Manuscript Number:2098

Improvements to Platt's SMO Algorithm for SVM Classifier Design

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya & K.R.K. Murthy[1]

[1]The first author is with the Dept. of Mechanical and Production Engineering, National University of Singapore, Singapore-119260. The other authors are with the Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore-560012, India.

Improvements to Platt's SMO Algorithm for SVM Classifier Design

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya & K.R.K. Murthy[†]

February 20, 2000

Abstract

This paper points out an important source of inefficiency in Platt's Sequential Minimal Optimization (SMO) algorithm 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. These modified algorithms perform significantly faster than the original SMO on all benchmark datasets tried.

1 Introduction

In the past few years, there has been a lot of excitement and interest in Support Vector Machines (Vapnik, 1995; Burges, 1998) because they have yielded excellent generalization performance on a wide range of problems. Recently, fast iterative algorithms that are also easy to implement have been suggested (Platt, 1998; Joachims, 1998; Mangasarian and Musicant, 1998; Friess, 1998; Keerthi et.al, 1999a). Platt's Sequential Minimization Algorithm (SMO) (Platt, 1998; Platt, 1999) is an important example. A remarkable feature of SMO is that

[†]The first author is with the Dept. of Mechanical and Production Engineering, National University of Singapore, Singapore-119260. The other authors are with the Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore-560012, India.

it is also extremely easy to implement. Comparative testing against other algorithms, done by Platt, have shown that SMO is often much faster and has better scaling properties.

In this paper we enhance the value of SMO 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 KKT conditions for the dual, we suggest the use of two threshold parameters and devise two modified versions of SMO that are more efficient than the original SMO. Computational comparison on benchmark datasets shows that the modifications perform significantly faster than the original SMO in most situations. The ideas mentioned in this paper can also be applied to the SMO regression algorithm (Smola, 1998). We will report the results of that extension in another paper (Shevade et.al, 1999).

The paper is organized as follows. In section 2 we briefly discuss the SVM 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 classifiers. Section 3 gives a short summary of Platt's SMO algorithm. In section 4 we point out the inefficiency associated with the way SMO uses a single threshold value, and describe the modified algorithms in section 5. Computational comparison is done in section 6.

2 SVM Problem and Optimality Conditions.

The basic problem addressed in this paper is the two category classification problem. The tutorial paper by Burges (Burges, 1998) gives a good overview of the solution of this problem using SVMs. Throughout the paper we will use x to denote the input vector of the support vector machine and z to denote the feature space vector which is related to x by a transformation, $z = \phi(x)$. As in all SVM designs, we do not assume ϕ to be known; all computations will be done using only the kernel function, $k(x, \hat{x}) = \phi(x) \cdot \phi(\hat{x})$, where "$\cdot$" denotes inner product in the z space. Let $\{(x_i, y_i)\}$ denote the training set, where x_i is the i-th input pattern and y_i is the corresponding target value; $y_i = 1$ means x_i is in class 1 and $y_i = -1$ means x_i is in class 2. Let $z_i = \phi(x_i)$. The optimization problem solved by the support vector machine is:

$$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i \quad \text{s.t.} \;\; y_i(w \cdot z_i - b) \geq 1 - \xi_i \;\; \forall\, i; \;\; \xi_i \geq 0 \;\forall\, i \tag{1}$$

This problem is referred to as the *primal* problem.

Let us define $w(\alpha) = \sum_i \alpha_i y_i z_i$. We will refer to the α_i's as Lagrange multipliers. The α_i's are obtained by solving the following *dual* problem:

$$\max W(\alpha) = \sum_i \alpha_i - \frac{1}{2} w(\alpha) \cdot w(\alpha) \quad \text{s.t.} \;\; 0 \leq \alpha_i \leq C \;\forall i; \;\; \sum_i \alpha_i y_i = 0 \tag{2}$$

Once the α_i's are obtained, the other primal variables, w, b and ξ can be easily determined by using the KKT conditions for the primal problem. It is possible that the solution is non-unique; for instance, when all α_is take the boundary values of 0 and C, it is possible that b is not unique.

