Sequential support vector classifiers and regression

Proc. International Conference on Soft Computing (SOCO'99), Genoa, Italy, pp.610-619 (1999).

Sequential Support Vector Classifiers and Regression

Sethu Vijayakumar and Si Wu
RIKEN Brain Science Institute, The Institute for Physical and Chemical Research,
Hirosawa 2-1, Wako-shi, Saitama, Japan 351-0198
{sethu,phwusi}@brain.riken.go.jp

1 Abstract

Support Vector Machines(SVMs) map the input training data into a high dimensional feature space and finds a maximal margin hyperplane separating the data in that feature space. Extensions of this approach account for non-separable or noisy training data (soft classifiers) as well as support vector based regression. The optimal hyperplane is usually found by solving a quadratic programming problem which is usually quite complex, time consuming and prone to numerical instabilities. In this work, we introduce a sequential gradient ascent based algorithm for fast and simple implementation of the SVM for classification with soft classifiers. The fundamental idea is similar to applying the Adatron algorithm to SVM as developed independently in the Kernel-Adatron [7], although the details are different in many respects. We modify the formulation of the bias and consider a modified dual optimization problem. This formulation has made it possible to extend the framework for solving the SVM regression in an online setting. This paper looks at theoretical justifications of the algorithm, which is shown to converge robustly to the optimal solution very fast in terms of number of iterations, is orders of magnitude faster than conventional SVM solutions and is extremely simple to implement even for large sized problems. Experimental evaluations on benchmark classification problems of sonar data and USPS and MNIST databases substantiate the speed and robustness of the learning procedure.

2 Introduction

Support Vector Machines(SVMs) were introduced for solving pattern recognition problems by Vapnik et al.[6, 4] as a technique that minimized the structural risk - that is, the probability of misclassifying novel patterns for a fixed but unknown probability distribution of the input data - as opposed to traditional methods which minimize the empirical risk[15]. This induction principle is equivalent to minimizing an upper bound on the generalization error. What makes the SVMs so attractive is the ability to condense the information in the training data and provide a sparse representation using non-linear decision surfaces of relatively low VC dimension. Moreover, when extended for the regression case, the SVM solution provides a tool for some level of automatic model selection- for example, the SVM regression with RBF kernels leads to automatic selection of the optimal number of RBFs and their centers [13].

The SVM solution relies on maximizing the margin between the separating hyperplanes and the data. This solution is achieved by reducing the problem to a quadratic programming problem and using optimization routines from standard numerical packages. In spite of the successes of SVMs in real world applications like handwritten character recognition problems [3], it's uptake in practice has been limited because quadratic optimization through packages are computationally intensive, suspect to stability problems and it's implementation, especially for a large training data set, is non-trivial. Recently, some algorithms have dealt with chunking the data sets and solving smaller quadratic problems to deal with memory storage problems resulting from large training sets[10]. However, it still uses the quadratic programming solvers for the basic optimization. The idea of using a non-linear kernel on top of the Adatron algorithm to obtain the Kernel-Adatron algorithm for solving the support vector classification problem has been proposed in [7] in parallel to our work. However, in the initial implementations[7], they do not consider bias nor soft margin classifiers. In later modifications [8], the authors talk of an implementation of bias and soft margin which is conceptually different from our method. Based on the same idea of applying the Adatron method to SVM, in this work, by exploiting an alternate formulation for the bias term in the classifier, we derive a sequential gradient ascent type learning algorithm which can maximize the margins for the quadratic programming problem in high-dimensional feature space and provide very fast convergence in terms of number of iterations to the optimal solution within particular analytically derived bounds of the learning rate. We look at theoretical implications of the modification of the cost function and derive conditions that are necessary to obtain optimal class sep-

aration. Experimental results on benchmark datasets have shown that the algorithm provides generalization abilities equal to the standard SVM implementations while being orders of magnitude faster. The algorithm is capable of handling noisy data and outliers through implementation of the soft classifier with adaptable trade-off parameter. Furthermore, the algorithm has been extended to provide a fast implementation of the Support Vector regression, a unified approach to classification and regression which very few other algorithms have addressed. It should be noted in relation to the work in Friess et al. [7] that although the the fundamental idea is the same and final algorithm looks quite similar, there are conceptual differences in the way the bias and soft margin is handled. A difference in the update procedures (clipping) can be shown to lead to an extended convergence plateau in [7] resulting in qualitatively similar results but affecting the speed of solution and the number of resulting SVMs.

Section 3 reviews the theoretical framework of the SVMs and looks at implementation of non-linear decision surfaces. Section 4 provides a new formulation for the bias and describes the sequential algorithm for classification along with rigorous justifications for it's convergence. We also perform a theoretically rigorous analysis on the margin maximization which clarifies the additional conditions that need to be satisfied to obtain a balanced classifier. Section 5 compares the learning performance of the proposed method against other techniques for standard benchmark problems. Section 6 describes an extension of the algorithm for implementing SVM regression while Section 7 summarizes the salient features of the proposed method and looks at future research directions.

3 The classification problem and Support Vectors

We start by introducing the theoretical underpinnings of the Support Vector learning. Consider the problem of classifying data $\{\boldsymbol{x}_i, y_i\}_{i=1}^{l}$, where $\boldsymbol{x}_i$ are the N dimensional pattern vectors and $y_i \in \{-1, 1\}$ are the target values.

