Department of Electrical
and
Computer Systems Engineering

Technical Report
MECSE-4-2003

Fast Linear Optimisation with Automatically Biased Support Vector Machines

D.Lai, N.Mani and M.Palaniswami

MONASH
UNIVERSITY

Fast Linear Optimisation with Automatically Biased Support Vector Machines

D. Lai*, N. Mani*, M.Palaniswami+
*Dept. of Electrical and Computer Systems Engineering
Monash University, Clayton, Vic. 3168, Australia.
+Dept. of Electrical and Electronic Engineering
The University of Melbourne, Vic. 3010, Australia.

Abstract: We propose a new Support Vector Machine classifier formulation which allows for an automatic computation of the bias and eliminates the equality constraint. We also present a new training algorithm, which is capable of providing fast training for our automatically biased SVM. We then show that this method allows for the application of acceleration methods which further increases the rates of convergence. Comparisons between our agorithm to the well-known Sequential Minimal Optimization (SMO) algorithm are also made.

1. INTRODUCTION

In recent years, Support Vector Machines have gained increasing attention as a new supervised machine learning formulation. The difference between Support Vector Machines and existing machine learning methods such as Neural Networks (NN) is that instead of being based on the traditional Empirical Risk Minimization (ERM) principle, it performs Structural Risk Minimization (SRM). Since, the introduction of SVMs many researchers have applied it to non-linear problems with remarkable results. The formulation is well understood, but practical use for online implementations has not been widespread due to excessive training times. The optimization speed scales with the size of the dataset. Several iterative optimisation techniques have been employed which include methods such as gradient ascent/decent methods[1], Sequential Minimal Optimization(SMO)[6], Successive Over Relaxation(SOR)[8] and so on. Faster optimisation methods suffer from less than optimal solutions and more elegant methods require increased code complexity.

In this report, we propose an automatically biased SVM formulation (ASVM), which eliminates the need to enforce an equality constraint at each optimization step. It has been shown that the main source of inefficiency in optimisation programs like the Sequential Minimal Optimization algorithm was the search for the second point to update[7]. This slow down was due to the algorithm maintaining a single threshold, b and getting confused. An improved SMO was proposed and it involved using two thresholds instead of one. We note that the problems observed due to the setting of the threshold is closely related to the strict enforcing of the equality in the dual formulation. We would like to remove this equality constraint from the problem but continue to retain the sparseness of our Support Vector solution. Our argument is that the removal of the equality constraint would allow the solution to be derived from solving a group of linear equations while enforcing only the inequality constraint. We employ a modified linear stationary method known as the Jacobi Method. We show how the optimisation speed can be further increased by correctly applying acceleration methods such as the extrapolation method. This document is divided into the following sections; section 2 will describe the pattern

recognition SVM formulation and our ASVM variant, section 3 will describe our iterative method and the remaining sections will be left for our experimental results and discussion.

2. SUPPORT VECTOR MACHINE CLASSIFIER FORMULATION

2.1 VAPNIK'S SUPPORT VECTOR MACHINE CLASSIFIER

Given a training set, $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$ where $\mathbf{x}_i$ is an attribute vector and y_i is the corresponding class marker, the task is then to train a machine to learn the non-linear relationship between the attributes and their respective markers. i.e. y = f(x).The **X** vector space is referred to as the *input space* and consists of the attribute vectors of dimension, N meaning that we are including N different measurements of the specific problem. This means that the input space is of dimension N or $\Re^N$. The **Y** vector consists of scalars, y_i that are usually +1 or –1 and referred to as *labels* or *markers*. Thereafter the dataset is defined as,

$$\begin{aligned} &\mathbf{D}=\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2)...(\mathbf{x}_l, y_l)\} \\ &\mathbf{x}_i \in \mathbb{R}^N \\ &y_i = \{1, -1\} \end{aligned} \tag{1.1}$$

If our training set is not linearly separable, as is the case in most real life applications, we define a non-linear mapping from input space to some higher dimensional feature space, denoted by $\varphi : \Re^N \rightarrow \Re^M$, so that the points in feature space map via**,** $\mathbf{x}_F$ **:** $\mathbf{x} \rightarrow \varphi(\mathbf{x})$. We then construct a linear discriminant function in feature space so that,

$$\begin{aligned} &f(\mathbf{x}) = \sum_{i=1}^{l} w_i \varphi_i(\mathbf{x}) + b \\ &f(\mathbf{x}) > 0 \qquad \forall \quad i : y_i = +1 \\ &f(\mathbf{x}) < 0 \qquad \forall \quad i : y_i = -1 \end{aligned} \tag{1.2}$$

The hyperplane that separates the two classes is the decision surface defined by,

$$\sum_{i=1}^{l} w_i \varphi_i(\mathbf{x}) + b = 0 \tag{1.3}$$

where **w** is a vector space of weights and **x** is the corresponding input space vector. The bias, b determines the position of the hyperplane in feature space. Fig 2.0 is a simple toy example of data containing two attributes plotted in a R^2 dimensioned input space. The separating hyperplane is the line $f(x) = 0$. It can be seen that there are an infinite number of such separating hyperplanes.

It turns out that only the points closest to the separating hyperplane are important, and these points are referred to as Support Vectors. The distance from the hyperplane to a support vector is $\frac{1}{\|\mathbf{w}\|}$ and the distance between the support vectors of one class to the other

class is simply $\frac{2}{\|\mathbf{w}\|}$ by simple geometry. The task is then to derive the best possible hyperplane with the maximum possible margin between the classes. The SVM classifier problem is then,

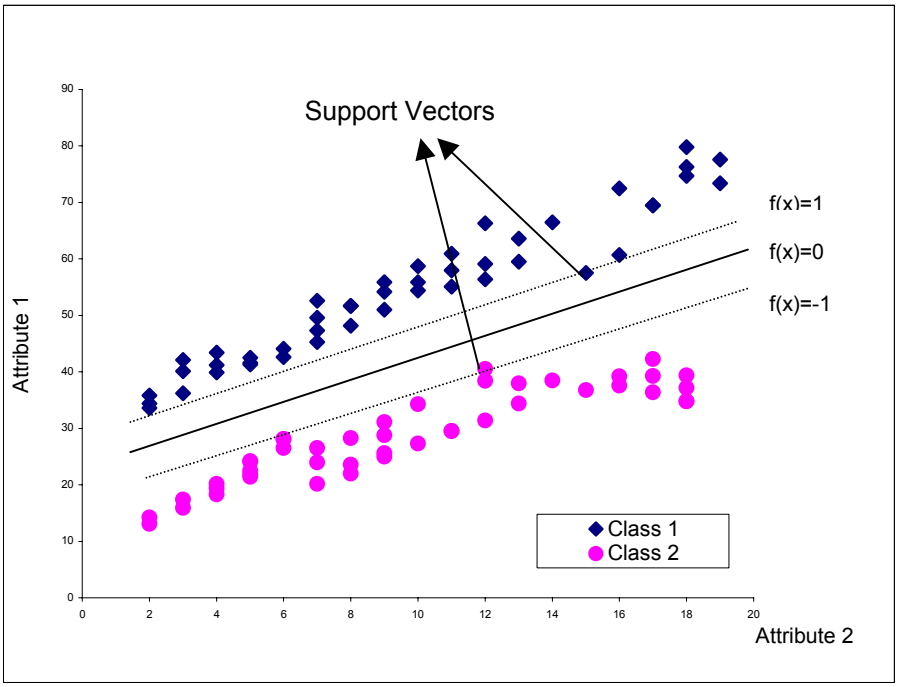

Fig 1. 1 Separating hyperplane in $\mathbb{R}^2$ space with support vectors shown