The numerical approach in SVM design is to solve the dual (instead of the primal) because it is a finite-dimensional optimization problem. (Note that $w(\alpha) \cdot w(\alpha) = \sum_i \sum_j y_i y_j \alpha_i \alpha_j k(x_i, x_j)$.) 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:

$$\bar{L} = \frac{1}{2} w(\alpha) \cdot w(\alpha) - \sum_i \alpha_i - \sum_i \delta_i \alpha_i + \sum_i \mu_i(\alpha_i - C) - \beta \sum_i \alpha_i y_i$$

Define

$$F_i = w(\alpha) \cdot z_i - y_i = \sum_j \alpha_j y_j k(x_i, x_j) - y_i$$

The KKT conditions for the dual problem are:

$$\frac{\partial \bar{L}}{\partial \alpha_i} = (F_i - \beta) y_i - \delta_i + \mu_i = 0, \;\; \delta_i \geq 0, \;\; \delta_i \alpha_i = 0, \;\; \mu_i \geq 0, \;\; \mu_i(\alpha_i - C) = 0 \;\; \forall\, i$$

These conditions[1] can be simplified by considering three cases for each i.

Case 1. $\alpha_i = 0$

$$\delta_i \geq 0,\ \mu_i = 0 \ \ \Rightarrow (F_i - \beta) y_i \geq 0 \tag{3a}$$

Case 2. $0 < \alpha_i < C$

$$\delta_i = 0,\ \mu_i = 0 \ \ \Rightarrow (F_i - \beta) y_i = 0 \tag{3b}$$

Case 3. $\alpha_i = C$

$$\delta_i = 0,\ \mu_i \geq 0 \ \ \Rightarrow (F_i - \beta) y_i \leq 0 \tag{3c}$$

Define the following index sets at a given α: $I_0 = \{i : 0 < \alpha_i < C\}$; $I_1 = \{i : y_i = 1, \alpha_i = 0\}$; $I_2 = \{i : y_i = -1, \alpha_i = C\}$; $I_3 = \{i : y_i = 1, \alpha_i = C\}$; and, $I_4 = \{i : y_i = -1, \alpha_i = 0\}$. *Note that these index sets depend on* α. The conditions in (3a)-(3c) can be rewritten as

$$\beta \leq F_i \ \forall\, i \in I_0 \cup I_1 \cup I_2\ ; \ \ \beta \geq F_i \ \forall\, i \in I_0 \cup I_3 \cup I_4 \tag{4}$$

It is easily seen that optimality conditions will hold iff there exists a β satisfying (4). The following result is an immediate consequence.

Lemma 1. *Define:*

$$b_{\text{up}} = \min\{F_i : i \in I_0 \cup I_1 \cup I_2\} \ \text{ and } \ b_{\text{low}} = \max\{F_i : i \in I_0 \cup I_3 \cup I_4\} \tag{5}$$

Then optimality conditions will hold at some α *iff*

$$b_{\text{low}} \leq b_{\text{up}} \tag{6}$$

It is easy to see the close relationship between the threshold parameter b in the primal problem and the multiplier, β. *In particular, at optimality,* β *and* b *are identical.* Therefore, in the rest of the paper β and b will denote one and the same quantity.

[1] The KKT conditions are both necessary and sufficient for optimality. Hereafter we will simply refer to them as *optimality conditions*.

