Improvements to the SMO Algorithm for SVM Regression

S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy

***Abstract*—This paper points out an important source of inefficiency in Smola and Schölkopf's sequential minimal optimization (SMO) algorithm for support vector machine (SVM) regression that is caused by the use of a single threshold value. Using clues from the KKT conditions for the dual problem, two threshold parameters are employed to derive modifications of SMO for regression. These modified algorithms perform significantly faster than the original SMO on the datasets tried.**

***Index Terms*—Quadratic programming, regression, sequential minimal optimization (SMO) algorithm, support vector machine (SVM).**

I. Introduction

SUPPORT vector machine (SVM) is an elegant tool for solving pattern recognition and regression problems. Over the past few years, it has attracted a lot of researchers from the neural network and mathematical programming community; the main reason for this being its ability to provide excellent generalization performance. SVMs have also been demonstrated to be valuable for several real-world applications.

In this paper, we address the SVM regression problem. Recently, Smola and Schölkopf [9], [10] proposed an iterative algorithm, called sequential minimal optimization (SMO), for solving the regression problem using SVM. This algorithm is an extension of the SMO algorithm proposed by Platt [7] for SVM classifier design. Computational speed and ease of implementation are some of the noteworthy features of the SMO algorithm. In a recent paper [6], some improvements to Platt's SMO algorithm for SVM classifier design were suggested. In this paper, we extend those ideas to Smola and Schölkopf's SMO algorithm for regression. The improvements suggested in this paper enhance the value of SMO for regression even further. In particular, we point out an important source of inefficiency caused by the way SMO maintains and updates a single threshold value. Getting clues from optimality criteria associated with the Karush–Kuhn–Tucker (KKT) conditions for the dual problem, we suggest the use of two threshold parameters and devise two modified versions of SMO for regression that are much more efficient than the original SMO. Computational comparison on datasets show that the modifications perform significantly better than the original SMO.

This paper is organized as follows. In Section II we briefly discuss the SVM regression problem formulation, the dual problem and the associated KKT optimality conditions. We also point out how these conditions lead to proper criteria for terminating algorithms for designing SVM for regression. Section III gives a brief overview of the SMO algorithm for regression. In Section IV we point out the inefficiency associated with the way SMO uses a single threshold value and describe the modified algorithms in Section V. Computational comparison is done in Section VI. Pseudocodes that are needed for implementing the modified algorithms are given in [8].

Manuscript received December 7, 1999; revised May 20, 2000.

S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy are with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India (e-mail: shirish@csa.iisc.ernet.in; cbchiru@csa.iisc.ernet.in; murthy@csa.iisc.ernet.in).

S. S. Keerthi is with the Department of Mechanical and Production Engineering, National University of Singapore, Singapore 119260 (e-mail: mpessk@guppy.mpe.nus.edu.sg).

Publisher Item Identifier S 1045-9227(00)07868-1.

II. The SVM Regression Problem and Optimality Conditions

The basic problem addressed in this paper is the regression problem. The tutorial by Smola and Schölkopf [9] gives a good overview of the solution of this problem using SVM's. Throughout the paper we will use x to denote the input vector of the SVM and z to denote the feature space vector which is related to x by a transformation, $z = \phi(x)$. Let the training set, $\{x_i, d_i\}$, consist of m data points where x_i is the i-th input pattern and d_i is the corresponding target value, $d_i \in \mathbb{R}$. The goal of SVM regression is to estimate a function $f(x)$ that is as "close" as possible to the target values d_i for every x_i and at the same time, is as "flat" as possible for good generalization. The function f is represented using a linear function in the feature space

$$f(x) = w \cdot \phi(x) + b$$

where b denotes the bias. As in all SVM designs, we define the kernel function $k(x, \hat{x}) = \phi(x) \cdot \phi(\hat{x})$, where "$\cdot$" denotes inner product in the z space. Thus, all computations will be done using only the kernel function. This inner-product kernel helps in taking the dot product of two vectors in the feature space without having to construct the feature space explicitly. Mercer's theorem [2] tells the conditions under which this kernel operator is useful for SVM designs.

For SVM regression purposes, Vapnik [11] suggested the use of ϵ-insensitive loss function where the error is not penalized as long as it is less than ϵ. It is assumed here that ϵ is known *a priori*. Using this error function together with a regularizing term, and letting $z_i = \phi(x_i)$, the optimization problem solved by the support vector machine can be formulated as

$$\begin{aligned} \min\ & \frac{1}{2}\|w\|^2 + C\sum_i (\xi_i + \xi_i') \\ \text{s.t. } & d_i - w \cdot z_i - b \le \epsilon + \xi_i \\ & w \cdot z_i + b - d_i \le \epsilon + \xi_i' \\ & \xi_i, \xi_i' \ge 0 \end{aligned} \tag{P}$$

The above problem is referred to as the *primal* problem. The constant $C > 0$ determines the tradeoff between the smoothness

of f and the amount up to which deviations larger than ϵ are tolerated.