3.1 The linearly separable case

We first consider the case when the data is linearly separable. The classification problem can be reformulated as one of finding a hyperplane $f(\boldsymbol{w}, b) = \boldsymbol{x}_i \cdot \boldsymbol{w} + b$ which separate the positive and negative examples:

$$\begin{aligned} \boldsymbol{x}_i \cdot \boldsymbol{w} + b &\geq 1 \text{ for } y_i = 1, & (1)\\ \boldsymbol{x}_i \cdot \boldsymbol{w} + b &\leq 1 \text{ for } y_i = -1. & (2) \end{aligned}$$

which is equivalent to

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 \geq 0 \text{ for } i = 1, \ldots, l. \quad (3)$$

where $\boldsymbol{a} \cdot \boldsymbol{b} \equiv \sum_i a_i b_i$ represents the dot product. All points for which eq.(1) hold lie on the hyperplane $H_1 : \boldsymbol{x}_i \cdot \boldsymbol{w} + b = 1$ with normal $\boldsymbol{w}$ and perpendicular distance from origin $d_{H_1} = \frac{(1-b)}{\|\boldsymbol{w}\|}$. Similarly, those points satisfying eq.(2) lie on the hyperplane $H_2 : \boldsymbol{x}_i \cdot \boldsymbol{w} + b = -1$ with distance from the origin $d_{H_2} = \frac{(-1-b)}{\|\boldsymbol{w}\|}$. The distance between the two canonical hyperplanes, H_1 and H_2 is a quantity we refer to as the *margin* and is given by :

$$\text{Margin} = |d_{H_1} - d_{H_2}| = \frac{2}{\|\boldsymbol{w}\|}. \quad (4)$$

There are (in general) many hyperplanes satisfying the condition given in eq.(3) for a data set that is separable. To choose one with the best generalization capabilities, we try to maximize the margin between the canonical hyperplanes [5]. This is justified from the computational theory perspective since maximizing margins correspond to minimizing the V-C dimension of the resulting classifier [15] and from the interpretation of attractor type networks in physics, achieves maximal stability[2].

Therefore, the optimization problem to be solved becomes

$$\begin{aligned} \text{Minimize} \quad & J_1[\boldsymbol{w}] = \frac{1}{2}\|\boldsymbol{w}\|^2 & (5)\\ \text{subject to} \quad & y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 \geq 0, i = 1, \ldots, l & (6) \end{aligned}$$

3.2 The linearly non separable case

In case the data are not linearly separable, as is true in many cases, we introduce slack variables $\xi_i \geq 0$ to take care of misclassifications such that the following constraint is satisfied :

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i \geq 0 \text{ for } i = 1, \ldots, l. \quad (7)$$

Here, we choose a tradeoff between maximizing margins and reducing the number of misclassifications by using a user defined parameter C such that the optimization problem looks like :

$$\begin{aligned} \text{Minimize} \quad & J_2[\boldsymbol{w}, \xi_i] = (\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\xi_i) & (8)\\ \text{subject to} \quad & y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i \geq 0, & (9)\\ & \xi_i \geq 0,\ i = 1, \ldots, l. & (10) \end{aligned}$$

We can write this constrained optimization as an unconstrained quadratic convex programming problem by introducing Lagrangian multipliers as

$$\text{Minimize} \quad L(\boldsymbol{w}, b, \xi_i, \boldsymbol{h}, \boldsymbol{r}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\xi_i$$

$$-\sum_{i=1}^{l} h_i[y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i] - \sum_{i=1}^{l} r_i \xi_i \quad (11)$$

where $\boldsymbol{h} = (h_1, \ldots, h_l)$ and $\boldsymbol{r} = (r_1, \ldots, r_l)$ are the non-negative Lagrange multipliers.

3.3 The dual problem and sparsity of solution

The optimization problem of eq.(11) can be converted into a dual problem which is much more easier to handle[5] by looking at the saddle points of eq.(11).

$$\text{Maximize} \qquad L_D(\boldsymbol{h}) = (\sum_{i=1}^{l} h_i - \frac{1}{2}\boldsymbol{h} \cdot \mathbf{D}\boldsymbol{h}) \quad (12)$$

$$\text{subject to} \qquad \sum_{i=1}^{l} y_i h_i = 0, \quad (13)$$

$$0 \leq h_i \leq C, \quad i = 1, \ldots, l. \quad (14)$$

where $D_{ij} = y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j$.

Solutions to this dual problem is traditionally found by using standard quadratic programming packages. Once the optimal multipliers h_i are found, the classifier is simply

$$f(\boldsymbol{x}) = sign(\sum_i h_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x} + b_0). \quad (15)$$

Usually, based on the Kuhn-Tucker optimality condition, the number of non-zero Lagrange multipliers h_i are much smaller than the number of data and hence, the kind of expansion for the classifier given by eq.(15) is very *sparse*. The data points corresponding to these non-zero Lagrangians are referred to as *support vectors*. The bias term b_0 can be found using any support vector $\boldsymbol{x}_{sv}$ as

$$b_0 = y_{sv} - \sum_{i \in \mathrm{SV}} h_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x}_{sv}. \quad (16)$$

3.4 Extension to non-linear decision surfaces

Obviously, the power of linear decision surfaces are very limited. SVMs provide a convenient way of extending the analysis from the input space to a non-linear feature space by using a high-dimensional mapping $\Phi(\boldsymbol{x})$. Finding a linear separating hyperplane in this feature space is equivalent to finding a non-linear decision boundary in the input space.

It should be noted that the input vectors appear only in the form of dot products $\Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)$ in the dual problem and it's solution. By using the reproducing kernels of a Hilbert space, we can rewrite this dot product of two feature vectors in terms of the kernel, provided some (non-trivial) conditions are satisfied :

$$(\Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)) = K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \sum_k \phi_k(\boldsymbol{x}_i)\phi_k(\boldsymbol{x}_j). \quad (17)$$

Some common examples of the kernels are the Gaussian RBF kernels, given as

$$K(\boldsymbol{x}, \boldsymbol{x}') \equiv e^{-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/2\sigma^2} \quad (18)$$

and polynomial kernel of degree d, written as

$$K(\boldsymbol{x}, \boldsymbol{x}') \equiv (\boldsymbol{x} \cdot \boldsymbol{x}' + 1)^d. \quad (19)$$

The use of kernels instead of dot products in the optimization provides an automatic implementation of representing hyperplanes in the feature space rather than the input space. For further details on SVMs and use of kernels, the reader is referred to the tutorial by Burges[5].