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

$$i \in I_0 \cup I_3 \cup I_4\ ,\quad j \in I_0 \cup I_1 \cup I_2 \;\text{ and }\; F_i > F_j \tag{7a}$$

$$i \in I_0 \cup I_1 \cup I_2\ ,\quad j \in I_0 \cup I_3 \cup I_4 \;\text{ and }\; F_i < F_j \tag{7b}$$

Note that optimality conditions 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 (6) can be replaced by

$$b_{\text{low}} \le b_{\text{up}} + 2\tau \tag{8}$$

where τ is a positive tolerance parameter. (In the pseudo-code given in (Platt, 1998) this parameter is referred to as `tol`.) Correspondingly, the definition of violation can be altered by replacing (7a) and (7b) by:

$$i \in I_0 \cup I_3 \cup I_4\ ,\quad j \in I_0 \cup I_1 \cup I_2 \;\text{ and }\; F_i > F_j + 2\tau \tag{9a}$$

$$i \in I_0 \cup I_1 \cup I_2\ ,\quad j \in I_0 \cup I_3 \cup I_4 \;\text{ and }\; F_i < F_j - 2\tau \tag{9b}$$

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

Since β can be placed halfway between b_{low} and b_{up}, approximate optimality conditions will hold iff there exists a β such that (3a)-(3c) are satisfied with a τ-margin, i.e.,

$$(F_i - \beta) y_i \ge -\tau \;\text{ if }\; \alpha_i = 0 \tag{10a}$$

$$|(F_i - \beta)| \le \tau \;\text{ if }\; 0 < \alpha_i < C \tag{10b}$$

$$(F_i - \beta) y_i \le \tau \;\text{ if }\; \alpha_i = C \tag{10c}$$

(10a)-(10c) are the approximate optimality conditions employed by Platt, 1998, Joachims, 1998 and others. Note that, if (10) is violated for one value of β, that does not mean that optimality conditions are violated.

It is also possible to give an alternate approximate optimality condition based on the closeness of $W(\alpha)$ to the optimal value, estimated using duality gap ideas. This is fine, but care is needed in choosing the tolerance used; see Keerthi et.al, 1999a for a related discussion. This criterion has the disadvantage that it is a single global condition involving all i. On the other hand, (9) consists of an individual condition for each pair of indices and hence it is much better suited for the methods discussed in this paper. In particular, when (i, j) satisfies (9) then a strict improvement in the dual objective function can be achieved by optimizing only α_i and α_j. (This is true even if $\tau = 0$.)

3 Platt's SMO Algorithm.

We now give a brief description of the SMO algorithm. The basic step consists of choosing a pair of indices, (i_1, i_2) and optimizing the dual objective function by adjusting α_{i_1} and α_{i_2} only. Because the working set is only of size 2 and the equality constraint in (2) can be used to eliminate one of the two Lagrange multipliers, the optimization problem at each step is a quadratic minimization in just one variable. It is straightforward to write down an analytic solution for it. Complete details are derived in Platt, 1998. The procedure, `takeStep` (which is a part of the pseudocode given there) gives a clear description of the implementation. There is no need to recall all details here. We only make one important comment on the role of the threshold parameter, β. As in Platt, 1998 define the output error on the i-th pattern as

$$E_i = F_i - \beta$$

Consistent with the pseudocode of Platt, 1998 let us call the indices of the two multipliers chosen for optimization in one step as i_2 and i_1. A look at the details in Platt, 1998 shows that to take a step by varying α_{i_1} and α_{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 efficient solution of the problem. Based on a number of experiments Platt came up with a good set of heuristics. He 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 lower boundary (in Platt'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 Platt maintains and updates a cache for E_i values for indices i corresponding to non-boundary 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 objective function, the index i_1 is chosen to maximize $|E_{i_2} - E_{i_1}|$. (If we define $\rho(t) = W(\alpha(t))$ where $\alpha_{i_1}(t) = \alpha_{i_1}^{\text{old}} + y_{i_1}t$, $\alpha_{i_2}(t) = \alpha_{i_2}^{\text{old}} - y_{i_2}t$, and $\alpha_i = \alpha_i^{\text{old}}$ $\forall$ $i \notin \{i_1, i_2\}$, then $|\rho'(0)| = |E_{i_1} - E_{i_2}|$.) Since E_i is available in cache for non-boundary multiplier indices, only such indices 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 indices corresponding to non-bound multipliers are tried as choices for i_1, one by one. If still sufficient progress is not possible, all indices 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 (10a)-(10c) 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 (3) for $i \in \{i_1, i_2\}$. If, after a step involving (i_1, i_2), one of α_{i_1}, α_{i_2} (or both) takes a non-boundary value then (3b) 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: $\beta = (\beta_{\text{low}} + \beta_{\text{up}})/2$.