Let us define $w(\alpha,\alpha') = \sum_i(\alpha_i - \alpha_i')z_i$. We will refer to the $\alpha_i^{(\prime)}$ as Lagrange multipliers. Using Wolfe duality theory, it can be shown that the $\alpha^{(\prime)}$ are obtained by solving the following *Dual* problem:

$$\begin{aligned}&\max \sum_i d_i(\alpha_i - \alpha_i') - \epsilon\sum_i(\alpha_i + \alpha_i') - \frac{1}{2}\|w(\alpha,\alpha')\|^2\\ &\text{s.t. } \sum_i(\alpha_i - \alpha_i') = 0\\ &\quad \alpha_i, \alpha_i' \in [0, C] \quad \forall i. \end{aligned} \tag{D}$$

Once the α_i and α_i' are obtained, the primal variables, w, b, ξ_i, and ξ_i' can be easily determined by using the KKT conditions for the primal problem.

The feature space (and hence w) can be infinite dimensional. This makes it computationally difficult to solve the primal problem (P). The numerical approach in SVM design is to solve the dual problem since it is a finite-dimensional optimization problem. (Note that $w(\alpha,\alpha')\cdot w(\alpha,\alpha') = \sum_i\sum_j(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')k(x_i, x_j)$.) To derive proper stopping conditions for algorithms which solve the dual, it is important to write down the optimality conditions for the dual. The Lagrangian for the dual is

$$\begin{aligned}L_D = &\frac{1}{2}\|w(\alpha,\alpha')\|^2 - \sum_i d_i(\alpha_i - \alpha_i')\\ &+ \epsilon\sum_i(\alpha_i + \alpha_i') + \beta\sum_i(\alpha_i - \alpha_i') - \sum_i \pi_i\alpha_i\\ &- \sum_i \psi_i\alpha_i' - \sum_i \delta_i(C - \alpha_i) - \sum_i \eta_i(C - \alpha_i').\end{aligned}$$

Let

$$F_i = d_i - w(\alpha,\alpha')\cdot z_i.$$

The KKT conditions for the dual problem are

$$\begin{aligned}\frac{\partial L_D}{\partial \alpha_i} &= -F_i + \epsilon + \beta - \pi_i + \delta_i = 0\\ \frac{\partial L_D}{\partial \alpha_i'} &= F_i + \epsilon - \beta - \psi_i + \eta_i = 0\\ \pi_i\alpha_i = 0, &\quad \pi_i \ge 0, \quad \alpha_i \ge 0\\ \psi_i\alpha_i' = 0, &\quad \psi_i \ge 0, \quad \alpha_i' \ge 0\\ \delta_i(C - \alpha_i) = 0, &\quad \delta_i \ge 0, \quad \alpha_i \le C\\ \eta_i(C - \alpha_i') = 0, &\quad \eta_i \ge 0, \quad \alpha_i' \le C.\end{aligned}$$

These conditions are both necessary and sufficient for optimality. Hereafter, we will refer to them as *optimality conditions*. These optimality conditions can be simplified by considering the following five cases. It is easy to check that at optimality, for every i, α_i and α_i' cannot be nonzero at the same time. Hence cases corresponding to $\alpha_i\alpha_i' \neq 0$ have been left out. (It is worth noting here that in the SMO regression algorithm and its modifications discussed in this paper, the condition, $\alpha_i\alpha_i' = 0\ \forall i$ is maintained throughout.)

Case 1) $\alpha_i = \alpha_i' = 0$

$$-\epsilon \le (F_i - \beta) \le \epsilon. \tag{1a}$$

Case 2) $\alpha_i = C$

$$F_i - \beta \ge \epsilon. \tag{1b}$$

Case 3) $\alpha_i' = C$

$$F_i - \beta \le -\epsilon. \tag{1c}$$

Case 4) $\alpha_i \in (0, C)$

$$F_i - \beta = \epsilon. \tag{1d}$$

Case 5) $\alpha_i' \in (0, C)$

$$F_i - \beta = -\epsilon. \tag{1e}$$

Define the following index sets at a given α: $I_{0a} = \{i : 0 < \alpha_i < C\}$; $I_{0b} = \{i : 0 < \alpha_i' < C\}$; $I_1 = \{i : \alpha_i = 0, \alpha_i' = 0\}$; $I_2 = \{i : \alpha_i = 0, \alpha_i' = C\}$; $I_3 = \{i : \alpha_i = C, \alpha_i' = 0\}$. Also, let $I_0 = I_{0a} \cup I_{0b}$. Let us also define $\tilde{F}_i$ and $\bar{F}_i$ as

