmas [**?**,**?**].

Lemma 2. *If k is a continuous symmetric kernel of a positive integral operator K of the form*

$$(Kf)(\mathbf{y}) = \int_C k(\mathbf{x}, \mathbf{y}) f(\mathbf{x})\, d\mathbf{x} \tag{34}$$

with

$$\int_{C \times C} k(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y} \geq 0 \tag{35}$$

for all $f \in L^2(C)$ where C is a compact subset of R^n, it can be expanded in a uniformly convergent series (on $C \times C$) in terms of Eigenfunction ψ_j and positive Eigenvalues λ_j

$$k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}), \tag{36}$$

where $N_F \leq \infty$.

Lemma 3. *If k is a continuous kernel of a positive integral operator, one can construct a mapping ϕ into a space where k acts as a dot product,*

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = k(\mathbf{x}, \mathbf{y}) \tag{37}$$

We refer the reader to Chapter 2 of [**?**] for an excellent technical discussion on Mercer's conditions and related topics.

A few widely used kernel functions are the following:

- Gaussian Kernels of the form

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\| \mathbf{x} - \mathbf{y} \|^2}{2\sigma^2}\right) \tag{38}$$

  with $\sigma > 0$ and
- Polynomial Kernels

$$k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}.\mathbf{y} \rangle)^m \tag{39}$$

  where m is a positive integer.
- Sigmoid Kernels

$$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\langle \mathbf{x}.\mathbf{y} \rangle) + \Theta) \tag{40}$$

  where $\kappa > 0$ and $\Theta < 0$.

Thus, by using kernel functions we can map data points onto a higher dimensional space where they are linearly separable. Note that this mapping is implicit and at no point do we actually need to calculate the mapping function ϕ. As a result all calculations are carried out in the space in which the data points reside. In fact any algorithm which uses similarity between points can be *kernelized* to work in higher dimensional space.

5 Multi Category Support Vector Machines (MPSVM)

Training a SVM involves solving a quadratic optimization problem which requires the use of optimization routines from numerical libraries. This step is computationally intensive, can be subject to stability problems and is non-trivial to implement [**?**]. Attractive iterative algorithms like the SMO, NPA and SSVM have been proposed to overcome this problem [**?**,**?**,**?**,**?**]. These algorithms solve the quadratic optimization problem using various heuristics, some of which are geometrically motivated. Most of these algorithms scale as quadratic or cubic in the input size.

The MPSVM algorithm is motivated by SVM's but is computationally more attractive since it solves a linear optimization problem instead of a quadratic problem. We briefly discuss the MPSVM formulation in this section. Our discussion follows [**?**,**?**].

Let, $X = \{x_1, x_2, \ldots x_m\}$ be a set of m points in n-dimensional real space R^n represented by a $m \times n$ matrix A. We consider the problem of classifying these points according to the membership of each point in the class $A+$ or $A-$ as specified by a given $m \times m$ diagonal matrix D which contains $+1$ or -1 along the diagonal. For this problem, the Proximal Support Vector Machine (PSVM) formulation for a linear kernel with $\nu > 0$ is given by [**?**]

$$\min_{(w,\gamma,y)\in R^{n+1+m}} \quad \frac{\nu}{2}\|Ny\|^2 + \frac{1}{2}\left\| \begin{bmatrix} w \\ \gamma \end{bmatrix} \right\|^2 \qquad s.t \quad D(Aw - e\gamma) + y = e \tag{41}$$

Here e is a vector of ones and N is a diagonal normalization matrix used to account for the difference in the number of points in the two classes. If there are p samples in class $A+$ and $m-p$ samples in $A-$ then, N contains $1/p$ on rows corresponding to entries of class $A+$ and $1/(m-p)$ on rows corresponding to entries of class $A-$. y is an error variable and corresponds to the deviation of a point from the proximal plane.

Applying the KKT conditions and solving the above equations yields

$$w = \nu A^T DN[I - H(\frac{I}{\nu} + H^T NH)^{-1} H^T N] \tag{42}$$

and

$$\gamma = -\nu e^T DN[I - H(\frac{I}{\nu} + H^T NH)^{-1} H^T N] \tag{43}$$

Mangasarian *et. al.* have proposed a non linear extension to the PSVM formulation using the *kernel trick* [**?**]. We do not use the non linear formulation in our experiments.