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$y_i \left(\left(\sum_{i=1}^{l} w_i \varphi_i(\mathbf{x}) + b\right) > 1 \qquad \forall i = 1..l \right. \tag{1.4}$$

In practice, even separation in feature space is not perfect. It is possible to account for this by introducing a penalizing term to the problem to allow for misclassified data. We do this by introducing slack variables, ξ to relax the equality constraint in (1.4) giving the following problem to be solved;

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\mathbf{1}^T\boldsymbol{\xi}$$

subject to:

$$y_i \left(\left(\sum_{i=1}^{l} w_i \varphi_i(\mathbf{x}) + b\right) > 1\text{-}\xi_i \qquad \forall i = 1..l \qquad \xi_i > 0 \right. \tag{1.5}$$

The parameter C is a regularizer which controls the tradeoff between generalization capability and number of misclassifications. Large C reduces the number of classification errors but gives poorer generalization capabilities. Lagrange Theory describes how optimization of an objective function subject to equality and inequality constraints is to be

done. Given an objective function, f(**w**), equality constraints g(**w**) and inequality constraints h(**w**), the Lagrange function is defined as

$$\mathrm{L}(\mathbf{w}, \boldsymbol{\alpha}, b) = f(\mathrm{w}) + \sum_{\mathrm{i}=1}^{l} \beta_i g_i(w) + \sum_{\mathrm{j}=1}^{l} \alpha_j h_j(w) \tag{1.6}$$

$\alpha, \beta = \text{Lagrange Multipliers}$

With that in mind, we note the inequality constraint in (1.5) and write the Lagrange Primal problem,

$$\underset{\text{minimize } \mathbf{w}, \boldsymbol{\alpha}, b, \xi}{\mathrm{L}(\mathbf{w}, \boldsymbol{\alpha}, b, \xi)} = \frac{1}{2}\sum_{i=1}^{l} w_i^2 + \frac{C}{2}\sum_{i=1}^{l}\xi_i + \sum_{i=1}^{l}\alpha_i(y_i(\sum_{j=1}^{l} w_j\varphi(x_j) + b) - 1 + \xi) \tag{1.7}$$

The primal form of (1.7) offers a dual representation, found by differentiating (1.7) with respect to the variables and finding the stationary conditions,

$$\frac{dL(\mathbf{w}, \boldsymbol{\alpha}, b, \xi)}{d\mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} \alpha_i y_i \varphi(\mathbf{x}_i) = \mathbf{0} \tag{1.8}$$

$$\frac{dL(\mathbf{w}, \boldsymbol{\alpha}, b, \xi)}{d\boldsymbol{\alpha}} = \mathbf{y}(\sum_{i=1}^{l} w_i\varphi_i(\mathbf{x}) + b) - 1 + \xi = 0 \tag{1.9}$$

$$\frac{dL(\mathbf{w}, \boldsymbol{\alpha}, b, \xi)}{db} = \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{1.10}$$

$$\frac{dL(\mathbf{w}, \boldsymbol{\alpha}, b, \xi)}{d\xi} = C\boldsymbol{\xi} - \boldsymbol{\alpha} = \mathbf{0} \tag{1.11}$$

Then substituting them back into (1.7), we obtain

$$\underset{\text{minimize } \boldsymbol{\alpha}}{\mathrm{L}(\boldsymbol{\alpha})} = -\sum_{\mathrm{i}=1}^{l} \alpha_i + \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \left\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \right\rangle \tag{1.12}$$

subject to: $0 < \alpha_{\mathrm{i}} < C$

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

By taking the negative of (1.12), we obtain the more commonly used Wolfe dual representation,

$$\underset{\text{maximize } \alpha}{\mathrm{W}(\boldsymbol{\alpha})} = \sum_{\mathrm{i}=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \left\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \right\rangle$$

subject to: $0 < \alpha_{\mathrm{i}} < C$

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{1.13}$$

Using Mercer's Theorem, we can compute the inner products of the vectors $\varphi(\mathbf{x})$ using a kernel function that is,

$$K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \right\rangle \tag{1.14}$$

This *kernel trick* is a powerful method, which implicitly computes the inner product of the vectors without the need to explicitly define the mapping $\mathbf{x} \mapsto \varphi(\mathbf{x})$ of inputs to higher dimensional space, which usually is of infinite dimension. After optimization, substituting for **w** into (1.11) gives the decision function as,

$$f(x) = \text{sign}(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b) \tag{1.15}$$

It should be noted that the solution is ***sparse***, meaning that a lot of $\alpha_i = 0$ and the decision function, $f(x)$ could be represented solely by the Support Vectors, (i.e. $\alpha_i \neq 0$).

2.2 PROPOSED AUTOMATICALLY BIASED SUPPORT VECTOR MACHINES

It has been shown in **(1.2)** that we can define a feature space via a non-linear mapping, $\phi(x)$. We propose to remove the bias, *b* by defining an augmented feature mapping φ_A: $\Re^N \rightarrow \Re^{M+1}$, where $\varphi_A = \{\varphi_j: \Re^N \rightarrow \Re^{M+1} | \ 0 \leq j \leq M\}$. The resulting augmented feature space is identical to the original feature space defined previously with the restriction that $\varphi_0 = \beta$. Then, we can define a similar decision surface as,

$$\begin{aligned} f(\mathbf{x}) &= \sum_{i=1}^{l} w_{Ai} \varphi_i(\mathbf{x}) \\ f(\mathbf{x}) &> 0 \qquad \forall \quad i : y_i = +1 \\ f(\mathbf{x}) &< 0 \qquad \forall \quad i : y_i = -1 \end{aligned} \tag{1.16}$$

with the hyperplane now defined by:

$$\mathbf{w}_A \varphi_i(\mathbf{x}) = 0 \tag{1.17}$$

where setting :

$$\mathbf{w}_A = \begin{bmatrix} \frac{b}{\beta} \\ \mathbf{w} \end{bmatrix} \tag{1.18}$$

Instead of having a bias,*b* in the estimator form of our hyperplane, we include the bias as a weighted sum by setting $w_0 = \frac{b}{\beta}$ and the number of weights for our *l* sample space is increased to *l+1*.

The classification problem is then simply,

$$\text{minimize} \quad \frac{1}{2}\left\|\mathbf{w}_A\right\|^2 + C\mathbf{1}^T\boldsymbol{\xi}$$

subject to:

$$y_i \left(\sum_{i=1}^{l} w_{Ai}\varphi_i(\mathbf{x}) \right) > 1\text{-}\xi_i \qquad \forall i = 1..l \qquad \xi_i > 0 \tag{1.19}$$

Applying the same steps as before, we obtain;

Lagrangrian Primal

$$\underset{\text{minimize } \mathbf{w}_A,\,\boldsymbol{\alpha},\,\xi}{\mathrm{L}\,(\,\mathbf{w}_A, \boldsymbol{\alpha}, \xi\,)} = \frac{1}{2}\sum_{i=1}^{l} w_{Ai}^2 + \frac{C}{2}\sum_{i=1}^{l}\xi_i + \sum_{i=1}^{l}\alpha_i\left(y_i\sum_{j=1}^{l} w_{Aj}\varphi(x_j) - 1 + \xi\right) \tag{1.20}$$

Lagrangrian Dual

$$\underset{\text{minimize } \boldsymbol{\alpha}}{\mathrm{L}\,(\boldsymbol{\alpha})} = \sum_{i=1}^{l}\alpha_i^2 - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j\left(\left\langle\varphi(x_i),\varphi(x_j)\right\rangle + \beta^2\right)$$

The dual of this form could be written in the form of the augmented kernel function, $K_A(\mathbf{x}_i , \mathbf{x}_j)$ where;

$$\begin{aligned} K_A(\mathbf{x}_i,\mathbf{x}_j) &= \left\langle\varphi(\mathbf{x}_i),\varphi(\mathbf{x}_j)\right\rangle + \beta^2 \\ &= K(\mathbf{x}_i,\mathbf{x}_j) + \beta^2 \end{aligned} \tag{1.21}$$

The resulting decision surface for the classifier formulation is then:

$$f(x) = \operatorname{sign}\left(\sum_{i=1}^{l}\alpha_i y_i K_A(x_i, x)\right) \tag{1.22}$$

In (1.22), we can see that the bias term, *b* in **(1.15)** has been incorporated into the kernel function and does not require to be computed separately as suggested by. Optimization of this form will automatically bias the hyperplane in feature space and avoid the unnecessary extra computation for the threshold. The quadratic programming problem now does not require the enforcing of the equality constraint in **(1.12)**.