4 Sequential learning of support vectors for classification

The main aim of this work is to implement the support vectors in a fast, simple, sequential scheme. To this effect, we make a slight modification in the problem formulation.

4.1 An alternate formulation for bias

In the previous sections, the bias has been represented explicitly using an additional variable b. From now on, we will use an additional dimension for the pattern vectors such that $\boldsymbol{x}' = (x_1, \ldots, x_N, \lambda)$ and incorporate the bias term in the weight vector, $\boldsymbol{w}' = (w_1, \ldots, w_N, b/\lambda)$, where λ is a scalar constant. The choice of the magnitude of the augmenting factor λ is discussed in Section 4.1.1.

Based on this modification, we write an analogous optimization problem.

$$\text{Mininimize} \quad L'(\boldsymbol{w}', \xi_i) = (\frac{1}{2}\|\boldsymbol{w}'\|^2 + C\sum_{i=1}^{l} \xi_i) \quad (20)$$

$$\text{subject to} \quad y_i(\boldsymbol{x}_i' \cdot \boldsymbol{w}') - 1 + \xi_i \geq 0, \quad (21)$$

$$\xi_i \geq 0, \quad i = 1, \ldots, l. \quad (22)$$

If we look carefully, the margin we are trying to maximize is slightly different from the original problem. Instead of maximizing $\frac{2}{\|\boldsymbol{w}\|}$ or minimizing $\|\boldsymbol{w}\|^2$, we are minimizing $\|\boldsymbol{w}'\|^2 = \|\boldsymbol{w}\|^2 + b^2/\lambda^2$. We now look at the implications of this approximation.

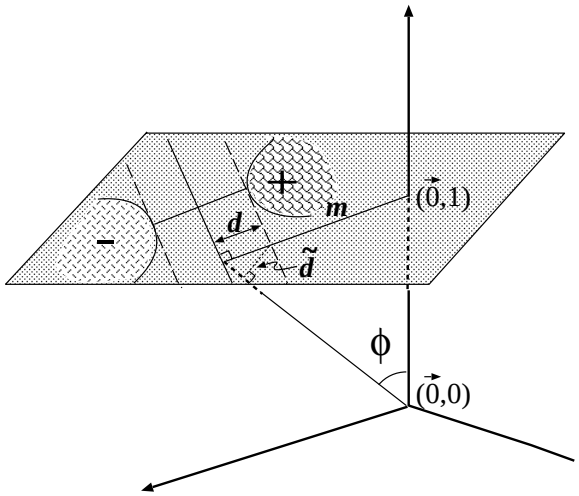


Figure 1: Visualization of the modified margin in higher dimensional space

Table 1: Effect of augmenting factor on approximation quality

Augmenting Factor(λ)	θ(degrees)
0.1	31.823032
0.5	16.999935
1.0	6.831879
2.0	0.977416
5.0	0.275545

4.1.1 Effect of modified bias formulation on margin maximization

Here, we look at the implications of the maximizing a modified margin (refer Section 4.1) which involves using an input vector augmented with a scalar constant λ. For the sake of analysis, let us assume that the augmenting factor $\lambda = 1$. Fig. 1 gives a schematic diagram of a linearly separable classification problem in a higher dimensional space after the input vector has been augmented. The margin being maximized in the original problem with explicit bias, based on eq.(4), is

$$d^2 = 1/\|\boldsymbol{w}\|^2. \tag{23}$$

The modified margin $\tilde{d}$ is the closest distance from the data points to a hyperplane in the higher dimensional space passing through the origin. Based on simple geometry, it can be seen from Fig. 1 that

$$\tilde{d}^2 = d^2 \cdot cos^2\phi. \tag{24}$$

The angle ϕ satisfies the following relationship:

$$cos^2\phi = 1/(1+m^2) \quad \text{where } m = b/\|\boldsymbol{w}\| \tag{25}$$

Therefore, eqs.(23), (24) and (25) yield an explicit relationship for the modified margin:

$$\tilde{d}^2 = 1/(\|\boldsymbol{w}\|^2 + b^2). \tag{26}$$

So, maximizing the modified margin will, in general, only give an approximate solution of the original optimization problem. In case the input dimension is high, the contribution of the bias term in the margin $1/(\|\boldsymbol{w}\|^2+b^2)$ is negligible and the approximation will be very good as confirmed in our simulation experiments.

From eq.(24), we see that if the angle ϕ (refer Fig. 1) is smaller, the approximation tends to be better. This can be ensured by augmenting the input vector by a 'larger' number instead of '1'. To prove this point and also show how good the approximation is, we carried out simulations on an artificial two dimensional linearly separable data set. One data cluster is randomly generated in the range $[-1,1] \times [-1,1]$ with uniform distribution. The other one is obtained in the same way in the range $[1.5,3.5] \times [1.5,3.5]$. It is easy to see that the normal of the optimal classifying hyperplane $\boldsymbol{w}^*$ can be obtained in the following way: Select a pair of data points, say $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, from different clusters such that the distance between them is the minimum among all such pairs. $\boldsymbol{w}^*$ can then be expressed as $\boldsymbol{w}^* = (\boldsymbol{x}_2 - \boldsymbol{x}_1)/\|\boldsymbol{x}_2 - \boldsymbol{x}_1\|$. Let the solution achieved through SVMseq be $\boldsymbol{w}$. The angle θ between $\boldsymbol{w}$ and $\boldsymbol{w}^*$, which indicates how good the approximation is, can be written as

$$\theta = \arccos \frac{\boldsymbol{w} \cdot \boldsymbol{w}^*}{\|\boldsymbol{w}\|}. \tag{27}$$

Table 1 compares the effect of the augmenting factor (λ) on the approximation quality. We see that the approximation improved vastly with the increase of the magnitude of the augmenting factor. The simulations used a fixed set of 50 data points for the experiments. In practice, however, there is a tradeoff resulting from increasing the augmenting factor. Large augmenting factor will normally lead to slower convergence speeds and instability in the learning process. An appropriate solution is to choose an augmenting factor having the same magnitude as the input vector. However, while using the non-linear kernels, the effect of the augmenting factor is not directly interpretable as above and depends on the exact kernels we use. We have noticed that setting the augmenting term to zero (equivalent to neglecting the bias term) in high dimensional kernels gives satisfactory results on real world data.