The MPSVM is a straight forward extension to PSVM where multiple classes are handled by using the one against the rest scheme i.e. samples of one class are considered to constitute the class $A+$ and the rest of the samples are considered to belong to class $A-$, this is repeated for every class in the data set [**?**].

6 Data Set Reduction

Our algorithm for data set reduction is conceptually simple to understand. We reduce the size of the data set in two ways

- Boundary patterns which are most likely to cause confusion while classifying a point are pruned away from the training set.
- Very *typical* patterns of the class which are far removed from the boundary are often not useful for classification and can be safely ignored [**?**].

For each of the classes, we use the MPSVM algorithm to obtain separating planes. For the next iteration we retain only those points which are enclosed by these separating planes. These are points which satisfy

$$x^T w - \gamma \in (-1, 1) \tag{44}$$

This reduced set is used iteratively as input for the MPSVM algorithm. Once the number of data points enclosed by these separating planes becomes less

Algorithm 1 EliminateBoundary(A, D, ν, T, M)

```
Reduced := A
for classLabel=1:No Of Classes do
  ToDelete := Reduced
  for i=1:M do
    [W, γ] = MPSVM(ToDelete, D, ν)
    ToDelete := {x ∈ Reduced : x^T W − γ ∈ (−1, 1)}
    if size(ToDelete) < T then
      break
    end if
  end for
  Reduced := Reduced \ ToDelete
end for
return Reduced
```

than a preset threshold (T) we treat them as boundary points and prune them from the training set. We state this in Algorithm 1.

During the first iteration, all points of a given class which lie on the other side of the separating plane are classified as far removed from the boundary and are removed. Assuming that samples of one class belong to $A+$ and the rest of the samples are classified as $A-$, these are points which satisfy

$$x^T w - \gamma > 1 \tag{45}$$

We state this procedure in Algorithm 2.

Algorithm 2 EliminateTypical(A, D, ν)

```
Reduced := A
for classLabel=1:No Of Classes do
  ToDelete := Reduced
  [W, γ] = MPSVM(ToDelete, D, ν)
  ToDelete = {x ∈ Reduced : x^T W − γ > 1}
  Reduced := Reduced \ ToDelete
end for
return Reduced
```

In both the algorithms, A is the full training set, D is a diagonal matrix indicating the class label of the training samples, T is the threshold on the number of samples to discard, M is the maximum number of iterations to perform per class.

7 Experimental Results

Our experiments were performed using two well known publicly available data sets used widely by us and other researchers [**?**,**?**,**?**].

The first training set consists of 6670 pre-processed hand written digits $(0-9)$ with roughly 667 samples per class. The test set consists of 3333 samples with roughly 333 samples per class. All the samples had a dimensionality of 192 after initial preprocessing and there were no missing values.The second dataset that we used was the well known USPS dataset. It consists of a training set with 7291 images and a test set with 2007 images. Each image has a dimensionality of 256 and there are no missing values.

7.1 *k*-Nearest Neighbor Classifier Using Full Data Set

We performed k-Nearest Neighbor classification of the test set using all the samples in the training set. We experimented with values of k between 1 and 10 and the best classification accuracy for both the datasets is reported in Table 1.

Dataset	% Accuracy	k-value
OCR	92.5	5
USPS	94.469	5

Table 1. Accuracy of k-Nearest Neighbor using complete training set.

7.2 *k*-Nearest Neighbor Classifier Using Reduced Data Set

We used the Linear MPSVM to reduce the training set size as described in Section 6. A few outliers from the OCR data set which were eliminated by our algorithm are shown in Figure 4. We experimented with various values of threshold and ν and report the best results in Table 2 and 3. As the number of training points decreases the density of data points also decreases. We observed that the best classification accuracies are found for lower values of k.

7.3 Discussion of the results

As can be seen, the best accuracy is obtained for the k-Nearest Neighbor classifier which uses the full training set for classification. But, even after elimination of a substantial number of training samples the classification accuracy does not drop significantly. In fact, more than half the samples can be throw away with out significantly affecting test set accuracy. Saradhi *et. al.* report a accuracy of 86.32% on the OCR dataset after elimination of 2390 *atypical* patterns and 75.61% after elimination of 3972 *atypical* patterns [**?**]. Thus, it can be clearly seen that our method is effective in reducing the size of the data set without losing the patterns that are important for classification.