4 Inefficiency of the SMO Algorithm.

SMO is a carefully organized algorithm which has excellent computational efficiency. However, because of its way of computing and using a single threshold value it can carry some inefficiency unnecessarily. At any instant, the SMO algorithm fixes β based on the current two indices which are being optimized. 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 has reached a value where optimality is satisfied (i.e., (8)), SMO hasn't 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 (10) is employed using an "incorrect" value of β although this index may not be able to pair with another to define a violation. 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 an important 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 (1) of a primal feasible solution is greater than or equal to the objective function value in (2) 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 is given and $w = w(\alpha)$. The 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.[2] This corresponds to the subproblem,

$$\min \sum_i \max\{0, y_i(\beta - F_i)\}$$

[2] This idea was suggested to us by one of the reviewers.

This is an easy problem to solve. Let p denote the number of indices in class 2, i.e., the number of i having $y_i = -1$. In an increasing order arrangement of $\{F_i\}$ let f_p and f_{p+1} be the p-th and $(p+1)$-th values. Then any β in the interval, $[f_p, f_{p+1}]$ is a minimizer. The determination of f_p and f_{p+1} can be done efficiently using a "median finding" technique. Since all F_i are typically not available at a given stage of the algorithm, it is appropriate to apply the above idea to that subset of indices for which F_i are available. (This set contains I_0.) We implemented this idea and tested it on some benchmark problems. It gave a mixed performance, performing well in some situations, performing poorly in some situations, and failing in a few cases. More detailed study is needed to understand the cause of this. See section 6 for details of the performance on three examples.

5 Modifications of the SMO Algorithm.

In this section we suggest two modified versions of the SMO algorithm, each of which overcomes the problems mentioned in the last section. As we will see in the computational evaluation of section 6, these modifications are almost always better than the original SMO algorithm and, in most situations they give quite good improvement in efficiency when tested on several benchmark problems.

In short, the modifications avoid the use of a single threshold value β and the use of (10) for checking optimality. Instead, two threshold parameters, b_{up} and b_{low} are maintained and (8) (or (9)) is employed for checking optimality. Assuming that the reader is familiar with Platt, 1998 and the pseudo-codes given there, we only give a set of pointers that describe the changes that are made to Platt's SMO algorithm. Pseudo-codes that fully describe these changes can be found in Keerthi et.al, 1999b.

1. Suppose, at any instant, F_i is available for all i. Let *i_low* and *i_up*

be indices such that

$$F_{i\_low} = b_{\text{low}} = \max\{F_i : i \in I_0 \cup I_3 \cup I_4\} \tag{11a}$$

$$F_{i\_up} = b_{\text{up}} = \min\{F_i : i \in I_0 \cup I_1 \cup I_2\} \tag{11b}$$

Then checking a particular i for optimality is easy. For example, suppose $i \in I_1 \cup I_2$. We only have to check if $F_i < F_{i\_low} - 2\tau$. 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 indices in the other sets. Thus, in our approach, the checking of optimality of i_2 and the choice of the second index, i_1 go hand in hand, unlike the original SMO algorithm. As we will see below, we compute and use (i_low, b_{low}) and (i_up, b_{up}) via an efficient updating process.

2. To be efficient, we would, like in the SMO algorithm, spend much of the effort altering α_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 examine all indices for optimality.