$$\begin{aligned}\tilde{F}_i &= F_i + \epsilon \quad \text{if } i \in I_{0b} \cup I_2,\\ &= F_i - \epsilon \quad \text{if } i \in I_{0a} \cup I_1.\end{aligned}$$

and

$$\begin{aligned}\bar{F}_i &= F_i - \epsilon \quad \text{if } i \in I_{0a} \cup I_3,\\ &= F_i + \epsilon \quad \text{if } i \in I_{0b} \cup I_1.\end{aligned}$$

Using these definitions we can rewrite the necessary conditions mentioned in (1a)–(1e) as

$$\begin{aligned}\beta &\ge \tilde{F}_i \quad \forall i \in I_0 \cup I_1 \cup I_2\\ \beta &\le \bar{F}_i \quad \forall i \in I_0 \cup I_1 \cup I_3.\end{aligned} \tag{2}$$

It is easily seen that the optimality conditions will hold iff there exists a β satisfying (2).

Let us define

$$\begin{aligned}b_{\text{up}} &= \min\{\bar{F}_i : i \in I_0 \cup I_1 \cup I_3\}\\ b_{\text{low}} &= \max\{\tilde{F}_i : i \in I_0 \cup I_1 \cup I_2\}.\end{aligned} \tag{3}$$

Then the optimality conditions will hold at some α iff

$$b_{\text{low}} \le b_{\text{up}}. \tag{4}$$

There exists a close relationship between the threshold parameter b in the primal problem and the multiplier, β. The KKT conditions of the primal (P) and dual (D) problem imply that, *at optimality, β and b are identical*. This can be seen from the following: 1) setting the derivative of the Lagrangian of (P) with respect to b to zero gives the equality constraint in (D) and 2) in L_D, β is the Lagrangian multiplier of that equality constraint; and hence, at optimality b and β coincide. Therefore, in the rest of the paper, β and b will denote one and the same quantity.

We will say that an index pair (i, j) defines a *violation* at (α,α') if one of the following two sets of conditions holds:

$$i \in I_0 \cup I_1 \cup I_2, \quad j \in I_0 \cup I_1 \cup I_3 \quad \text{and} \quad \tilde{F}_i > \bar{F}_j \tag{5a}$$

$$i \in I_0 \cup I_1 \cup I_3, \quad j \in I_0 \cup I_1 \cup I_2 \quad \text{and} \quad \bar{F}_i < \tilde{F}_j. \tag{5b}$$

Note that optimality condition will hold at α iff there does not exist any index pair (i, j) that defines a violation.

Since, in numerical solution, it is usually not possible to achieve optimality exactly, there is a need to define approximate optimality conditions. The condition (4) can be replaced by

$$b_{\text{low}} \leq b_{\text{up}} + 2\tau \tag{6}$$

where τ is a positive tolerance parameter. (In the pseudocodes given in [8], [10], this parameter is referred to as `tol`). Correspondingly, the definition of violation can be altered by replacing (5a) and (5b), respectively, by

$$i \in I_0 \cup I_1 \cup I_2, \quad j \in I_0 \cup I_1 \cup I_3 \quad \text{and} \quad \tilde{F}_i > \bar{F}_j + 2\tau \tag{7a}$$

$$i \in I_0 \cup I_1 \cup I_3, \quad j \in I_0 \cup I_1 \cup I_2 \quad \text{and} \quad \bar{F}_i < \tilde{F}_j - 2\tau. \tag{7b}$$

Hereafter in the paper, when optimality is mentioned it will mean approximate optimality.

Let

$$E_i = F_i - \beta.$$

Using (1) it is easy to check that optimality holds iff there exists a b such that the following hold for every i:

$$\alpha_i > 0 \Rightarrow E_i \geq \epsilon - \tau \tag{8a}$$

$$\alpha_i < C \Rightarrow E_i \leq \epsilon + \tau \tag{8b}$$

$$\alpha_i' > 0 \Rightarrow E_i \leq -\epsilon + \tau \tag{8c}$$

$$\alpha_i' < C \Rightarrow E_i \geq -\epsilon - \tau. \tag{8d}$$

These conditions are used in [9], [10] together with a special choice of b to check if an example violates the KKT conditions. However, unless the choice of b turns out to be right, using the above conditions for checking optimality can be incorrect. We will say more about this in Section IV after a brief discussion of Smola and Schölkopf's SMO algorithm in the next section.

