Faster Directions for Second Order SMO

Álvaro Barbero ⋆ and José R. Dorronsoro⋆

Universidad Autómoma de Madrid and Instituto de Ingeniería del Conocimiento
Francisco Tomás y Valiente 11, 28049, Madrid, Spain
{alvaro.barbero@,jose.dorronsoro@}uam.es

Abstract. Second order SMO represents the state–of–the–art in SVM training for moderate size problems. In it, the solution is attained by solving a series of subproblems which are optimized w.r.t just a pair of multipliers. In this paper we will illustrate how SMO works in a two stage fashion, setting first the values of the bounded multipliers to the penalty factor C and proceeding then to adjust the non–bounded multipliers. Furthermore, during this second stage the selected pairs for update often appear repeatedly during the algorithm. Taking advantage of this, we shall propose a procedure to combine previously used descent directions that results in much fewer iterations in this second stage and that may also lead to noticeable savings in kernel operations.

1 Introduction

Given a training sample $\mathcal{S} = \{(X_i, y_i) : i = 1, \ldots, N\}$ with $y_i = \pm 1$, SVM training seeks [1] to find a separating hyperplane in the form $W \cdot X + b$ with maximal margin by solving the dual problem

$$\min_\alpha f(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e \cdot \alpha \qquad s.t. \begin{cases} 0 \le \alpha \le C \\ \alpha \cdot y = 0 \end{cases} \tag{1}$$

where $Q = (Q_{ij})$ with $Q_{ij} = y_i y_j X_i \cdot X_j$, e is an all-ones vector, α^T denotes the transpose of α, $\cdot$ indicates the standard dot product and C is a penalty parameter. Once the problem is solved, the primal problem solution can be obtained as well using $W = \sum_i \alpha_i y_i X_i$ and computing b though the Karush–Kuhn–Tucker optimality conditions [1]. At first sight, this problem is a relatively simple constrained quadratic minimization problem and, as such, easy to solve. However, $\dim(\alpha) = N$ and so we may not be able to store the full matrix Q into memory, even for moderate size problems. Furthermore, non-linearity is usually introduced in the SVM by using the kernel trick as $Q_{ij} = y_i y_j K(X_i, X_j)$, making the entries of Q costly to evaluate, as non-linear Kernel Operations (KOs) are required. These conditions make impossible to apply standard and fast inner point solvers to the problem. The solution to this are decomposition

⋆ All authors have been partially supported by Spain's TIN 2007–66862 and "Cátedra IIC Modelado y Prediccion". The first author is kindly supported by the FPU–MEC grant reference AP2006–02285.

methods, where iteratively a series of subproblems are solved, each of them involving only a small number q of the multipliers. Among the most effective decomposition procedures is Joachims' SVM–Light [2] where the subproblem multipliers are chosen using gradient information. For $q = 2$ SVM–Light reduces to Sequential Minimal Optimization (SMO) [3], that iteratively changes α to $\alpha' = \alpha + \delta(e_U - y_U y_L e_L)$ for appropriate L, U and δ (see below).

However, decomposition methods are not problem free, because as we will see, the gradient of $f(\alpha)$ needs to be updated at every iteration, which requires at least $N \times q$ KOs. Therefore, if the number of iterations does not decrease substantially, the cost in KOs of a q–multiplier procedure may degrade as q grows [4]. The SMO method has a cost of $2N$ KOs per iteration and benefits from the ability of solving its corresponding subproblems in closed form. Therefore, as implemented for instance in the LIBSVM package [5] (also known as second order SMO) is often the most efficient choice, at least for moderate size problems.

In the experimental use of SMO there are two folk observations. The first one is that the initial iterations of second order SMO concentrate predominantly on the bounded multipliers, i.e., those α_i for which at the optimum $\alpha_i^* = C$, as their number increases until it becomes stable. Then SMO focuses on the unbounded multipliers, which are adjusted to arrive at their optimal values $0 < \alpha_i^* < C$. The decrease of $f(\alpha)$ is very fast in the first phase but much slower in the second one. The second observation is that there are often several pairs that are selected repeatedly, particularly as SMO training advances. These observations suggest that, in order to improve the speed of SMO, one should concentrate in this second stage and try to exploit the repeated pairs to derive better descent directions.

In this paper we present an improvement over SMO, which constructs accelerating directions much in the way the Hooke–Jeeves (H–J) method improves cyclic coordinate descent [6], therefore allowing for updating directions unavailable to standard SMO while keeping the burden in KOs under control. Whereas in H–J an accelerating direction is built after a fixed number of iterations, here we will attempt to do so each time an updating pair of multipliers reappears during the optimization process. We shall briefly review first and second order SMO in section 2 and in section 3 we will give the details of our accelerated version. Both second order SMO procedures are compared in section 4 and the paper ends with a short discussion.