4.1.2 Obtaining a balanced classifier

In the case when the data is linearly separable, a balanced classifier is defined as the hyperplane ($f(\boldsymbol{w}, b) =$

$\boldsymbol{x}\cdot\boldsymbol{w}+b=0$) that lies in the middle of the two clusters, or more precisely, the bias term b is given as:

$$b = -\frac{1}{2}(\boldsymbol{x}_i^+ \cdot \boldsymbol{w} + \boldsymbol{x}_i^- \cdot \boldsymbol{w}), \tag{28}$$

where $\boldsymbol{x}_i^+$ and $\boldsymbol{x}_i^-$ are any two support vectors from the two clusters, respectively. Certainly, a balanced classifier is superior to an unbalanced one. For the original problem, a balanced classifier is automatically obtained. However, while maximizing the modified margin, it is no longer the case. A constraint on the magnitude of $\boldsymbol{w}$ needs to be satisfied.

Suppose there is an unbalanced classifier $f(\boldsymbol{w}', b')$. Without loss of generality, we assume it is closer to the negative cluster, that is,

$$\begin{aligned} \boldsymbol{x}_i^+ \cdot \boldsymbol{w}' + b' &= c, \quad c > 1 \\ \boldsymbol{x}_i^- \cdot \boldsymbol{w}' + b' &= -1. \end{aligned}$$

It is easy to check by using the transformation $\boldsymbol{w}^* = \frac{2}{1+c}\boldsymbol{w}'$ and $b^* = \frac{2}{1+c}b' + \frac{1-c}{1+c}$ that we can get a balanced classifier $f(\boldsymbol{w}^*, b^*)$ parallel to the unbalanced one :

$$\begin{aligned} \boldsymbol{x}_i^+ \cdot \boldsymbol{w}^* + b^* &= 1, \\ \boldsymbol{x}_i^- \cdot \boldsymbol{w}^* + b^* &= -1. \end{aligned}$$

So, we can construct a family of parallel classifiers parameterized by c, for $c \geq 1$ in the following way:

$$\boldsymbol{w}(c) = \frac{1+c}{2}\boldsymbol{w}^*, \tag{29}$$

$$b(c) = \frac{1+c}{2}b^* - \frac{1-c}{2}, \tag{30}$$

such that

$$\boldsymbol{x}_i^+ \cdot \boldsymbol{w}(c) + b(c) = c, \quad c > 1 \tag{31}$$

$$\boldsymbol{x}_i^- \cdot \boldsymbol{w}(c) + b(c) = -1. \tag{32}$$

Here, $f(\boldsymbol{w}(1), b(1))$ gives the balanced classifier, which corresponds to $c = 1$ and $f(\boldsymbol{w}^*, b^*)$. For $c > 1$, an unbalanced classifier closer to the negative cluster is obtained. We will, for the sake of simplicity, assume the augmenting factor to be 1. Then, SVMseq minimizes $\|\boldsymbol{w}\|^2 + b^2$. So, when

$$L(c) = \|\boldsymbol{w}(c)\|^2 + b(c)^2 \tag{33}$$

takes the minimum value at $c = 1$, SVMseq will provide a balanced classifier. Calculating the derivative of $L(c)$ with respect to c, we have

$$\frac{dL(c)}{dc} = \frac{1+c}{2}[\|\boldsymbol{w}^*\|^2 - \frac{1}{(1+c)^2}] + \frac{1+c}{2}[b^* + \frac{c}{1+c}]^2. \tag{34}$$

Note that since $c \geq 1$, $\|\boldsymbol{w}^*\|^2 > \frac{1}{4}$ ensures $\frac{dL(c)}{dc} > 0$, for any c. So, $\|\boldsymbol{w}^*\|^2 > \frac{1}{4}$ is the sufficient condition that a balanced classifier will be obtained in our method. In practice, this is easy to satisfy by just scaling the input data to smaller values, an automatic result of which leads to larger magnitudes of $\boldsymbol{w}$.

Working along the lines of Section 3.3, and solving for the hyperplane in the high dimensional feature space, we can convert the modified optimization problem of eq.(20) into a dual problem:

$$\text{Maximize} \qquad L'_D(\boldsymbol{h}) = (\sum_i h_i - \frac{1}{2}\boldsymbol{h} \cdot \mathbf{D}'\boldsymbol{h}) \tag{35}$$

$$\text{subject to} \qquad 0 \leq h_i \leq C, \quad i = 1, \ldots, l. \tag{36}$$

$$\tag{37}$$

where $D'_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \lambda^2 y_i y_j$. Here, in the non-linear case, we augment the input vector with an extra dimension in the feature space, i.e., $\Phi(\boldsymbol{x}') = (\Phi(\boldsymbol{x})\ \lambda)^T$. The resultant non-linear classifier is computed as

$$f(\boldsymbol{x}) = sign(\sum_{i \in \mathrm{SV}} h_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + h_i y_i \lambda^2). \tag{38}$$

For the sake of clarity, we will stop using the 'dash' notation and from now on assume that vectors $\boldsymbol{x}$ and matrix $\mathbf{D}$ refers to the modified input vector augmented with the scalar λ and modified matrix, respectively, as described in the beginning of this section.

4.2 SVMseq for classification

In this section, we provide the pseudocode for the sequential learning algorithm. The algorithm, on convergence, provides the optimal set of Lagrange multipliers h_i for the classification problem in high-dimensional feature space which corresponds to the maximum of the cost function (35).