It is also possible to give an alternate approximate optimality condition based on the closeness of the dual objective function to the optimal value, estimated using duality gap ideas. This is fine, but care is needed in choosing the tolerance used; see [5] for a related discussion. This criterion has the disadvantage that it is a single global condition involving all i. On the other hand, (7) consists of an individual condition for each pair of indexes and hence is much better suited for the methods discussed in this paper. In particular, when (i, j) satisfies (7) then a strict improvement in the dual objective function can be achieved by optimizing only the Lagrange multipliers corresponding to the examples i and j. (This is true even if $\tau = 0$.)

III. Smola and Schölkopf's SMO Algorithm for Regression

A number of algorithms have been suggested for solving the dual problem. Smola and Schölkopf [9], [10] give a detailed view of these algorithms and their implementations. Traditional quadratic programming algorithms such as interior point algorithms are not suitable for large size problems because of the following reasons. First, they require that the kernel matrix be computed and stored in memory. This requires extremely large memory. Second, these methods involve expensive matrix operations such as Cholesky decomposition of a large submatrix of the kernel matrix. Third, coding of these algorithms is difficult.

Attempts have been made to develop methods that overcome some or all of these problems. One such method is chunking. The idea here is to operate on a fixed size subset of the training set at a time. This subset is called the working set and the optimization subproblem is solved with respect to the variables corresponding to the examples in the working set and a set of support vectors for the current working set is found. These current support vectors are then used to determine the new working set. This new set is nothing but a set of worst input data points which violate the optimality conditions for the current estimator. The new optimization subproblem is solved and this process is repeated until the optimality conditions are satisfied for all the examples.

Platt [7] proposed an algorithm, called SMO for the SVM classifier design. This algorithm puts chunking to the extreme by iteratively selecting working sets of size two and optimizing the target function with respect to them. One advantage of using working sets of size two is that the optimization subproblem can be solved analytically. The chunking process is repeated till all the training examples satisfy the optimality conditions. Smola and Schölkopf [9], [10] extended these ideas for solving the regression problem using SVM's. We describe this algorithm very briefly below. The details, together with a pseudocode can be found in [9], [10]. We assume that the reader is familiar with them. To convey our ideas compactly we employ the notations used in [9], [10].

The basic step in the SMO algorithm consists of choosing a pair of indexes, (i_1, i_2) and optimizing the dual objective function in (D) by varying the Lagrange multipliers corresponding to i_1 and i_2 only. We make one important comment here on the role of the threshold parameter, β. As in [9], [10] and in Section II, let E_i denote the output error on the ith pattern. Let us call the indices of the two multipliers chosen for joint optimization in one step as i_2 and i_1. To take a step by varying the Lagrange multipliers of examples i_1 and i_2, we only need to know $E_{i_1} - E_{i_2} = F_{i_1} - F_{i_2}$. *Therefore a knowledge of the value of β is not needed to take a step.*

The method followed to choose i_1 and i_2 at each step is crucial for finding the solution of the problem efficiently. The SMO algorithm employs a two loop approach: the outer loop chooses i_2; and, for a chosen i_2 the inner loop chooses i_1. The outer loop iterates over all patterns violating the optimality conditions, first only over those with Lagrange multipliers neither on the upper nor on the lower boundary (in Smola and Schölkopf's pseudocode this looping is indicated by `examineAll = 0`), and once all of them are satisfied, over all patterns violating the optimality conditions (`examineALL = 1`) to ensure that the problem has indeed been solved. For efficient implementation a cache for E_i is maintained and updated for those indices i corresponding to nonboundary Lagrange multipliers. The remaining E_i are computed as and when needed.

Let us now see how the SMO algorithm chooses i_1. The aim is to make a large increase in the objective function. Since it is

expensive to try out all possible choices of i_1 and choose the one that gives the best increase in the objective function, the index i_1 is chosen to maximize $|E_{i_2} - E_{i_1}|$ or $|E_{i_2} - E_{i_1} \pm 2\epsilon|$ depending on the multipliers of i_1 and i_2. Since E_i is available in cache for nonboundary multiplier indexes, only such indexes are initially used in the above choice of i_1. If such a choice of i_1 does not yield sufficient progress, then the following steps are taken. Starting from a randomly chosen index, all indexes corresponding to nonbound multipliers are tried as a choice for i_1, one by one. If still sufficient progress is not possible, all indexes are tried as choices for i_1, one by one, again starting from a randomly chosen index. Thus the choice of random seed affects the running time of SMO.