We note that for the given training set, **D** the maximization of the dual Lagrange form is implicitly formulated in equation and the problem is then to find the $\boldsymbol{\alpha}$ vector such that for all training examples the following is satisfied within a tolerance δ,

$$\begin{aligned} &y_i f(x_i) > 1+\delta && \text{for} \quad \alpha_i = 0 \\ &1-\delta < y_i f(x_i) < 1+\delta && \text{for} \quad 0<\alpha_i<C \\ &y_i f(x_i) < 1-\delta && \text{for} \quad \alpha_i = C \end{aligned} \tag{1.23}$$

We note that the automatically biased SVM form does not have the equality constraint and for a training set, we can write the conditions (1.23) in matrix form,

$$\mathbf{Gu} = \mathbf{Y} + \mathbf{1}\delta \tag{1.24}$$

where $\mathbf{G} = \{ K_A(\mathbf{x}_i , \mathbf{x}_j) \}_{i,j=1}^{s}$

$\mathbf{u} = \{ \alpha_1, \alpha_2 ... \alpha_s \}$

$\mathbf{Y} = \{ y_1, y_2 .... y_s \}$

We will show in next section, how this form is suitable for application of our iterative method, with the addition of enforcing at each step:

$$u_i^{t+1} \Leftarrow \begin{cases} C & \text{if } u_i^{t+1} > C \\ u_i^{t+1} & \text{if } 0 < u_i^{t+1} < C \\ 0 & \text{if } u_i^{t+1} < 0 \end{cases}$$

We point out that the optimal solution will be in the form (1.22) and it is a trivial matter to obtain **(1.15)** by substitution. However, we show next that this is not required for we can control the sparseness of our solution by the parameter β^2.

2.3 EFFECT OF THE PARAMETER β^2

We present below a simple analysis of the effect of the parameter β^2 by recomputing the threshold b. We note that it is possible to give a graphical interpretation of this parameter as well as a rigourous statistical analysis of its effect on the optimal estimator $f(x)$, however we will leave that for a later discussion.

Consider the optimal decision function $g(\mathbf{y})$,

$$g(\mathbf{y}) = \sum_{i=1}^{n} \alpha_i^* y_i K_A(\mathbf{x}_i, \mathbf{y}) \tag{1.25}$$

where α^* denotes an optimal solution within a tolerance range, δ

We ignore points that are well classified and concentrate on points that are closest to the hyperplane, or the Support Vectors. In fact it has been shown that (1.25) can be written entirely in terms of Support Vectors only,

$$g(\mathbf{y}) = \sum_{i \in SV} \alpha_i^* y_i K_A(\mathbf{x}_i, \mathbf{y}) \tag{1.26}$$

Then, for every Support Vector $\mathbf{x}_{sv}$, the following holds,

$$y_i g(\mathbf{x}_{sv}) - 1 < \delta$$

Expanding we get,

$$\Rightarrow y_i \left(\sum_{i \in SV} \alpha_i^* y_i K_A(\mathbf{x}_i, \mathbf{y}) \right) - 1 < \delta$$

$$\Rightarrow y_i \left(\sum_{i \in SV} \alpha_i^* y_i \left(K(\mathbf{x}_i, \mathbf{y}) + \beta^2 \right) \right) < \delta + 1$$

which holds for a particular value of β^2.

We make a substitution for $K(\mathbf{x}_i, \mathbf{x}_{sv}) = \tau_i \beta^2$

$$\Rightarrow y_i \left(\sum_{i \in SV} \alpha_i^* y_i \left(\tau\beta^2 + \beta^2 \right) \right) < \delta + 1$$

$$\Rightarrow y_i \left(\sum_{i \in SV} \alpha_i^* y_i \left(\tau + 1 \right) \right) < \frac{\delta + 1}{\beta^2}$$

At a fixed tolerance, δ and after some algebra we get,

$$\Rightarrow \sum_{i \in SV} \alpha_i^* y_i < \frac{y_i (\delta + 1)}{\beta^2} - \sum_{i \in SV} \alpha_i^* y_i \tau \qquad (1.27)$$

Note that the contribution of the rightmost term decreases due to

$$\tau_i = \frac{K(\mathbf{x}_i, \mathbf{x}_{sv})}{\beta^2}$$

and we would expect $\sum \alpha^*_i y_i \tau_i \approx 0$ as β^2 becomes larger so that (1.27) approximately becomes

$$\sum_{i \in SV} \alpha_i^* y_i \approx \frac{y_i (\delta + 1)}{\beta^2}$$

Differentiating with respect to β^2, we derive the approximate gradient at large β^2

$$\Rightarrow \frac{\partial \sum_{i \in SV} \alpha_i^* y_i}{\partial \beta^2} \approx -\frac{y_i (\delta + 1)}{\left(\beta^2 \right)^2}$$

We can determine the approximate range of β^2 to use for optimal solution by setting the gradient to be less than a small threshold parameter, η.

$$\Rightarrow \frac{y_i(\delta+1)}{\left(\beta^2\right)^2} < \eta$$
$$\Rightarrow \beta^2 > \sqrt{\frac{y_i(\delta+1)}{\eta}} \tag{1.28}$$

Lemma 1: *The ASVM formulation becomes the standard SVM classifier formulation when* $\beta^2 \to \infty$ *and* $\sum_{i=1}^{l} \alpha_i y_i \to 0$.

Proof : Expanding (1.19) and using (1.18), we obtain the maximization problem as

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\left(\frac{b}{\beta^2}\right) + C\mathbf{1}^T\boldsymbol{\xi}$$

subject to:

$$y_i \left(\sum_{i=1}^{l} w_i \varphi_i(\mathbf{x}) + b \right) > 1 - \xi_i \qquad \forall i = 1..l \qquad \xi_i > 0$$

It is easy to see then that as $\beta^2 \to \infty$, the problem just reduces to **(1.5)**□

3. ITERATIVE LINEAR STATIONARY METHODS

The basic iterative linear stationary method of first degree is applicable to a system of q linear equations written in the following matrix form,

$$\mathbf{Hu} = \mathbf{F} \tag{1.29}$$

where:

$\mathbf{H} = qxq$ matrix

$\mathbf{u} = \{u_1, u_2 ..... u_q\}$

$\mathbf{F} = \{f_1, f_2 .... f_q\}$

The vector **u** consists of the unknowns while **F** is usually the boundary values or equation values. The matrix **H** contains the coefficients of the linear equations. We note here that this form is identical to (1.24) and hence allows us to apply it directly to solve the SVM optimization problem.