Sequential Algorithm for Classification

1. Initialize $h_i = 0$. Compute matrix $[D]_{ij} = y_i y_j (K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \lambda^2)$ for $i, j = 1, \ldots, l$.

2. For each pattern, i=1 to l, compute

2.1 $E_i = \sum_{j=1}^{l} h_j D_{ij}$.

2.2 $\delta h_i = \min\{\max[\gamma(1 - E_i), -h_i], C - h_i\}$.

2.3 $h_i = h_i + \delta h_i$.

3. If the training has converged, then **stop** else **goto step 2**.

Here, γ refers to the learning rate, theoretical bounds for which will be derived in the following analysis. The kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ can be any function which satisfies the Mercer's condition[6]

$$\int\int K(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{x}) f(\boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} \geq 0, \forall\ f \in L^2, \tag{39}$$

examples of which are the Gaussian and polynomial kernels given in Section 3.4. We can check for the convergence of the algorithm either by monitoring the updates of each Lagrange multiplier δh_i or by looking at the rate of change of the cost function, $\Delta L'_D$, which can be written as

$$\Delta L'_D = \delta h_i^t (1 - E_i^t - \frac{1}{2} D_{ii} \delta h_i^t) \tag{40}$$

as shown in Section 4.3.2.

4.3 Rigorous proofs on convergence

Now we prove the learning algorithm can indeed find the optimal solution of the dual problem given by eq.(35). The proof involves showing that the constraint of eq.(36) is always satisfied and that the algorithm will converge to the global maximum of the cost function given by eq.(35).

4.3.1 The constraint (36) is always satisfied during the learning process

For all the possible updates of h_i, we have

1. If $\delta h_i^t = C - h_i^t$; then, $h_i^{t+1} = h_i^t + \delta h_i^t = C$.
2. If $\delta h_i^t = \max\{\gamma(1 - E_i^t), -h_i^t\}$;
   then,$h_i^{t+1} = h_i^t + \delta h_i^t \geq h_i^t - h_i^t = 0$.
   Since, in this case $C - h_i^t \geq \max\{\gamma(1 - E_i^t), -h_i^t\}$;
   hence, $h_i^{t+1} \leq h_i^t + C - h_i^t = C$

So, for any initial values of h_i^0, h_i^1 will satisfy the constraint (36) after just one step learning and will maintain this property forever.

4.3.2 The learning process will monotonically increase $L'_D(\boldsymbol{h})$ and stop when a local maximum is reached

Define $a \equiv \max_{\{i\}} D_{ii}$. We prove that for the learning rate bounded by $0 < \gamma < 2/a$, $L'_D(\boldsymbol{h})$ will monotonically increase during the learning process.

$$\begin{aligned}
\Delta L'_D &= L'_D(h_i^t + \delta h_i^t) - L'_D(h_i^t) \\
&= \delta h_i^t - \frac{1}{2} \sum_j h_j^t D_{ji} \delta h_i^t - \frac{1}{2} \sum_j D_{ij} h_j^t \delta h_i^t \\
&\quad - \frac{1}{2} D_{ii} (\delta h_i^t)^2.
\end{aligned}$$

Since $D_{ij} = D_{ji}$ and $\sum_j D_{ij} h_j^t = E_i^t$, we have

$$\Delta L'_D = \delta h_i^t (1 - E_i^t - \frac{1}{2} D_{ii} \delta h_i^t). \tag{41}$$

Now, considering the three possible updates of δh_i^t:

Case 1: $\delta h_i^t = \gamma(1 - E_i^t)$

$$\begin{aligned}
\Delta L'_D &= (1 - E_i^t)^2 (\gamma - \frac{1}{2} D'_{ii} \gamma^2) \\
&\geq (1 - E_i^t)^2 (\gamma - \frac{1}{2} a\, \gamma^2) \geq 0
\end{aligned}$$

Case 2: $\delta h_i^t = -h_i^t$

This case arises only when $-h_i^t \geq \gamma(1 - E_i^t)$. Remember $h_i^t \geq 0$, which implies $E_i^t \geq 1$ in this case. Substituting δh_i^t in eq.(41), we get

$$\Delta L'_D = -h_i^t (1 - E_i^t + \frac{1}{2} D_{ii} h_i^t) \tag{42}$$

Notice that

$$\begin{aligned}
1 - E_i^t + \frac{1}{2} D_{ii} h_i^t &\leq 1 - E_i^t + \frac{1}{2} D_{ii} \gamma (E_i^t - 1) \\
&\leq (1 - E_i^t)(1 - \frac{1}{2} D_{ii} \gamma) \leq 0
\end{aligned} \tag{43}$$

So, from eqs.(42) and (43) $\Delta L'_D \geq 0$.

Case 3: $\delta h_i^t = C - h_i^t$

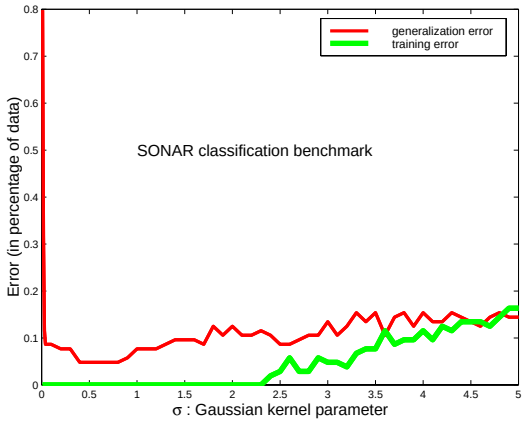
This case arises only when $C - h_i^t \leq \gamma(1 - E_i^t)$. Remember $h_i^t \leq C$, which implies $E_i^t \leq 1$ in this case. Therefore,

$$\begin{aligned}
\Delta L'_D &= (C - h_i^t)[1 - E_i^t - \frac{1}{2} D_{ii} (C - h_i^t)] \\
&\geq (C - h_i^t)[1 - E_i^t - \frac{1}{2} D_{ii} \gamma (1 - E_i^t)] \\
&\geq (C - h_i^t)(1 - E_i^t)(1 - \frac{1}{2} D_{ii} \gamma) \geq 0.
\end{aligned}$$