Although a value of β is not needed to take a step, it is needed if (8a)–(8d) are employed for checking optimality. In the SMO algorithm β is updated after each step. A value of β is chosen so as to satisfy (1) for $i \in \{i_1, i_2\}$. If, after a step involving (i_1, i_2), one of $\alpha_{i_1}, \alpha_{i_2}$ (or both) takes a nonboundary value then (1d) or (1e) is exploited to update the value of β. In the rare case that this does not happen, there exists a whole interval, say, $[\beta_{\text{low}}, \beta_{\text{up}}]$, of admissible thresholds. In this situation SMO simply chooses β to be the midpoint of this interval.

IV. Inefficiency of the SMO Algorithm

The SMO algorithm for regression, discussed above, is very simple and easy to implement. However it can become inefficient because of its way of computing and maintaining a single threshold value. At any instant, the SMO algorithm fixes β based on the current two indexes used for joint optimization. However, while checking whether the remaining examples violate optimality or not, it is quite possible that a different, shifted choice of β may do a better job. So, in the SMO algorithm it is quite possible that, even though (α, α') has reached a value where optimality is satisfied (i.e., (6)), SMO has not detected this because it has not identified the correct choice of β. It is also quite possible that, a particular index may appear to violate the optimality conditions because (8) is employed using an "incorrect" value of β although this index may not be able to pair with another to make progress in the objective function. In such a situation the SMO algorithm does an *expensive* and wasteful search looking for a second index so as to take a step. We believe that this is a major source of inefficiency in the SMO algorithm.

There is one simple alternate way of choosing β that involves all indices. By duality theory, the objective function value in (P) of a primal feasible solution is greater than or equal to the objective function value in (D) of a dual feasible solution. The difference between these two values is referred to as the *duality gap*. The duality gap is zero only at optimality. Suppose (α, α') is given and $w = w(\alpha, \alpha')$. For each i, the term ξ_i can be chosen optimally (as a function of β). The result is that the duality gap is expressed as a function of β only. One possible way of improving the SMO algorithm is to always choose β so as to minimize the duality gap. This corresponds to the subproblem

$$\min_{\beta} \sum_i \max(0, F_i - \beta - \epsilon, -F_i + \beta - \epsilon).$$

The solution of this subproblem is easy. Let m denote the number of examples. In an increasing order arrangement of $\{F_i - \epsilon\}$ and $\{F_i + \epsilon\}$ let f_j and f_{j+1} be the jth and $(j+1)$-th values. The function optimized in the above minimization problem is a convex function of β. The slope of this convex function is negative at all β values smaller than f_m, it is positive at all β values bigger than f_{m+1} and it is zero for $\beta \in (f_m, f_{m+1})$. Therefore any β in the interval, $[f_m, f_{m+1}]$ is a minimizer. The determination of f_m and f_{m+1} can be done efficiently using a "median-finding" technique. Since all F_i are not typically available at a given stage of the algorithm, it is appropriate to apply the above idea to that subset of indexces for which F_i are available. This set contains I_0. We implemented this idea and tested it on some benchmark problems. The idea led to a reasonable improvement over the original SMO. However the modifications that we suggest in the next section gave much greater improvements. See Section VI for performances on some examples.

V. Modifications of the SMO Algorithm

In this section, we suggest two modified versions of the SMO algorithm for regression, each of which overcomes the problems mentioned in the last section. As we will see in the computational evaluation of Section VI, these modifications are always better than the original SMO algorithm for regression and in most situations, they also give quite a remarkable improvement in efficiency.

In short, the modifications avoid the use of a single threshold value β and the use of (8) for checking optimality. Instead, two threshold parameters, b_{up} and b_{low} are maintained and (6) (or (7)) is employed for checking optimality. Assuming that the reader is familiar with [9] and the pseudocodes for SMO given there, we only give a set of pointers that describe the changes that are made to Smola and Schölkopf's SMO algorithm for regression. Pseudocodes that fully describe these can be found in [8].