3. Some extra steps are added to the `takeStep` procedure. After a successful step using a pair of indices, (i_2, i_1), let $\tilde{I} = I_0 \cup \{i_1, i_2\}$. We compute, *partially*, (i_low, b_{low}) and (i_up, b_{up}) using $\tilde{I}$ only (i.e., use only $i \in \tilde{I}$ in (11)). 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, $\tilde{I} \cap (I_0 \cup I_1 \cup I_2)$ and $\tilde{I} \cap (I_0 \cup I_3 \cup I_4)$, is non-empty; hence the partially computed (i_low, b_{low}) and (i_up, b_{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 only with α_i, $i \in I_0$, i.e., a loop with `examineAll=0`, one should note that, if (8) 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 $b_{\rm low}$ and $b_{\rm up}$ are influenced by all indices in I_0.) This gives an easy way of exiting this loop.

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

Method 1. This is in line with 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 $F_{i_2} < F_{i\_low} - 2\tau$ then there is a violation, and, in that case choose $i_1 = i\_low$.

Method 2. Always work with the worst violating pair, i.e., choose $i_2 = i\_low$ and $i_1 = i\_up$.

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. 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 indices (`examineAll=1`). Here we loop through all indices, one by one. Since $(b_{\rm low}, i\_low)$ and $(b_{\rm up}, i\_up)$ 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 $(b_{\rm low}, i\_low)$ and $(b_{\rm up}, i\_up)$; if there is no violation, F_i is used to update these quantities. For example, if $i \in I_1 \cup I_2$ and $F_i < b_{\rm low} - 2\tau$, 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 $(i\_up, b_{\rm up})$ is modified using F_i, i.e, if $F_i < b_{\rm up}$ then we do: $i\_up := i$ and $b_{\rm up} := F_i$.

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 is: *YES*. 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 (9). Let us say, $i < j$. Then j would not have satisfied the optimality check in the above described implementation because F_i would have, earlier than j is seen, affected either the calculation of b_{low} and/or b_{up} 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 analysed. Only when (8) holds it is possible for all indices to satisfy the optimality checks. Furthermore, when (8) holds and the loop over all indices has been completed, the true values of b_{up} and b_{low}, as defined in (5) would have been computed since all indices have been encountered.

6 Computational Comparison.

In this section we compare the performance of our modifications against the original SMO algorithm and the SMO algorithm which uses duality gap ideas for optimizing β, as explained in section 4. We implemented all these methods in Fortran and ran them using f77 on a 200 MHz Pentium machine. The value, $\tau = 0.001$ was used for all experiments. Although we have tested on a number of problems, here we only report the performance on the following three problems: Wisconsin Breast Cancer data (Bennett and Mangasarian, 1992); UCI Adult data (Platt, 1998); and Web page classification data (Joachims, 1998, Platt, 1998). In the case of Adult and Web data sets, the inputs are represented in a special binary format, as used by Platt in his testing of SMO. To study scaling properties as training data grows, Platt did staged experiments on the Adult and Web data. We have used only the data from the seventh stage. The gaussian kernel, $k(x_i, x_j) = \exp(-0.5\|x_i - x_j\|^2/\sigma^2)$ was used in all experiments. The σ^2 values employed, together with n, the dimension of the input, and m, the number of training points, are given in Table 1.

When a particular method is used for SVM design, the value of C is

Data Set	σ^2	n	m
Wisconsin Breast Cancer	4.0	9	683
Adult-7	10.0	123	16100
Web-7	10.0	300	24692

Table 1: Data Set Properties.