2 First and Second Order SMO

In principle, the SMO updates would be of the form $\alpha' = \alpha + \delta_U e_U + \delta_L e_L$, where e_i is an all-zeros vector except for the $i-th$ component which is valued 1, i.e. only two coefficients are allowed to change. However the constraint $y \cdot \alpha = 0$ implies that $\delta_L y_L + \delta_U y_U = 0$; that is, $\delta_L = -\delta_U y_L y_U$. Therefore, the SMO updates become $\alpha' = \alpha + \delta(e_U - y_U y_L e_L)$ where we write δ instead of δ_U. Note that this update can be thought as performing a step of size δ in the direction $d_{U,L} = (e_U - y_U y_L e_L)$. As a consequence of this and if we ignore the problem's constraints for the moment, by solving $\frac{\partial}{\partial\delta} f(\alpha + \delta d_{U,L}) = 0$ we obtain optimal an step δ^* as

$$\delta^* = \frac{d_{U,L} \cdot (\alpha - Q\alpha)}{d_{U,L}^T Q d_{U,L}} = \frac{-d_{U,L} \cdot \nabla f(\alpha)}{d_{U,L}^T Q d_{U,L}} = -y_U \frac{\Delta_{U,L}}{Z_{U,L}} = -y_U \lambda^* \tag{2}$$

where $\nabla f(\alpha)$ stands for the gradient of $f(\alpha)$ and we write $\Delta_{U,L} = y_U \nabla f(\alpha)_U - y_L \nabla f(\alpha)_L$, $Z_{U,L} = d_{U,L}^T Q d_{U,L} = K(X_U, X_U) + K(X_L, X_L) - 2K(X_L, X_U)$ and $\lambda^* = \Delta_{U,L}/Z_{U,L}$. The corresponding multiplier updates can be expressed as $\alpha'_L = \alpha_L + y_L \lambda^*$, $\alpha'_U = \alpha_U - y_U \lambda^*$ and $\alpha'_j = \alpha_j$ for any other j.

Several proposals can be found in the literature regarding how to choose the updating pair (L, U). In the so-called first order SMO, also known as Modification 2 [3], L and U are chosen as the pair that most violates at α the Karush–Kuhn–Tucker optimality conditions, i.e.

$$\begin{aligned} L = \text{arg min}_j \{y_j \nabla f(\alpha)_j : j \in I_L\}, \quad U = \text{arg max}_j \{y_j \nabla f(\alpha)_j : j \in I_U\}, \\ I_L = \{j | (y_j = -1, \alpha_j > 0) \text{ or } (y_j = 1, \alpha_j < C)\}, \\ I_U = \{j | (y_j = -1, \alpha_j < C) \text{ or } (y_j = 1, \alpha_j > 0)\}. \end{aligned} \tag{3}$$

Notice that this choice implies $\Delta_{U,L} > 0$ and $\lambda^* > 0$, and the restrictions on the α multipliers are needed so that we have $0 \leq \alpha'_L \leq C$, $0 \leq \alpha'_U \leq C$. Moreover, the initial λ^* value given by (2) may have to be clipped down so that these bounds hold.

Note also that, since $\Delta_{U,L} = y_U \nabla f(\alpha)_U - y_L \nabla f(\alpha)_L$, we have $\Delta_{U,L} \geq \Delta_{j,i}$ for any other feasible direction $d_{j,i}$, and it follows that the most violating pair U, L choice also gives the feasible direction more aligned with $\nabla f(\alpha)$. In other words, $d_{U,L}$ is the best first order feasible descent direction. However, if no clipping is needed for λ^*, is is easy to see [7] that the gain can also be written as $f(\alpha) - f(\alpha') = \frac{\Delta_{U,L}^2}{Z_{U,L}}$. This suggests a second order choice of L, U (see [7] for more details) as the pair for which the full gain is maximal. To avoid a nested loop on L and U, one chooses first L as in (3) and then U is selected as

$$U = \text{arg max}_j \left\{ \frac{\Delta_{j,L}^2}{Z_{U,L}} : j \in I_U, \Delta_{j,L} > 0 \right\}. \tag{4}$$

These second order index choices result in a much faster convergence of SMO and are implemented, for instance, in the latest versions of LIBSVM [5], the SVM training tool that can be considered representative of the current state–of–the–art.

To close this section we remark that we have to keep track of the gradient in order to compute the optimal L, U indices at each. After an α update we can also update the gradient efficiently using

$$\nabla f(\alpha + \delta d) = \nabla f(\alpha) + \delta Q d = \nabla f(\alpha) + \delta(Q_U - y_U y_L Q_L),$$

where Q_i stands for the i-th column of Q. It follows that SMO requires essentially $2N$ KOs at each iteration.

3 Better Directions for Second Order SMO

In this section we will propose a way to improve SMO convergence speed by combining recently used descent directions. Let $d^t = d_{L_t,U_t}$ denote a certain update vector that we assume has appeared again after K steps from a former use as the update vector d^{t-K}; in other words, we are assuming that $L_t = L_{t-K}, U_t = U_{t-K}$. Consider then $v = \sum_{j=1}^{K} \delta_{t-j} d^{t-j}$, with δ_{t-j} the optimal updating coefficient at step $t-j$.