1) Suppose, at any instant, F_i is available for all i. Let i_{low} and i_{up} be indices such that

$$\tilde{F}_{i_{\text{low}}} = b_{\text{low}} = \max\{\tilde{F}_i : i \in I_0 \cup I_1 \cup I_2\} \tag{9a}$$

and

$$\bar{F}_{i_{\text{up}}} = b_{\text{up}} = \min\{\bar{F}_i : i \in I_0 \cup I_1 \cup I_3\}. \tag{9b}$$

Then checking a particular i for optimality is easy. For example, suppose $i \in I_3$. We only have to check if $\bar{F}_i < b_{\text{low}} - 2\tau$. If this condition holds, then there is a violation and in that case SMO's `takeStep` procedure can be applied to the index pair (i, i_{low}). Similar steps can be given for indexes in other sets. Thus, in our approach, the checking of optimality of the first index, i_2 and the choice of second index, i_1, go hand in hand, unlike the original SMO algorithm. As we will see below, we compute and use $(i_{\text{low}}, b_{\text{low}})$ and $(i_{\text{up}}, b_{\text{up}})$ via an efficient updating process.

2) To be efficient, we would, like in the SMO algorithm, spend much of the effort altering $\alpha_i, i \in I_0$; cache for $F_i, i \in I_0$ are maintained and updated to do this efficiently. And, when optimality holds for all $i \in I_0$, only then all indices are examined for optimality.

3) The procedure `takeStep` is modified. After a successful step using a pair of indices, (i_2, i_1), let $\hat{I} = I_0 \cup \{i_1, i_2\}$. We compute, *partially*, $(i_{\text{low}}, b_{\text{low}})$ and $(i_{\text{up}}, b_{\text{up}})$ using $\hat{I}$ only (i.e., use only $i \in \hat{I}$ in (9)). Note that these extra steps are inexpensive because cache for $\{F_i, i \in I_0\}$ is available and updates of F_{i_1}, F_{i_2} are easily done. A careful look shows that, since i_2 and i_1 have been just involved in a successful step, each of the two sets, $\hat{I} \cap (I_0 \cup I_1 \cup I_2)$ and $\hat{I} \cap (I_0 \cup I_1 \cup I_3)$, is nonempty; hence the partially computed $(i_{\text{low}}, b_{\text{low}})$ and $(i_{\text{up}}, b_{\text{up}})$ will not be null elements. Since i_{low} and i_{up} could take values from $\{i_2, i_1\}$ and they are used as choices for i_1 in the subsequent step (see item 1 above), we keep the values of F_{i_1} and F_{i_2} also in cache.

4) When working with only $\alpha_i, \alpha_i', i \in I_0$, i.e., a loop with `examineAll = 0`, one should note that, if (6) holds at some point then it implies that optimality holds as far as I_0 is concerned. (This is because, as mentioned in item 3 above, the choice of b_{low} and b_{up} are influenced by all indexes in I_0.) This gives an easy way of exiting this loop.

5) There are two ways of implementing the loop involving indexes in I_0 only (`examineAll = 0`).

Method 1: This is similar to what is done in SMO. Loop through all $i_2 \in I_0$. For each i_2, check optimality and if violated, choose i_1 appropriately. For example, if $\bar{F}_{i_2} < b_{\text{low}} - 2\tau$ then there is a violation and in that case choose $i_1 = i_{\text{low}}$.

Method 2: Always work with the worst violating pair, i.e., choose $i_2 = i_{\text{low}}$ and $i_1 = i_{\text{up}}$.

Depending on which one of these methods is used, we call the resulting overall modification of SMO as SMO-Modification 1 and SMO-Modification 2. SMO and SMO-Modification 1 are identical except in the way optimality is tested. On the other hand, SMO-Modification 2 can be thought of as a further improvement of SMO-Modification 1 where the cache is effectively used to choose the violating pair when `examineAll = 0`.

6) When optimality on I_0 holds, as already said we come back to check optimality on all indexes (`examineAll = 1`). Here we loop through all indexes, one by one. Since $(b_{\text{low}}, i_{\text{low}})$ and $(b_{\text{up}}, i_{\text{up}})$ have been partially computed using I_0 only, we update these quantities as each i is examined. For a given i, F_i is computed first and optimality is checked using the current $(b_{\text{low}}, i_{\text{low}})$ and $(b_{\text{up}}, i_{\text{up}})$; if there is no violation, F_i are used to update these quantities. For example, if $i \in I_3$ and $\bar{F}_i < b_{\text{low}} - 2\tau$, then there is a violation, in which case we take a step using (i, i_{low}). On the other hand, if there is no violation, then $(i_{\text{up}}, b_{\text{up}})$ is modified using $\bar{F}_i$, i.e., if $\bar{F}_i < b_{\text{up}}$ then we do: $i_{\text{up}} := i$ and $b_{\text{up}} := \bar{F}_i$.