The specific updates can be written in the following form,

$$\mathbf{u}^{t+1} = \mathbf{B}\mathbf{u}^t + \mathbf{k} \tag{1.30}$$

Proposition 1: There exists a solution, **u*** to the related system of real equations expressed as, (**I** – **B**)**u** = **k ,** if and only if it is unique to **u*** = $\mathbf{H}^{-1}\mathbf{F}$**.**

Proof: Let **B** = **I** – $\mathbf{Q}^{-1}\mathbf{H}$ and **k**=$\mathbf{Q}^{-1}\mathbf{F}$ where Q is some non-singular matrix, then substituting into the equation above, we get

$$(\mathbf{I} - (\mathbf{I} - \mathbf{Q}^{-1}\mathbf{H}))\mathbf{u} = \mathbf{Q}^{-1}\mathbf{F}$$

$$\mathbf{u} = \mathbf{H}^{-1}\mathbf{F}$$

If both **H** and **F** are real then it is simple to see that **u** must also be real and unique. □

3.1 The Basic Jacobi Method

We utilize the basic Jacobi method to demonstrate its effectiveness in obtaining a fast solution for the Support Vector Machine problem. This technique is a matrix splitting method. The method is applicable to a system of equations partitioned in the following form,

$$\begin{pmatrix} H_{11} & \cdots & H_{1q} \\ \vdots & \ddots & \vdots \\ H_{q1} & \cdots & H_{qq} \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_q \end{pmatrix} = \begin{pmatrix} F_1 \\ \vdots \\ F_q \end{pmatrix} \tag{1.31}$$

where $H_{ij} = n_i \; x \; n_j$ sub matrix with the condition

$$\sum_{i=1}^{q} n_i = q$$

For our SVM problem, we choose H_{ij}= 1 x 1 matrix or a scalar value. This is sometimes referred to as the Jacobi *point* form. The matrix **H** can then be expressed as a sum,

$$\mathbf{H} = \mathbf{D} + \mathbf{L} + \mathbf{L}^T \tag{1.32}$$

Where **D** is a diagonal matrix containing the diagonal elements of **H** and **L** is a strictly lower triangle matrix containing the lower elements of **H**.

$$D = \begin{pmatrix} H_{11} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & H_{qq} \end{pmatrix}$$

$$L = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ H_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ H_{q1} & \cdots & H_{qq-1} & 0 \end{pmatrix}$$

The iterates of the Jacobi Method are then given by the following update rule

$$H_{ii}\ u_i^{t+1} = -\sum_{i,j=1}^{q} H_{ij}u_j^t + F_i \qquad \forall i = 1...q \tag{1.33}$$

This can be rewritten in matrix form as

$$\begin{aligned} \mathbf{D}\mathbf{u}^{t+1} &= -(\mathbf{L}+\mathbf{L}^T)\mathbf{u}^t + \mathbf{F} \\ \mathbf{u}^{t+1} &= -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{L}^T)\mathbf{u}^t + \mathbf{D}^{-1}\mathbf{F} \end{aligned} \tag{1.34}$$

where:

$$\begin{aligned} \mathbf{B} &= -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{L}^T) = (\mathbf{I} - \mathbf{D}^{-1}\mathbf{H}) \\ \mathbf{k} &= \mathbf{D}^{-1}\mathbf{F} \end{aligned} \tag{1.35}$$

Then the general update for the basic Jacobi method becomes:

$$u_i^{t+1} = \sum_{j=1}^{q} B_{ij}u_j^t + k_i \qquad \forall i = 1...q \tag{1.36}$$

However, we propose a modification to this by using every new update of u_i^{t+1} immediately after it is computed to give an overall faster increase in the objective.

$$u_i^{t+1} = \sum_{j=1}^{t-1} B_{ij}u_j^{t+1} + \sum_{j=t}^{q} B_{ij}u_j^t + k_i \qquad \forall i \in [1,q] \tag{1.37}$$

3.2 EXTRAPOLATION METHOD

It is possible to further increase the speed of convergence by using an extrapolation technique, which is convergent whenever the basic method (1.30) is symmetrizable.

Definition 1 : The basic method is symmetrizable if for some non-singular matrix A**, the matrix** $\mathbf{A}(\mathbf{I}-\mathbf{B})\mathbf{A}^{-1}$ **is symmetric and positive definite. Otherwise, the method is non-symmetrizable.**

Preposition 2: The formulation of the SVM problem using the basic point Jacobi Method is symmetrizable if and only if the kernel matrix, **G** is symmetric and positive definite.

Proof:

Let $\mathbf{A}^T\mathbf{A} = \mathbf{D}$ and substituting for (1.35),

$$= \mathbf{A}\left(\mathbf{I}-\mathbf{B}\right)\mathbf{A}^{-1}$$
$$= \mathbf{A}(\mathbf{I}-(\mathbf{I}-\mathbf{D}^{-1}\mathbf{H}))\mathbf{A}^{-1}$$
$$= \mathbf{A}((\mathbf{A}^{T}\mathbf{A})^{-1}\mathbf{H})\mathbf{A}^{-1}$$
$$= \mathbf{A}^{-1}\mathbf{H}\mathbf{A}^{-1}$$

Since **H** is the kernel matrix, for our point Jacobi form then $\mathbf{A}^{-1}\mathbf{H}\ \mathbf{A}^{-1}$ is symmetric and positive definite (SPD) if and only if **H** is symmetric and positive definite and by **definition 1**, the method is symmetrizable. □

We will also state the following theorem without proof **[3]**,

*Theorem 1. If the basic method is symmetrizable then, a) the largest eigenvalue of **B**, $\varpi(\mathbf{B})$ is less than unity, b) the eigenvalues of **B** are real, c) the set of eigenvectors of **B** forms a basis of **B** .*

The extrapolation method is then defined by;

$$\mathbf{u}^{t+1} = \gamma\ (\ \mathbf{B}\mathbf{u}^{t} + \mathbf{k}\ ) + (\ 1-\gamma\ )\mathbf{u}^{t} \tag{1.38}$$

For a single iterate this can be expanded and written as,

$$u_i^{t+1} = \gamma\left(\sum_{j=1}^{t-1} B_{ij}u_j^{t+1} + \sum_{j=t}^{q} B_{ij}u_j^{t} + k_i - \mathrm{u}_i^{t}\right) + \mathrm{u}_i^{t} \qquad \forall i = 1\ldots q \tag{1.39}$$

The factor γ is referred to as the *extrapolation factor* and the optimum value is given by;

$$\gamma^{*} = \frac{2}{(2-\lambda_{\max}(\mathbf{B}) - \lambda_{\min}(\mathbf{B})\ )} \tag{1.40}$$

where $\lambda_{\max}(\mathbf{B})$ = largest eigenvalue of **B**
$\lambda_{\min}(\mathbf{B})$ = smallest eigenvalue of **B**

The eigenvalues of **B** are usually unknown prior, however we propose to make use of estimated eigenvalues instead. This would not give the optimal performance of the method but an estimated optimal performance. Then we have as before,

$$\gamma_{E} = \frac{2}{(2-\lambda'_{\max}(\mathbf{B}) - \lambda'_{\min}(\mathbf{B})\ )} \tag{1.41}$$

where $\lambda'_{\max}(\mathbf{B})$ = estimated largest eigenvalue of **B**
$\lambda'_{\min}(\mathbf{B})$ = estimated smallest eigenvalue of **B**

It is easy to see from (1.41), that γ_E is constrained to lie in the interval $1< \gamma_E <2$ due to Theorem 1.