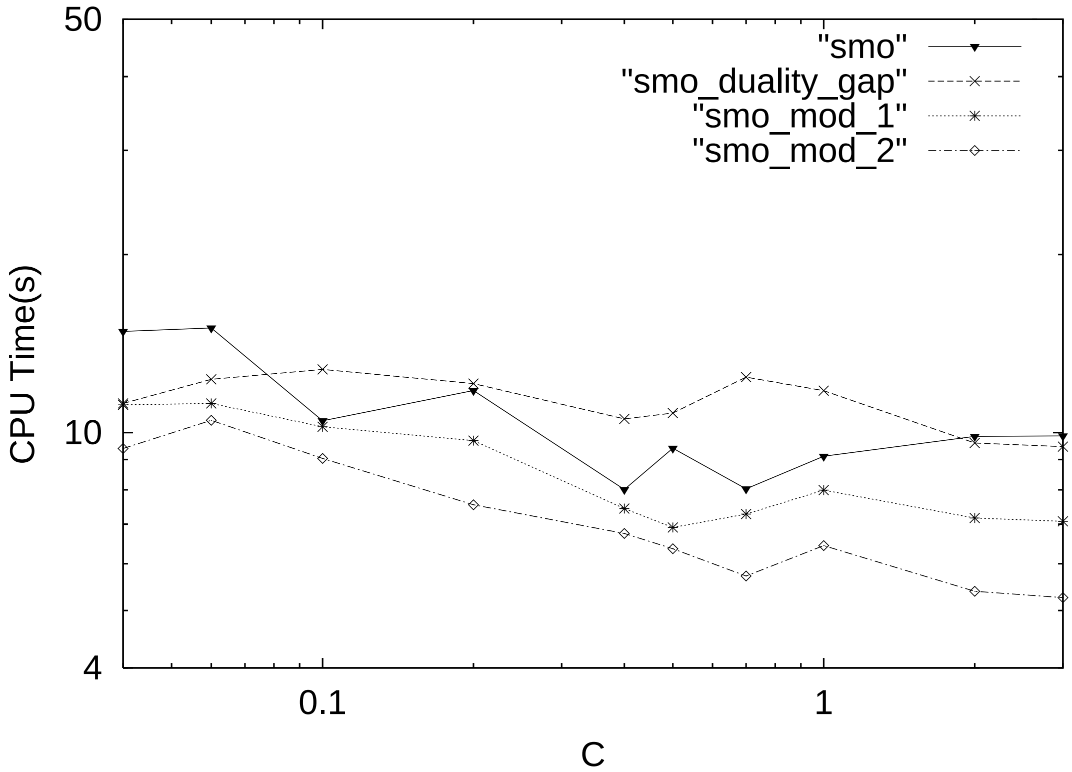

Figure 1: Wisconsin Breast Cancer data: CPU Time (in seconds) shown as a function of C.

usually unknown, and it has to be chosen by trying a number of values and using a validation set. Therefore, good performance of a method over a range of C values is important. Therefore for each problem we have tested the algorithms over an appropriate range of C values. Figures 1-3 give the computational costs for the two problems. For the Adult data set the SMO algorithm based on duality gap ideas had to be terminated for extreme C values since excessive computing times were needed. In the case of Web data this method failed for C=0.2.

It is very clear that the modifications outperform the original SMO algorithm. In most situations the improvement in efficiency is very good. Between