7) Suppose we do as described above. What happens if there is no violation for any i in a loop having `examineAll = 1`? Can we conclude that optimality holds for all i? The answer to this question is affirmative. This is easy to see from the following argument. Suppose, by contradiction, there does exist one (i, j) pair such that they define a violation, i.e., they satisfy (7). Let us say, $i < j$. Then j would not have satisfied the optimality check in the above described implementation because either $\bar{F}_i$ or $\tilde{F}_i$ would have, earlier than j is seen, affected either the calculation of b_{up} and/or b_{low} settings. In other words, even if i is mistakenly taken as having satisfied optimality earlier in the loop, j will be detected as violating optimality when it is analyzed. Only when (6) holds it is possible for all indices to satisfy the optimality checks. Furthermore, when (6) holds and the loop over all indices has been completed, the true values of b_{up} and b_{low}, as defined in (3) would have been computed since all indices have been encountered. As a final choice of b (for later use in doing inference) it is appropriate to set: $b = 0.5(b_{\text{up}} + b_{\text{low}})$.

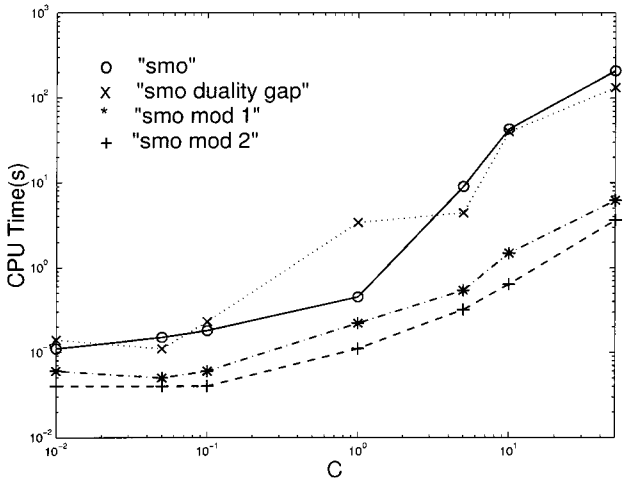

Fig. 1. Toy data: CPU time (in seconds) shown as a function of C.

VI. Computational Comparison

In this section we compare the performance of our modifications against Smola and Schölkopf's SMO algorithm for regression and the SMO algorithm based on duality gap ideas for threshold (see Section IV), on three datasets. We implemented all these methods in C and ran them using `gcc` on a P3 450 MHz Linux machine. The value, $\tau = 0.01$ was used for all experiments. For every dataset used in the experiments, we added an independent Gaussian noise of mean zero and some standard deviation. For every dataset, the value of ϵ was manually set in a random way depending upon the amount of noise added to the data. Similar experimental approaches include [4]. The input as well as the output are then normalized so as to have zero mean and a unit variance.

The first dataset is a toy dataset where the function to be approximated is a cubic polynomial, $.02x^3 + .05x^2 - x$. The domain of this function was fixed to $[-10, 10]$. The value, $\epsilon = 0.25$ was used. One hundred training samples were chosen randomly. The performance of the four algorithms for the polynomial kernel

$$k(x_i, x_j) = (1 + x_i \cdot x_j)^p$$

where p was chosen to be three (the degree of the polynomial from which the data is generated), is shown in Fig. 1.

The second dataset is the Boston housing dataset which is a standard benchmark for testing regression algorithms. This dataset is available at UCI Repository [1]. The dimension of the input is 13. We used a training set of size 406. The value, $\epsilon = .56$ was used. Fig. 2 shows the performance of the four algorithms on this dataset. For this as well as the third dataset the Gaussian kernel

$$k(x_i, x_j) = \exp(-0.5\|x_i - x_j\|^2/\sigma^2)$$

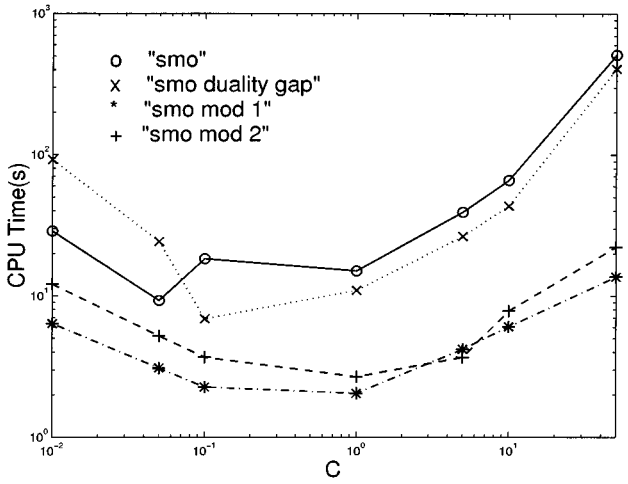

Fig. 2. Boston housing data: CPU time (in seconds) shown as a function of C.