3.3 CONSISTENCY AND CONVERGENCE

The iterative method is said to be *completely consistent*, if the solution, **u*** of (1.37) is also the solution for (1.29). This implies that for a sequence of iterates, $\{u^{(t)}\}=u^*$ for some t and $u^{t+1}=u^{t+2}=\ldots u^*$ or the final solution u* is static and does not change further.

The method is convergent, if the sequence of iterates $u^{(1)}$, $u^{(2)}$ .. converges to u^*. A necessary and sufficient condition of convergence **[3]** is that

$$\mathbf{S}(\mathbf{B})< 1 \tag{1.42}$$

$\mathbf{S}(B)$ is the spectral radius of the qxq matrix **B** and is defined as,

$$\mathbf{S}(B) = \max_{1\le i\le q} |\lambda_i| \tag{1.43}$$

Then from Preposition 2 and Theorem 1, the basic method applied to the SVM problem and the extrapolation method is convergent. Preposition 2 states that the basic method for our SVM problem is symmetrizable and Theorem 1 states that if the method is symmetrizable, the largest eigenvalue of **B** is less than unity and (1.42) is satisfied.

3.4 TRAINING LARGE DATASETS

For large datasets, it would be impossible to store all training vectors in fast memory, let alone cache the whole Gram matrix into memory. For example, a training set consisting of 5000 vectors with each vector a 4-byte integer would require 5000x5000x4 or 100 MB of fast memory. The calculations would have to be done ad hoc for each point. We note that the computation of our update requires the inverse of the diagonal of **H**, and we would have to compute the entire inverse manually. We first review our iterative method and write the updates in term of the respective matrix elements.

Consider the matrices from the basic update rule,

$$\begin{aligned}\mathbf{B} &= \mathbf{(I - D^{-1}H)} \\ \mathbf{k} &= \mathbf{D^{-1}Y}\end{aligned} \tag{1.44}$$

In matrix form, we get

$$\mathbf{B} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} - \begin{pmatrix} H_{11} & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & H_{qq} \end{pmatrix}^{-1} \begin{pmatrix} H_{11} & \ldots & H_{1q} \\ \vdots & \ddots & \vdots \\ H_{q1} & \cdots & H_{qq} \end{pmatrix}$$

$$= \begin{pmatrix} 1 - D_{11}^{-1}H_{11} & -D_{11}^{-1}H_{12} & \cdots & -D_{11}^{-1}H_{1q} \\ -D_{22}^{-1}H_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ -D_{qq}^{-1}H_{q1} & \cdots & \cdots & 1 - D_{qq}^{-1}H_{qq} \end{pmatrix} \tag{1.45}$$

We can now write our general update for u_i^{t+1} in terms of **D** and **H**,

$$u_i^{t+1} = (1 - D_{ii}^{-1}H_{ii})u_i^t - \sum_{j=1}^{i-1} D_{ii}^{-1}H_{ij}u_i^{t+1} - \sum_{j=i+1}^{q} D_{ii}^{-1}H_{ij}u_i^t + k_i \tag{1.46}$$

From (1.24) and (1.29), **H** is the augmented kernel matrix, K_A

$$\mathbf{H} = \left\{ \mathbf{K}(x_i, x_j) + \beta^2 \right\}_{i,j=1}^{n} \tag{1.47}$$

Then it is easy to see that the elements of the diagonal of H can simply be written as,

$$D_{ii} = \left\{ \mathbf{K}(x_i, x_i) + \beta^2 \right\}_{i=1}^{n} \tag{1.48}$$

Lemma 2*: The inverse of a square* n *x* n *diagonal matrix is also a square diagonal matrix with elements that are reciprocals of the elements of the original diagonal matrix.*

Proof:

Let

$$\mathbf{D} = \begin{pmatrix} D_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_{nn} \end{pmatrix}$$

Let the inverse of **D** be **Z**, namely $\mathbf{Z}=\mathbf{D}^{-1}$

$$\mathbf{Z} = \begin{pmatrix} Z_{11} & \cdots & Z_{1n} \\ \vdots & \ddots & \vdots \\ Z_{n1} & \cdots & Z_{nn} \end{pmatrix}$$

Then by properties of the matrices and inverses we have

$$DZ = I$$

$$\begin{pmatrix} D_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_{nn} \end{pmatrix}\begin{pmatrix} Z_{11} & \cdots & Z_{1n} \\ \vdots & \ddots & \vdots \\ Z_{n1} & \cdots & Z_{nn} \end{pmatrix}=\begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

$$\begin{pmatrix} D_{11}Z_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_{nn}Z_{nn} \end{pmatrix}=\begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

by inspection we note;

$$D_{ii}Z_{ii}=1$$

$$Z_{ii}=\frac{1}{D_{ii}}$$

which is the formula for the diagonal elements of **Z** or the inverse of **D** □.

Then by Lemma 1, we replace (1.37) with

$$u_i^{t+1}=(1-\frac{H_{ii}}{D_{ii}})u_i^t-\sum_{j=1}^{i-1}\frac{H_{ij}}{D_{ii}}u_i^{t+1}-\sum_{j=i+1}^{q}\frac{H_{ij}}{D_{ii}}u_i^t+k_i$$

$$=-\sum_{j=1}^{i-1}\frac{H_{ij}}{D_{ii}}u_i^{t+1}-\sum_{j=i+1}^{q}\frac{H_{ij}}{D_{ii}}u_i^t+k_i \qquad (1.49)$$

In terms of the support vector formulation, this becomes;

$$u_i^{t+1}=-\sum_{j=1}^{i-1}\frac{K(x_i,x_j)+\beta^2}{K(x_i,x_i)+\beta^2}u_i^{t+1}-\sum_{j=i+1}^{q}\frac{K(x_i,x_j)+\beta^2}{K(x_i,x_i)+\beta^2}u_i^t+\frac{Y_i}{K(x_i,x_i)+\beta^2} \qquad (1.50)$$

The general formula for extrapolation is then simply;

$$u_i^{t+1}=-\gamma\left(\sum_{j=1}^{i-1}\frac{K(x_i,x_j)+\beta^2}{K(x_i,x_i)+\beta^2}u_i^{t+1}+\sum_{j=i+1}^{q}\frac{K(x_i,x_j)+\beta^2}{K(x_i,x_i)+\beta^2}u_i^t-\frac{Y_i}{K(x_i,x_i)+\beta^2}\right)+(1-\gamma)u_i^t \qquad (1.51)$$

3.4.1 Special Case when using a Gaussian Kernel

We show that if we use a Gaussian kernel of the following form, we can simplify (1.50)

$$K(\mathbf{x},\mathbf{y})=e^{\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} \qquad (1.52)$$

From (1.48), it can be easily deduced that no matter what the value of σ, the diagonal of **H** is always,

$$\mathbf{D} = \begin{pmatrix} 1+\beta^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1+\beta^2 \end{pmatrix}$$

$$\mathbf{D} = (1+\beta^2)\mathbf{I}$$

$$\mathbf{D}^{-1} = \frac{1}{(1+\beta^2)}\mathbf{I} \tag{1.53}$$

Then substituting into (1.50), the general update becomes,

$$\begin{aligned} u_i^{t+1} &= -\sum_{j=1}^{i-1} \frac{K(x_i,x_j)+\beta^2}{1+\beta^2} u_i^{t+1} - \sum_{j=i+1}^{q} \frac{K(x_i,x_j)+\beta^2}{1+\beta^2} u_i^t + \frac{Y_i}{1+\beta^2} \\ &= -\frac{1}{1+\beta^2}\left(\sum_{j=1}^{i-1} \left(K(x_i,x_j)+\beta^2\right) u_i^{t+1} + \sum_{j=i+1}^{q} \left(K(x_i,x_j)+\beta^2\right) u_i^t - Y_i \right) \end{aligned} \tag{1.54}$$

For extrapolation, the updates become simply

$$u_i^{t+1} = -\frac{\gamma}{1+\beta^2}\left(\sum_{j=1}^{i-1} \left(K(x_i,x_j)+\beta^2\right) u_i^{t+1} + \sum_{j=i+1}^{q} \left(K(x_i,x_j)+\beta^2\right) u_i^t - Y_i \right) + (1-\gamma)u_i^t \tag{1.55}$$

It is easily seen, that in the case of the Gaussian kernel, the number of kernel computations for the updates can be reduced since the diagonal element is a constant value.