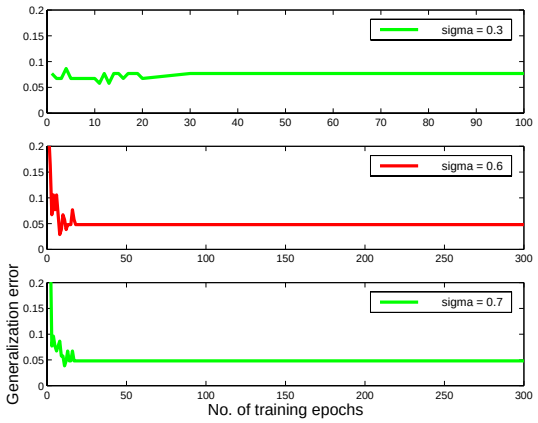
So for all the three possible values of δh_i, $L'_D(\boldsymbol{h})$ will monotonically increase and stop when a local maximum is reached. In the following we will show that the local maximum is also the global one.

4.3.3 Every local maximum is also the global one

Since $\sum_{ij} h_i D_{ij} h_j = (\sum_i h_i y_i \boldsymbol{x}_i) \cdot (\sum_i h_i y_i \boldsymbol{x}_i) \geq 0$ for any $\boldsymbol{h}$, $\mathbf{D}$ is a semidefinite matrix, and $L'_D(\boldsymbol{h})$ is a convex function. For the linear constraints that arises in this optimization problem, we show that each local maximum is equivalent to the global one. Suppose $\boldsymbol{h}_1$ is a local maximum point, while $\boldsymbol{h}_2$ is the global maximum which has a larger value, i.e., $L'_D(\boldsymbol{h}_2) > L'_D(\boldsymbol{h}_1)$. We can construct a straight line connecting $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$, which is parameterized as $(1 - \lambda)\boldsymbol{h}_1 + \lambda \boldsymbol{h}_2$, for $0 \leq \lambda \leq 1$. Since the constraints are linear, all the data points on the line are feasible solutions. On the other hand, since $L'_D(\boldsymbol{h})$ is a convex function, $L'_D(\boldsymbol{h}(\lambda)) \geq (1 - \lambda) L(\boldsymbol{h}_1) + \lambda L'_D(\boldsymbol{h}_2)$. For $\lambda > 0$, $L'_D(\boldsymbol{h}(\lambda)) > L'_D(\boldsymbol{h}_1)$, which contradicts the assumption that $\boldsymbol{h}_1$ is a local maximum. So $\boldsymbol{h}_1$ must also be the global maximum solution.



(a)



(b)

Figure 2: Sonar data classification using SVMseq: (a) Training and generalization error obtained using various Gaussian kernel parameter (b) Generalization error vs the number of training epochs plotted for three different kernel parameters

Sections 4.3.1, 4.3.2 and 4.3.3 combined completes the proof of convergence. The analysis also gives us a theoretical upper bound on the learning rate to be used :

$$\gamma \leq 2/\max_{\{i\}} D_{ii}. \tag{44}$$

5 Experimental results

We now proceed to look at some experimental classification results using the SVMseq algorithm. The aim of this section is to demonstrate that the SVMseq algorithm is capable of achieving the high levels of generalization capability boasted by the SVM kind of classifiers and it does so with much lesser amounts of computation besides being robust to parameter choices in a broad range. We use benchmark problems reported in [7] as well some others. Results for the the case without bias are similar but are improved by introducing bias and soft margin as done in our modified formulation. Moreover, in our method the fixing of learning rates and open parameters have a firm theoretical basis as an off-shoot of the rigorous theoretical analysis.

Here, we give examples of using RBF and polynomial type non-linear decision boundaries. The learning rate γ is fixed to $1.9/\max_{\{i\}} D_{ii}$ based on eq.(44) and the convergence of the algorithm is determined by looking at a trace of the cost function which is easily computed using eq.(40).

5.1 Sonar classification benchmark

We consider a benchmark problem to classify the sonar data set of 208 patterns representing metal cylinders(mines) and rocks[9]. Each of the patterns has 60 input dimensions and the aspect-angle dependent data set has been split into a training set and testing set of 104 patterns each as recommended by Gorman and Sejnowski[9]. We use a Gaussian RBF kernel (refer Section 3.4) with various values of the variance parameter σ in our non-linear implementation.

Table 2: Sonar classification: Gorman and Sejnowski's result with standard NNs using BP algo[9]

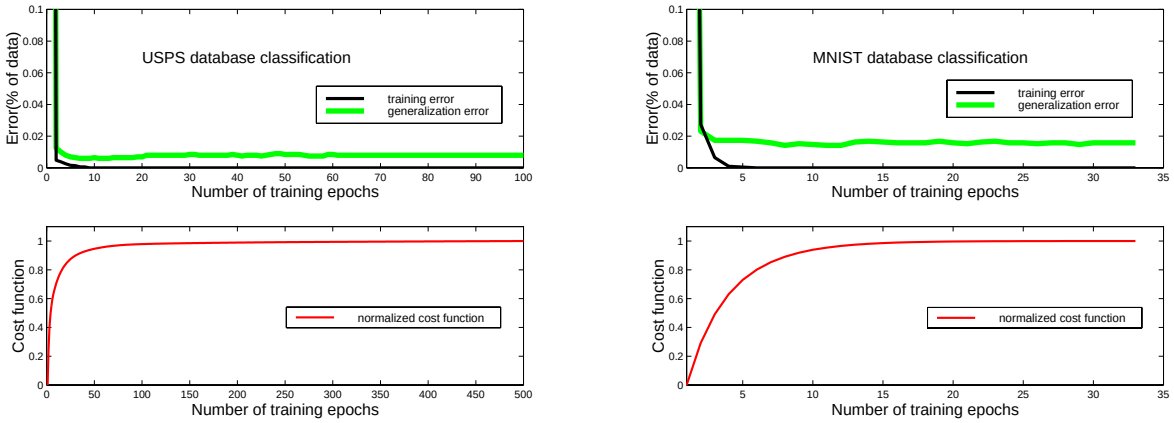
#hidden	0	2	6	12	24
%gen.	73.1	85.7	89.3	90.4	89.2