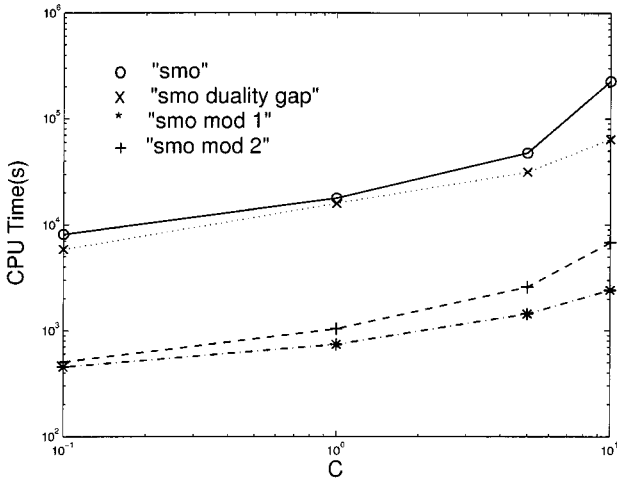

Fig. 3. Comp-Activ data: CPU time (in seconds) shown as a function of C.

was used and the σ^2 value employed was 2.5. The value of σ was chosen randomly. The SVM is rather insensitive to different choices of σ.

The third dataset, Comp-Activ, is available at the Delve website [3]. This dataset contains 8192 data points of which we used 5764. We implemented the "cpuSmall" prototask, which involves using 12 attributes to predict the fraction of time (in percentage) the CPU runs in user mode. We used $\epsilon = .48$ for this dataset. The performance of the four algorithms on this dataset is shown in Fig. 3.

It is very clear that both modifications outperform the original SMO algorithm. In many situations the improvement in ef-fi-ciency is remarkable. In particular, at large values of C the improvement is by an order of magnitude. Between the two modifications, it is difficult to say which one is better.

The primary purpose of these experiments is to examine differences in training times of all the four methods. However, studying the generalization abilities of these methods is an important task and we are currently investigating it. The results of this comparison will be reported elsewhere.

VII. Conclusion

In this paper, we have pointed out an important source of inefficiency in Smola and Schölkopf's SMO algorithm that is caused by the operation with a single threshold value. We have suggested two modifications of the SMO algorithm that overcome the problem by efficiently maintaining and updating two threshold parameters. Our computational experiments show that these modifications speed up the SMO algorithm significantly in most situations. The modifications can be easily implemented using the pseudocodes given in [8].

References

[1] C. L. Blake and C. J. Merz. (1998) UCI Repository of Machine Learning Databases. Univ. California, Dept. Inform. Comput. Sci., Irvine, CA. [Online]. Available: http://www.ics.uci.edu/~mlearn/ML-Repository.html

[2] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowledge Discovery*, vol. 3, no. 2, 1998.

[3] Delve: Data for Evaluating Learning in Valid Experiments [Online]. Available: http://www.cs.utoronto.ca/~delve

[4] O. L. Mangasarian and D. R. Musicant. (1999, Aug.) Massive Support Vector Regression. [Online] . Available: ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-02.ps

[5] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design," *IEEE Trans. Neural Networks*, vol. 11, pp. 124–136, Jan. 2000.

[6] ——, (1999, Aug.) Improvements to Platt's SMO Algorithm for SVM Classifier Design, Accepted for Publication in Neural Computation. Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore, Singapore. [Online]. Available: http://guppy.mpe.nus.edu.sg/~mpessk

[7] J. C. Platt, *Advances in Kernel Methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, Dec. 1998. Fast training of support vector machines using sequential minimal optimization.

[8] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. (1999, Aug.) Improvements to SMO Algorithm for Regression. Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore, Singapore. [Online]. Available: http://guppy.mpe.nus.edu.sg/~mpessk

[9] A. J. Smola, "Learning with Kernels," Ph.D. dissertation, GMD, Birlinghoven, Germany, 1998.

[10] A. J. Smola and B. Schölkopf, "A Tutorial on Support Vector Regression," Royal Holloway College, London, U.K., NeuroCOLT Tech. Rep. TR 1998-030, 1998.

[11] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.