4. EXPERIMENTAL METHODOLOGY

In order to verify our proposed model, we implemented the algorithms in Section 3 on Matlab 5.0 and ran it on the UCI adult datasets[10] and a real life image recognition application[9]. The pseudocode is provided at the end of this document. All our experiments are done on a Pentium 4, 1.6Ghz machine with 256MB RAM. We cached the kernel matrix for all experiments.

We first ran a series of experiments to investigate the effect of β^2 against the recomputed value of the bias,*b*. This was followed by a series of experiments designed to investigate the speed of our iterative algorithm and the effectiveness of acceleration methods, in this case the extrapolation method. We finally compare our ASVM model

against the original SVM model for a real life image recognition application. For the original SVM formulation, we utilize the the efficient Sequential Minimal Optimization (SMO)[6] to obtain the solution. This allows us to also compare our proposed iterative method against SMO.

Experiment I: Effect of β^2 on the recomputed bias

We ran our algorithm on the first three UCI datasets and recomputed the threshold,b and the equality constraint as we varied β^2. We used a Gaussian kernel and fixed C=1 with a tolerance $\delta = 0.001$. The trends can be seen in Fig 1. 2 and Fig 1. 3.

Experiment II: Sorting of Support Vectors according to change of magnitude

We applied our iterative update rules while retaining J. Platt's method of sweeping through all data points first and then through only the support vectors until all the conditions are satisfied within a certain tolerance. We note that we can also keep a cache of Support Vectors in memory instead of having to store the entire alpha vector, but this is unnecessary for the purpose of our experiments. We also propose a new metric to cache the Support Vectors in order to increase the speed of convergence.

We note that in J. Platt's algorithm, the sweep through Support Vectors is done randomly. In Mangasarian's work[8], the support vectors are sorted according to magnitude before sweeping through them in descending order. However, we hypothesize that this is still some random form of sweeping through the Support Vectors. Instead, at each pass through the data points or Support Vectors, we compute the magnitude of change, ε_i for each new Support Vector,

$$\varepsilon_i (n+1) = | \alpha_i^{n+1} - \alpha_i^n | \qquad i \in SV \tag{1.56}$$

We then sort the Support Vectors in descending order of the magnitudes of ε_i, so that on the next pass, the Support Vector with the largest ε_i is updated first. We ran our algorithm on the UCI adult dataset 1 which contains 1605 training examples. We use β^2=1 and δ=0.001 for all experiments. The experiments are done to compare between no sorting, sorting according to magnitude of Support Vectors (sorting 1) and sorting according to our proposed method, (sorting 2) and with extrapolation (sorting 2 with exp). The results are presented in Fig 1. 4.

Experiment III : Comparison of basic method and acceleration with extrapolation

We next applied the extrapolation method to the basic iterative method. We investigate the effect of choosing an estimate of the extrapolation factor, γ by varying the range of γ through a series of evenly spaced values. For γ =1, we see from (1.39) that the extrapolation method reduces to the basic iterative method of (1.37). We ran the experiment on the first adult dataset, UCI 1, using β=1 and β=100. The results are presented in the trend for $\delta=10^{-3}$ in **Fig 1. 5** and Fig 1. 6

Experiment IV: Comparison between SMO and ASVM, an application to image recognition

In this experiment, we were interested in comparing the optimization speed of our algorithm against the SMO.We applied both algorithms to a practical image recognition problem that is electronic monitoring of fishways. The complete experimental setup is found in **[9]**. Our dataset consists of 5 different fish species with 4 representatives each. Each representative had 70 images giving a total of 1400 images. We extracted 10 attributes, which gave the best results from each image, such as fish dimensions. For comparison, we ran SMO and our algorithm using $\delta=10^{-3}$. We obtained a flop count and real computation times. In addition, we used 5-fold cross validation in all cases to investigate the generalization capabilities. The kernel matrix was cached for both algorithms since there was enough memory available. We ran experiments on various kernels such as the Gaussian kernel, linear kernel and polynomial kernel. The results can be found in Fig 1. 7-Fig 1. 12

5. DISCUSSION

We first present empirical proof of Lemma 1 by the results of the recomputed threshold while varying the parameter β^2. Fig 1. 2 shows the asymptotic behavior of the threshold *b* when β^2 is varied. Fig 1. 3 indicates the behavior of the KKT condition $\sum_{i=1}^{l} \alpha_i y_i$ with the variation of β^2. It can be seen that the recomputed threshold approaches an asymptotic value as β^2 increases. From Lemma 1, the optimal solution is found at $\beta^2 = \infty$ and larger β^2 gives solutions closer to the original SVM formulation. However, care has to be taken when increasing β^2 with respect to the kernel function, K(**x**,**y**)in our augmented $K_A(\mathbf{x}, \mathbf{y})$. Too large a β^2 causes,

$$K_A(\mathbf{x}, \mathbf{y}) \approx \beta^2$$

This implicitly maps all training points to one point in feature space and makes the problem inseparable.

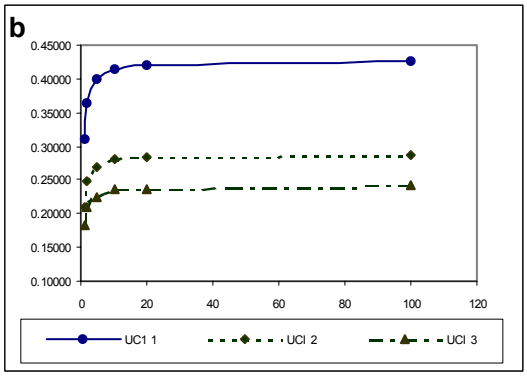

Fig 1. 2 Recomputed threshold b against β^2 for UCI 1-3 datasets

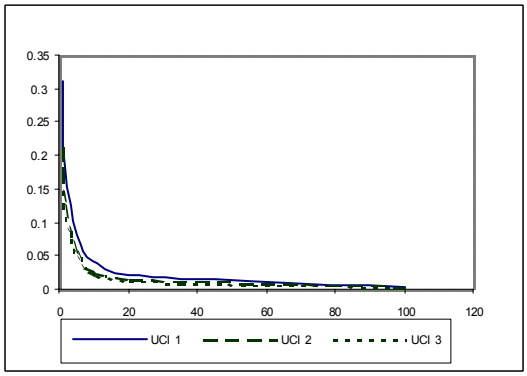

Fig 1. 3 Reomputed equality constraint $\sum \alpha_i y_i$ against β^2 for UCI 1-3 datasets