Table 3: Sonar classification: Gen. error and computational cost for standard SVM and SVMseq

ALGO	Epoch	%gen.(#Errors)	FLOPS
SVM	-	93.3% (7)	6.8×10^9
SVMseq	10	93.3% (7)	383520
SVMseq	50	95.2% (5)	1917600
SVMseq	130	95.2% (5)	4985760

Tables 2 and 3 compares the generalization performance of our method with the BP trained NNs of Gorman and Sejnowski [9] (upto 24 hidden units and 300 epochs through training data) and the standard SVM method on the lines of the analysis in Friess et al.[7]. The values shown are obtained with parameters $\sigma = 0.6$ and $C = 50$. The generalization performance achieved by SVMseq is superior to the NN based method (Table 2) while it is similar to the standard SVM. However, the SVMseq is orders faster than the standard SVM as is reflected through the FLOPS comparison in MATLAB (Table 3). The last row (epochs=130) is when the algorithm detected convergence.

We also plot the evolution of the generalization and



(a) (b)

Figure 3: Handwritten character recognition : Training, generalization error and the cost function trace with number of training epochs for (a) USPS 16×16 pixel database (b) MNIST 28×28 pixel database.

Table 4: USPS dataset two-class classification comparisons: number of test errors (out of 2007 test patterns)

Digits	0	1	2	3	4	5	6	7	8	9
classical RBF	20	16	43	38	46	31	15	18	37	26
Optimally tuned SVM	16	12	25	24	32	24	14	16	26	16
SVMseq (poly.deg.=3)	19	14	31	29	33	30	17	16	34	26
SVMseq (poly.deg.=4)	14	13	30	28	29	27	14	16	28	19
SVMseq (RBF:σ=3.5)	11	7	25	24	30	25	17	14	24	14

training error for various values of the kernel parameter in Fig.2(a). It is noted that generalization error is flat in a fairly broad region of the parameter space. Fig.2(b) shows that the algorithm has very fast convergence speed(in terms of training epochs) if we are in the region with an approximately correct kernel parameter.

5.2 USPS and MNIST handwritten character classification

The next classification task we considered used the USPS database of 9298 handwritten digits (7291 for training, 2007 for testing) collected from mail envelopes in Buffalo[11]. Each digit is a 16×16 vector with entries between -1 and +1. Preprocessing consists of smoothing with a Gaussian kernel of width $\sigma = 0.75$. We used polynomial kernels of third and fourth degree and RBF kernels to generate the non-linear decision surface.

Here, we consider the task of two-class binary classification, i.e. separating one digit from the rest of the digits. Table 4 shows comparisons of the generalization performance of the SVMseq using polynomial and RBF kernels against the optimally tuned SVM with RBF kernels (performance reported in Scholkopf et al.[13]) and classical RBF networks. We see that inspite of absolutely no tuning of parameters, SVMseq achieves much better results than classical RBF and comparable results to that of best tuned standard SVMs.

Fig.3(a) shows a typical learning performance, in this case for the classification of digit '0', using SVMseq. The training error drops to zero very fast(in a few epochs) and the test error also reaches a low of around .7% which corresponds to an average of about 14 test errors. The plot of the cost function $L'_D(\boldsymbol{h})$, which is an indicator of algorithm convergence, is also shown to reach the maximum very quickly.

6 SV regression and it's sequential implementation

In this section we will proceed to show how the idea can be extended to incorporate SVM regression. The basic idea of the SV algorithm for regression estimation is to compute a linear function in some high dimensional feature space (which possesses a dot product) and thereby, compute a non-linear function in the space of the input data. Working in the spirit of the SRM principle, we minimize empirical risk and maxi-

mize margins by solving the following constrained optimization problem[16]:

$$\text{Min. } J(\xi, \xi^*, \boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C(\sum_{i=1}^{l} \xi_i + \sum_{i=1^l} \xi_i^*) \quad (45)$$

$$\text{subject to} \quad y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i \leq \epsilon + \xi_i^*, \quad i = 1, \ldots, l \quad (46)$$

$$\boldsymbol{w} \cdot \boldsymbol{x}_i - y_i \leq \epsilon + \xi_i, \quad i = 1, \ldots, l \quad (47)$$

$$\xi_i, \xi_i^* \geq 0, \quad i = 1 \ldots, l. \quad (48)$$

Here, $\boldsymbol{x}$ refers to the input vector augmented with the augmenting factor λ to handle the bias term, ξ_i, ξ_i^* are the slack variables and ϵ is a user defined insensitivity bound for the error function[16]. The corresponding modified dual problem works out to be:

$$\text{Max. } J_D(\alpha, \alpha^*) = -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\boldsymbol{x}_i \cdot \boldsymbol{x}_j - \epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i^* - \alpha_i) \quad (49)$$

$$\text{subject to} \quad 0 \leq \alpha_i, \alpha_i^* \leq C, \quad i = 1, \ldots, l, \quad (50)$$

where α_i, α_i^* are non-negative Lagrange multipliers. The solution to this optimization problem is traditionally obtained using quadratic programming packages. The optimal approximation surface using the modified formulation, after extending the SVM to nonlinear case, is given as

$$f(\boldsymbol{x}) = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)(K(\boldsymbol{x}_i, \boldsymbol{x}) + \lambda^2). \quad (51)$$

As in the classification case, only some of the coefficients $(\alpha_i^* - \alpha_i)$ are non-zero, and the corresponding data points are called *support vectors*.