Threshold	ν	No. Eliminated	k-value	% Accuracy
50	1	2177	7	91.749175
100	1	2900	3	90.579058
150	1	3556	1	89.558956
200	1	3796	1	88.898890
250	1	4516	1	86.858686
300	1	4979	1	84.878488
50	5	2229	5	91.929193
100	5	2909	3	90.369037
150	5	3439	1	89.318932
200	5	3875	1	88.748875
250	5	4500	1	87.548755
300	5	4978	1	86.558656

Table 2. Accuracy of k-Nearest Neighbor on the OCR dataset using a reduced training set

Threshold	ν	No. Eliminated	k-value	% Accuracy
50	1	3103	3	94.120578
100	1	3831	1	93.771799
150	1	3980	1	93.423019
200	1	4710	1	92.575984
250	1	4886	1	92.426507
300	1	5126	1	92.326856
50	5	3083	1	94.020927
100	5	3802	5	93.373194
150	5	4112	1	93.124066
200	5	4424	1	92.924763
250	5	4851	1	92.526158
300	5	5198	1	91.778774

Table 3. Accuracy of k-Nearest Neighbor on the USPS dataset using a reduced training set

Another interesting feature is that as the data set size reduces the best classification accuracies are found for smaller k values. This can be explained by the fact that the density of training points in a given volume is decreasing because of dataset pruning. As a result the number of neighbors of a point in a given volume is also decreasing and thus results in better classification accuracies for lower values of k.

8 Conclusion

We have proposed a conceptually simple algorithm for data set reduction and outlier detection. In some sense the algorithm comes closer to methods like

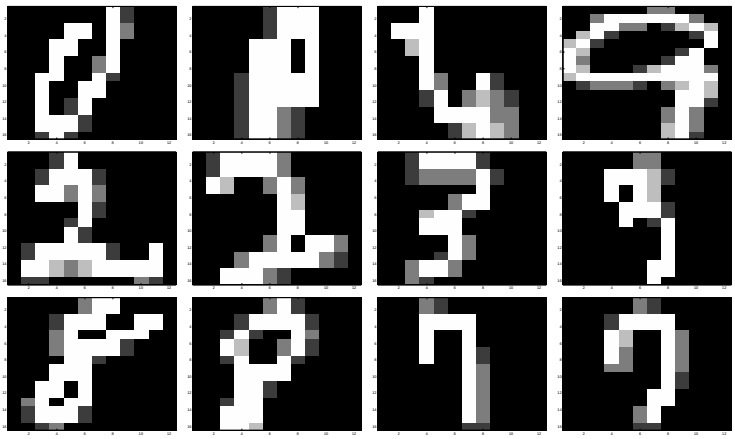

Fig. 4. Examples of outliers detected by our algorithm. Actual class labels (L to R) are 0, 0, 4, 9 (Row1) 2, 2, 3, 3 (Row 2) 8, 8, 7, 7 (Row 3).

bootstrapping which work by increasing the separation between the classes. Our algorithm increases separation between classes by eliminating the noisy patterns at the class boundaries and thus leads to better generalization. It also identifies the typical patterns in the dataset and prunes them away leading to a smaller data set.

One immediately apparent approach to extend our algorithm is to use the Non Linear MPSVM formulation which may lead to better separation in higher dimensional kernel space. At the time of writing results of this approach are not available. The main difficulty in implementing this approach seems that even a rectangular kernel of modest size requires a large amount of memory because of the large dimensionality of the problem. Use of some dimensionality reduction algorithm like the one proposed by us may be explored to overcome this limitation [**?**].

Saradhi *et. al.* report good results by applying data set reduction techniques after bootstrapping the data points [**?**]. This is an area of further investigation.

References

1. C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
2. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
3. R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, Inc, New York, 1953.
4. R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 2. Interscience Publishers, Inc, New York, 1962.

5. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 2001. Second edition.
6. T.-T. Frieß, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In J. Shavlik, editor, *Proceedings of the International Conference on Machine Learning*, pages 188–196. Morgan Kaufmann Publishers, 1998.
7. S. Haykin. *Neural Networks : A Comprehensive Foundation*. Macmillan, New York, 1994.
8. S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. Technical Report Technical Report TR-ISL-99-03, Indian Institute of Science, Bangalore, 1999.
9. J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.
10. B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.
11. P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 50–58, San Mateo, CA, 1993. Morgan Kaufmann Publishers.
12. A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 2001. Forthcoming.
13. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.
14. S.V.N. Vishwanathan and M.N. Murty. Geometric SVM: A fast and intuitive SVM algorithm. In *Proceedings of the ICPR*, 2002. accepted.