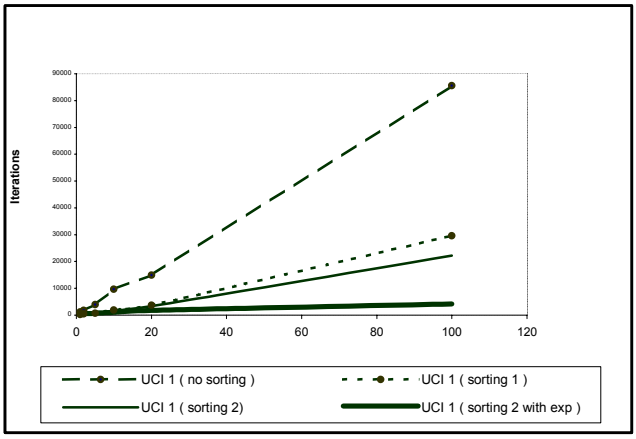

Fig 1. 4 Optimization speed for sorting and no sorting over a range of β^2 for UCI dataset

The advantage of our ASVM formulation is that it lends itself to the application of linear stationary iterative methods, which are easy to implement. Furthermore, several polynomial acceleration techniques exist which can be used to accelerate the rate of convergence further during the training phase. The elimination of the equality constraint causes a major speedup in optimization techniques since all we have to do is enforce the inequality constraint at each optimization step. However the generalization capabilities of this formulation will be a subject of further investigation. It is still unknown whether the ASVM model is an approximate solution to the original SVM model or should be treated as a different family of estimators.

From equation (1.54) and (1.55), the update u_i^{t+1} depends entirely on other values of **u** and can be seen as simply computing the update as an unknown in the decision function to achieve the required output. The advantages are many such as, easy implementation with possible extension to other polynomial acceleration methods to further increase the rate of convergence and no long searches for an update to be made. A more comprehensive study on the impact of sorting the Support Vectors could be undertaken to provide a proper understanding of the factors governing the speed of convergence using these family of iterative methods.

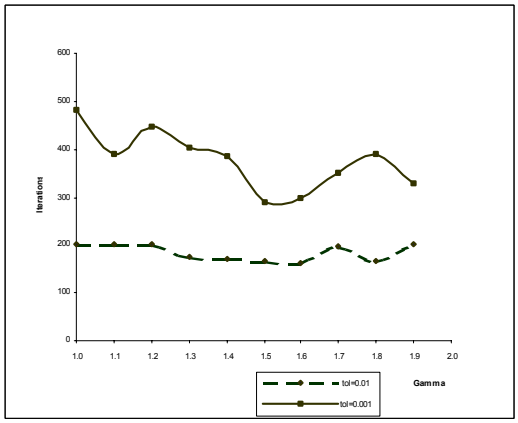

Fig 1. 5 Extrapolation on UCI 1 with β^2=1

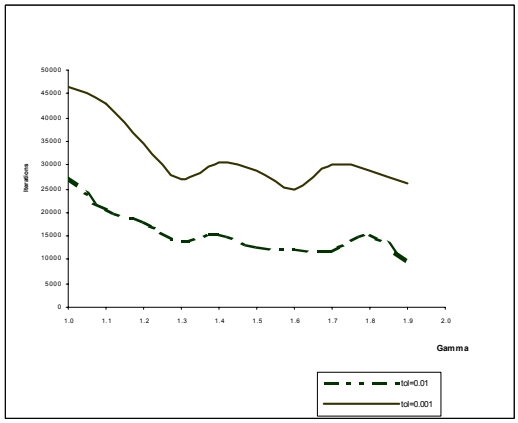

Fig 1. 6 Extrapolation on UCI 1 with β^2=100

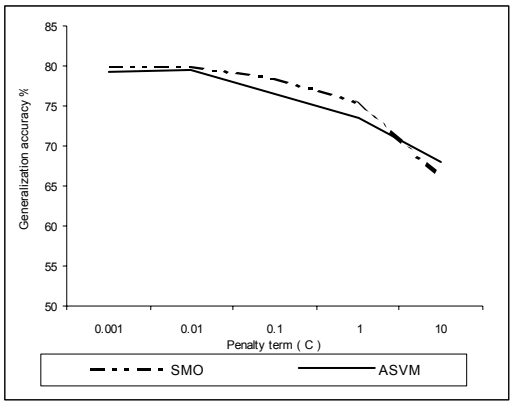

Fig 1. 7 Generalization on a Gaussian kernel with $\sigma = 10$

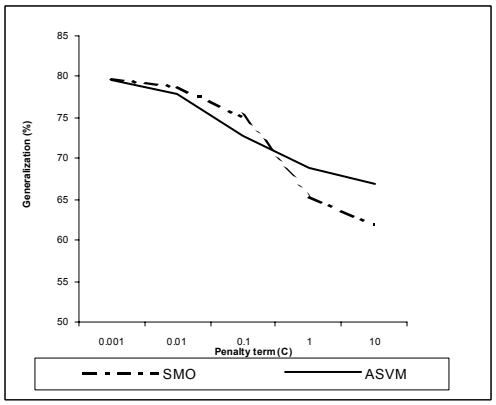

Fig 1. 8 Generalization on a linear kernel

We have demonstrate the validity of our hypothesis when sorting the Support Vectors according to the magnitudes of change. A significant increase in performance is observed when sorting the Support Vectors according to our method then compared to non-sorting. Against Mangasarian's method, we record a 15% increase in optimization speed for the basic iterative method when sorting using our proposed method. We achieved a further increase in performance when the extrapolation method was applied to the basic method.

Fig 1. 5 and Fig 1. 6 demonstrate that extrapolation can be applied for significant increase in the rates of convergence. In this document, we only demonstrate the capabilities of the extrapolation method, and we note that if we are to use them effectively, we should try to obtain good eigenvalues estimates to give optimal performance. From Fig. 4.2, we can see that the optimal value of γ is around 1.55 while in Fig 4.3, γ=1.9 is optimal. The true eigenvalues could be obtained using decomposition methods like Singular Value Decomposition and so on. However, they

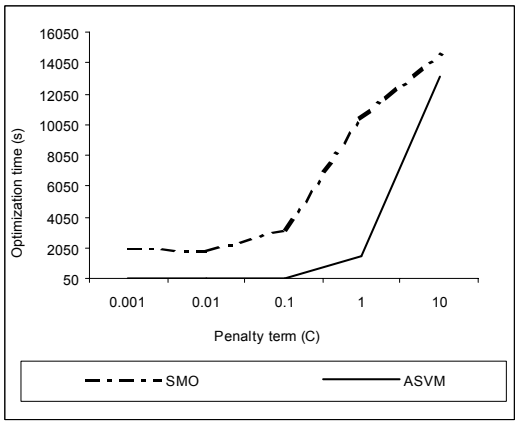

Fig 1. 9 Optimization time(s) on a Gaussian kernel with $\sigma = 10$

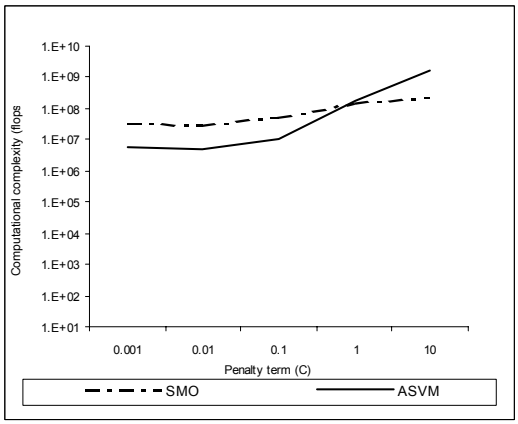

Fig 1. 10 Computational complexity on a Gaussian kernel with $\sigma = 10$

will require additional computational time and complexity and hence it is sufficient to use the estimated values. Overall, the extrapolation method gives us an increase of almost an order of magnitude in performance compared to just the basic method.