6.1 SVMseq algorithm for regression

We extend the sequential learning scheme devised for the classification problem to derive a sequential update scheme which can obtain the solution to the optimization problem of eq.(49) for SVM regression.

Sequential Algorithm for Regression

1. Initialize $\alpha_i = 0, \alpha_i^* = 0$. Compute $[R]_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \lambda^2$ for $i, j = 1, \ldots, l$.

2. For each training point, i=1 to l, compute

2.1 $E_i = y_i - \sum_{j=1}^{l}(\alpha_i^* - \alpha_i)R_{ij}$.

2.2 $\delta\alpha_i^* = \min\{\max[\gamma(E_i - \epsilon), -\alpha_i^*], C - \alpha_i^*\}$.
$\delta\alpha_i = \min\{\max[\gamma(-E_i - \epsilon), -\alpha_i], C - \alpha_i\}$.

2.3 $\alpha_i^* = \alpha_i^* + \delta\alpha_i^*$.
$\alpha_i = \alpha_i + \delta\alpha_i$.

3. If the training has converged, then **stop** else **goto step 2**.

Similar to the classification case, any non-linear kernel satisfying the Mercer's condition can be used. We have rigorously proved the convergence of the algorithm for the regression case, a result which will be part of another publication with detailed experimental evaluations of benchmark regression problems. Here, the user defined error insensitivity parameter ϵ controls the balance between the sparseness of the solution and the closeness to the training data. Recent works on finding the asymptotical optimal choice of ϵ-Loss [14] and soft ϵ-tube [12] give theoretical foundations for the optimal choice of the ϵ-loss function.

7 Discussion and concluding remarks

By modifying the formulation of the bias term, we were able to come up with a sequential update based learning scheme which solves the SVM classification and regression optimization problems with added advantages of conceptual simplicity and huge computational gains.

- By considering a modified dual problem, the global constraint in the optimization problem, $\sum_i h_i y_i = 0$, which is difficult to be satisfied in a sequential scheme is circumvented by absorbing it into the maximization functional.
- Unlike many other gradient ascent type algorithms, the SVMseq is guaranteed to converge (essentially due to the nature of the optimization problem) very fast in terms of the training epochs. Moreover, we have derived a theoretical upper bound for the learning rate, hence, ensuring that there are very few open parameters to be handled.
- The algorithm has been shown to be computationally more efficient than conventional SVM routines through comparisons of the computational complexity in MATLAB routines. Moreover, in the proposed learning procedure, non-support vector Lagrangian values go exactly to zero as opposed to numerical round-off errors that are characteristic of quadratic optimization routine packages.
- The algorithm also deals with noisy data by implementing soft classifier type decision surfaces which can take care of outliers. Most importantly, the algorithm has been extended to handle the SVM for regression under the same general framework, something which very few other speed up techniques have considered.

- The SVM paradigm in the regression domain provide an automatic complexity and model order selection for function approximation - a characteristic that should favor these techniques to standard RBF, Neural Network or other parametric approaches.

In addition, this work looks at implications of reformulation of the bias by using the augmenting factor from a geometric point of view and interprets the effects seen in simulation results theoretically. The choice of the augmenting factor is also discussed. Moreover, the conditions for obtaining a balanced classifier, which is simple but conceptually important, is also derived.

The proposed algorithm is essentially a gradient ascent method in the modified dual problem by using extra clipping terms in the updates to satisfy the linear constraints. It is well known that the gradient ascent/descent may be trapped in a long narrow valley if the Hessian matrix (in our context, the matrices D or R in the dual problem) has wide spread eigen values. In our future work, we will work on implementing more advanced gradient descent methods, like the natural gradient descent of Amari[1], to overcome this prospective problem.

Acknowledgements

We would like to thank Prof. Shunichi Amari and Dr. Noboru Murata for constructive suggestions. The authors acknowledges the support of the RIKEN Brain Science Institute for funding this project.

References

[1] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[2] J.K. Anlauf and M. Biehl. The Adatron: an adaptive perceptron algorithm. *Europhysics Letters*, 10(7):687–692, 1989.

[3] P. Barlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[4] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings, Fifth Annual Workshop on Computational Learning Theory*. ACM Press, 1992.

[5] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 1998. (In press.).

[6] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[7] T.T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In *Proceedings, 15th Intl. Conference on Machine Learning*. Morgan-Kaufman, 1998.

[8] T.T. Friess and R.F. Harrison. Support vector neural networks: The kernel adatron with bias and soft margin. Technical Report 752, The Univ of Sheffield, Dept. of ACSE, 1998.

[9] R.P. Gorman and T.J. Sejnowski. Finding the size of a neural network. *Neural Networks*, 1:75–89, 1988.

[10] T. Joachims. Making large scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[11] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[12] B. Scholkopf, P. Bartlett, A.J. Smola, and R. Williamson. Support vector regression with automatic accuracy control. In *Proceedings, Intl. Conf. on Artificial Neural Networks (ICANN'98)*, volume 1, pages 111–116. Springer, 1998.

[13] B. Scholkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing Support Vector Machines with Gaussian kernels to radial basis function classifiers. A.I.Memo 1599, M.I.T. AI Labs, 1996.

[14] A.J. Smola, N. Murata, B. Scholkopf, and K.-R. Muller. Asymptotically optimal choice of ϵ-loss for support vector machines. In *Proceedings, Intl. Conf. on Artificial Neural Networks (ICANN'98)*, volume 1, pages 105–110. Springer, 1998.

[15] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons,Inc.,New York, 1997.

[16] V. Vapnik, S.E. Golowich, and A. Smola. Support vector method for function approximation, regression estimation and signal processing. In *Advances in Neural Information Processing Systems*, volume 9, pages 281–287, 1997.