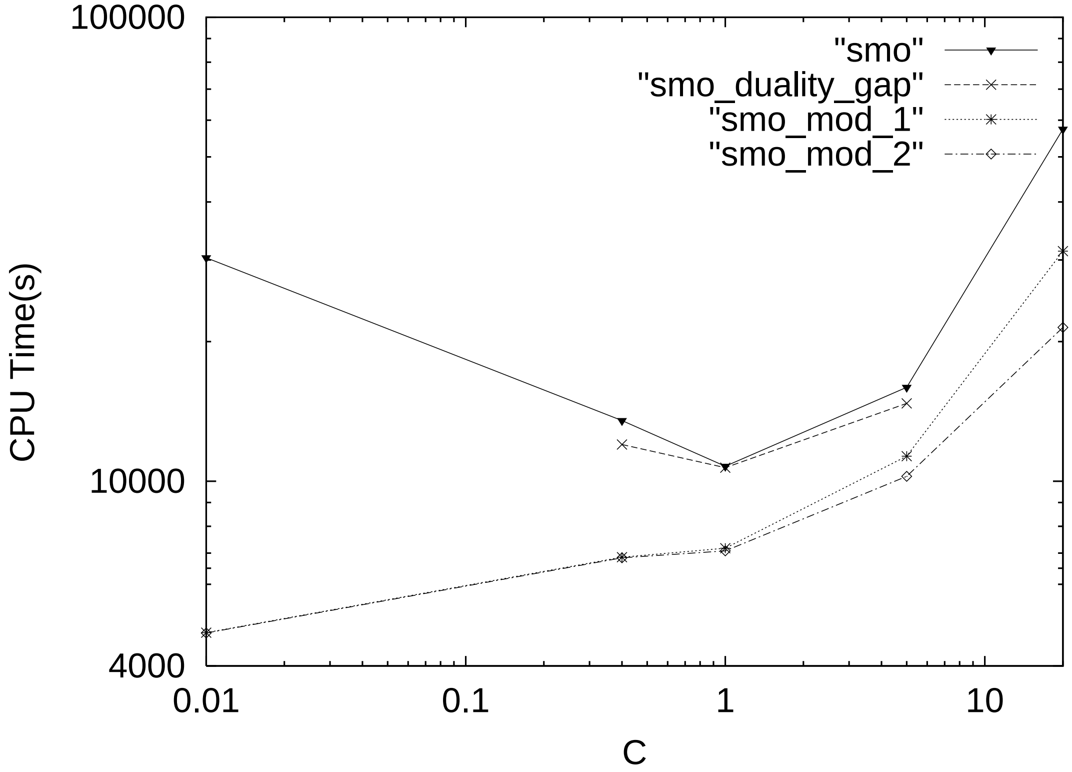

Figure 2: Adult-7 data: CPU Time (in seconds) shown as a function of C.