Against the SMO algorithm, we discovered that at $\beta^2 = 20$, our algorithm was comparable to SMO in terms of computation intensity. We note however, that with our formulation, we are maximizing a margin in $l+1$ dimensions and in the limit as $\beta^2 \to \infty$, we will maximize a margin in l dimensions as the standard Support Vector Machine. We could increase the speed of our algorithm if we choose a lower value of β^2, but we initially believed that this might compromise in terms of the generalization capability of our Support Vector Machine. However, in our fish experiment, this notion proved to be unfounded by empirical study as seen from Fig 1. 7 and Fig 1. 8. The generalization of our model was comparable to that achieved by SMO in the original classification formulation even when using a low value of $\beta^2 = 1$. For higher values of C, our ASVM model performed better than the original classification model instead. We attribute this to the distribution of the data but as of now; we have yet to prove concretely the observed phenomena.

We also show from Fig 1. 9-Fig 1. 12 that our algorithm is capable of achieving a much faster optimization compared to SMO. For small values of C, we found that our algorithm was more efficient than SMO, requiring lesser floating point operations (flops). However, we observed that the method became less efficient at higher values of C. This is of little concern since at higher values of C, the trained SVM's generalization capabilities (Fig 1. 7-Fig 1. 8) become unacceptable and we would not be using those values. Thus in the acceptable region, we have demonstrated that our algorithm is much faster and more computationally efficient than the SMO.

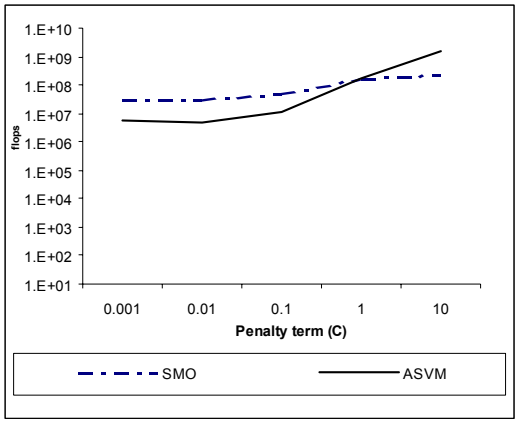

Fig 1. 11 Computational complexity on a linear kernel

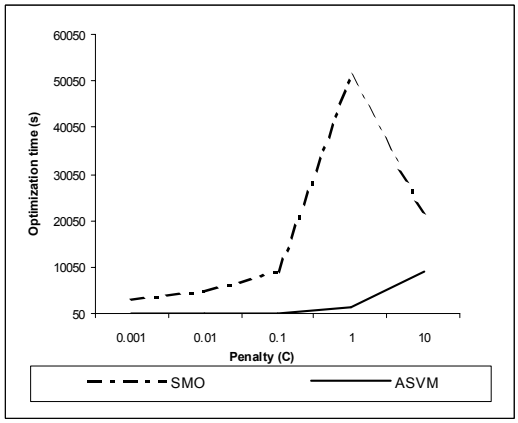

Fig 1. 12 Optimization time(s) on a linear kernel

6. CONCLUSION

We have presented a new formulation namely the Automatically biased Support Vector Machine classifier, which removes the equality constraint from the optimization problem. In this simple modification, we have implicitly allowed for the automatic calculation of the bias, though we stress that in our formulation, the bias *b* no longer has any meaning. Instead we have shown that our model can be controlled by the parameter β^2. We have presented empirical results to investigate the effect of our introduced parameter, β^2. We further show that this formulation allows for the application of a modified iterative method. We showed that the basic method could be accelerated with extrapolation and note that this method can be further extended to other polynomial acceleration methods. Future research will concentrate on extending this formulation to the regression and density estimation Support Vector formulations.

Pseudocode for Matlab

function TrainASVM ()

```
Initialize alpha vector to zero;
Initialize Error Cache to zero;
Initialize f(x) Cache to zero;
Choose C, tol, eps, gamma, beta;
NumChanged=0;
ExamineAll=1;

while ( NumChanged>0 | | ExamineAll==1)
  %sweep thru whole data set and update the KKT violaters
   if ( examineAll==1)
        for all i1 in n
          numChanged=numChanged+a1pha_up(i1);
        end
        up_cache;  ← sort values in Error Cache in decending order
   else
   %sweep thru non bound alphas or Support Vectors in the Error Cache
        for all i1 in Error Cache
          numChanged=numChanged+alpha_up(i1);
        end
        up_cache; ← sort values in Error Cache in decending order
  end
    if  ( examineAll==1)  examineAll=0;
  else
     if (numChanged==0)  examineAll=1;
  end
end  %end while
```

```
function alpha_up(i1)

  y1 =  Label of X(i1);
  E1 = f(i1) – y1 ( if i1!=SV compute E1 else obtain f(i1) from Error Cache);
  R1 = E1*y1;

if ( (R1<-tol & Alpha(i1)<C) | ( r1>tol & Alpha(i1)>0 ) )

 %update alpha rule
 Bu=fx(i1)-(beta+1)*Alpha(i1);
 U_new=-gamma/(beta+1)*(Bu-y1)+(1-gamma)*Alpha(i1);

 Alpha_new=U_new*y1;

 %clip alpha
  if ( Alpha_new>C )  then Alpha_new=C;
  else
    if (Alpha_new<0) then Alpha_new=0;
  end

 U_new=Alpha_new*y1;

 %admit specific tolerance
 if( abs(U_new-Alpha(id))<eps )
    return 0;
 end

%Add to cache if support vector

if( Alpha_new >0 & Alpha_new < C )
  new_change= abs(Alpha_new-Alpha(id)*Train(id,Cols));
 Cache_Temp( x , 1 ) = i1 ;
 Cache_Temp( x,  2 ) =new_change;
end

update f(x) for support vectors
Alpha(i1)=U_new; %store new value of alpha
   return 1;
else
    return 0;
end
```

REFERENCES

[1] N. Cristianini and J. Shawe-Taylor , *An Introduction to Support Vector Machines (and other kernel-based learning methods)*, Cambridge University Press 2000 ISBN: 0 521 78019 5

[2] C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2): 11-167, 1998.

[3] Louis A. Hageman , David M. Young , Applied Iterative Methods , Academic Press 1981

[4] Philip E. Gill , Walter Murray and Margaret H. Wright, Practical Optimization , Academic Press 1981

[5] Vladimir N. Vapnik , The nature of statistical learning, Springer-Verlag New York, 1995.

[6] J. Platt, *Fast Training of Support Vector Machines using Sequential Minimal Optimization*, in Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. Burges, and A. Smola, eds., MIT Press, (1998).

[7] Shevade, S.K.; Keerthi, S.S.; Bhattacharyya, C.; Murthy, K.R.K. , Improvements to the SMO algorithm for SVM regression, Neural Networks, IEEE Transactions on , Volume: 11 Issue: 5 , Sept. 2000

[8] Mangasarian, O.L.; Musicant, D.R., Successive overrelaxation for support vector machines Neural Networks, IEEE Transactions on , Volume: 10 Issue: 5 , Sept. 1999

[9] B. Owen, M. Palaniswami and J. Harris, Automated Monitoring of Fishways, in Proc. of 3rd IEEE International Conference on Intelligent Processing Systems, ICIPS, pp. 238-242, Gold Coast, Australia, August 1998.

[10] Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases Irvine, CA: University of California, Department of Information and Computer Science. [http://www.ics.uci.edu/~mlearn/MLRepository.html].