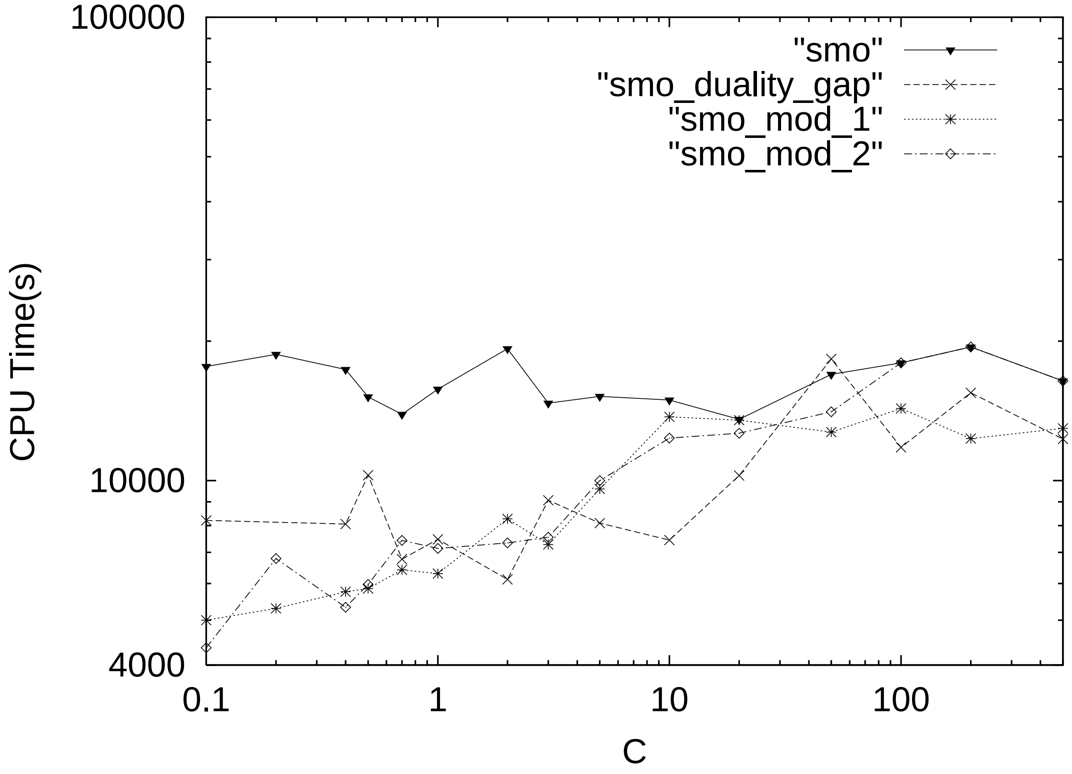

Figure 3: Web-7 data: CPU Time (in seconds) shown as a function of C.

the two modifications, the second one fares better overall. The SMO algorithm based on duality gap did not fare well overall. While in some cases it did well, in other cases it became slower than the original SMO; as already mentioned, it also failed in some cases.

7 Conclusion.

In this paper we have pointed out an important source of inefficiency in Platt'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 considerably in many situations. Platt has already established the SMO algorithm to be one of the fastest algorithms for SVM design. The modified versions of SMO presented in this paper enhance the value of the SMO algorithm even further. It should also be mentioned that heuristics such as *shrinking* and *kernel caching* that are effectively used in Joachims, 1998 can be employed to further speed-up our SMO modifications. The ideas mentioned in this paper for SVM classification can also be extended to the SMO regression algorithm (Smola, 1998). We have reported those results in Shevade et.al, 1999.

References

R. Bennett and O.L. Mangasarian, (1992). Robust linear programming discrimination of two linearly inseparable sets, *Optimization Methods and Software*, Vol.1, pp.23-34.

C.J.C. Burges, (1998). A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery*, Vol.2, Number 2.

T.T. Friess, (1998). Support vector networks: The kernel adatron with bias and soft-margin, Tech. Report, The University of Sheffield, Dept. of Automatic

Control and Systems Engineering, Sheffield, England.

T. Joachims, (1998). Making large-scale support vector machine learning practical, in B. Schölkopf, C. Burges, A. Smola. *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA.

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya and K.R.K. Murthy, (1999a). A fast iterative nearest point algorithm for support vector machine classifier design, Technical Report TR-ISL-99-03, Intelligent Systems Lab, Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, India; see *http://guppy.mpe.nus.edu.sg/˜mpessk*

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya and K.R.K. Murthy, (1999b). Improvements to Platt's SMO algorithm for SVM classifier design, Tech. Report CD-99-14, Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore; see *http://guppy.mpe.nus.edu.sg/˜mpessk*

O.L. Mangasarian and D.R. Musicant, (1998). Successive overrelaxation for support vector machines, Tech. Report, Computer Sciences Dept., University of Wisconsin, Madison, WI, USA.

J.C. Platt, (1998a). Fast training of support vector machines using sequential minimal optimization, in B. Schölkopf, C. Burges, A. Smola. *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA.

J.C. Platt, (1999). Using sparseness and analytic QP to speed training of support vector machines, in *Advances in Neural Information Processing Systems 11*, M.S. Kearns, S.A. Solla and D.A. Cohn, eds., MIT Press, 1999.

S.K. Shevade, S.S. Keerthi, C. Bhattacharyya and K.R.K. Murthy, (1999). Improvements to the SMO algorithm for SVM regression, Tech. Rept. CD-99-16, Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore, Singapore; see *http://guppy.mpe.nus.edu.sg/˜mpessk*

A.J. Smola, (1998). *Learning with Kernels.* PhD thesis, Technische Universität Berlin.

A.J. Smola and B. Schölkopf, (1998). A tutorial on support vector regression, *NeuroCOLT Technical Report TR-1998-030*, Royal Holloway College, London, UK, 1998.

V. Vapnik, (1982). *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, Berlin.

V. Vapnik, (1995). *The Nature of Statistical Learning Theory.* Springer-Verlag, New York.