Clearly, v would have been a better descent direction at α^{t-K} than d^{t-K}, as we would have arrived to α^t (or another α' such that $f(\alpha') < f(\alpha^t)$) in just one iteration, i.e., it is able to provide greater decrease in $f(\alpha)$. Thus, it makes sense to consider using v as a descent direction at α^T alternative to d^t, as it might still be a better direction; to choose the best option we have to compare the $\partial_1 = -\nabla f(\alpha^t) \cdot d^t$ and $\partial_2 = -\nabla f(\alpha^t) \cdot v$ values, i.e., the directional derivatives at α^t with respect d^t and v respectively, and decide on the most negative one. That is, we will be choosing the direction with largest negative steep. The computation of ∂_1 is straightforward, as $\partial_1 = \Delta_{U,L}$, and that of ∂_2 is only slightly more complex. In fact, we have

$$\begin{aligned}\nabla f(\alpha^t) \cdot v &= \sum_{j=1}^{K} \delta_{t-j} \nabla f(\alpha^t) \cdot d^{T-j} \\ &= \sum_{j=1}^{K} \delta_{t-j} (\nabla f(\alpha^t)_U - y_{U_{t-j}} y_{L_{t-j}} \nabla f(\alpha^t)_L)\end{aligned}$$

which can be computed without needing any KO. Once we have decided to perform an update in the v direction, observe that the value of the objective function will change as $f(\alpha + \lambda v) = f(\alpha) + \frac{1}{2}\lambda^2 v^T Q v + \delta v^T Q \alpha - \delta v \cdot \alpha$, and so the optimal stepsize ignoring constraints can be obtained as

$$\lambda^o = \frac{-v \cdot \nabla f(\alpha)}{v^T Q v} = \frac{-\partial_2}{v^T Q v}. \tag{5}$$

As v is sparse by construction, at most $2K$ entries are non-zero, and so $v^T Q v$ can be computed efficiently by defining $R_i = Q_i v = \sum_{v_j \neq 0} Q_{ij} v_j$ and using $v^T Q v = \sum_{v_j \neq 0} v_j R_j$, which requires at most $2KN$ kernel operations. Furthermore the vector R can be used to update the gradient as $\nabla f(\alpha + \lambda v) = \nabla f(\alpha) + \lambda R$.

Now, taking the constraints back into account note that by using v as updating direction the α will be modified as $\alpha_i^{t+1} = \alpha_i^t + \lambda v_i \ \forall \ v_i \neq 0$. Therefore we must have $0 \leq \alpha_i^{t+1} = \alpha_i^t + \lambda v_i \leq C$. Thus, if $v_i > 0$, the relevant bound is the right one, while the left one has to be met when $v_i < 0$. Define $M_C = \min\{(C - \alpha_i^t)/v_i : v_i > 0\}$ and $M_0 = \min\{-\alpha_i^t/v_i : v_i < 0\}$. By clipping λ^o from above as $\lambda^* = \min\{\lambda^o, M_0, M_C\}$ we guarantee feasibility.

Finally, to detect a repetition of the L_t, U_t indices, we keep them in a circular queue $\mathcal{Q}$ which is searched from its beginning each time a new pair L, U is selected. If the search fails we insert the pair in $\mathcal{Q}$, but if a previous copy

(L_{t-K}, U_{t-K}) is found we check, as mentioned before, whether the updating vector V actually defines a descent direction with larger steep than the standard updating direction. If it does so, we will perform a v update and reset $\mathcal{Q}$ afterwards. Conversely, if it does not, we will remove the previous appearance (L_{t-K}, U_{t-K}) from $\mathcal{Q}$ and, in order to keep the temporal structure of the (L, U) index pairs in $\mathcal{Q}$, we will also remove all pairs from $\mathcal{Q}$'s front up to the (L_{t-K}, U_{t-K}) position. All in all, the overall cost of a v update can be regarded as $O(2KN)$ KOs plus other non-KOs operations involving the queue management, which results in roughly K times the cost of standard SMO. Therefore a global speed-up will only happen if the total number iterations the algorithm requires to achieve convergence is sufficiently reduced to make up for the additional costs for these accelerating iterations. An outline of the algorithm is presented in 1.

4 Numerical Experiments

In this section we will compare the performance of standard second order SMO (SO) and of our accelerated procedure (AccSO) on the datasets taken from G. Rätsch's benchmark repository [8]. Unless otherwise stated, we shall always use Gaussian kernels with parameter values C and $2\sigma^2$ as reported in [8]. The stopping criterion will be that the maximum KKT violating value Δ be smaller than a tolerance $\epsilon = 10^{-5}$ and the initial α multipliers values are 0.

First we will briefly illustrate the two–stage nature of SO. Figure 1 shows the evolution of the number of multipliers at the C bound (upper bounded) and of unbounded multipliers $(0 < \alpha < C)$ for each of the datasets when SO is applied. A general trend among datasets can be noticed, in which during a first phase of the algorithm most of the updates are displacing multipliers to the upper bound, while a lesser quantity get moved to an unbounded state. Next,

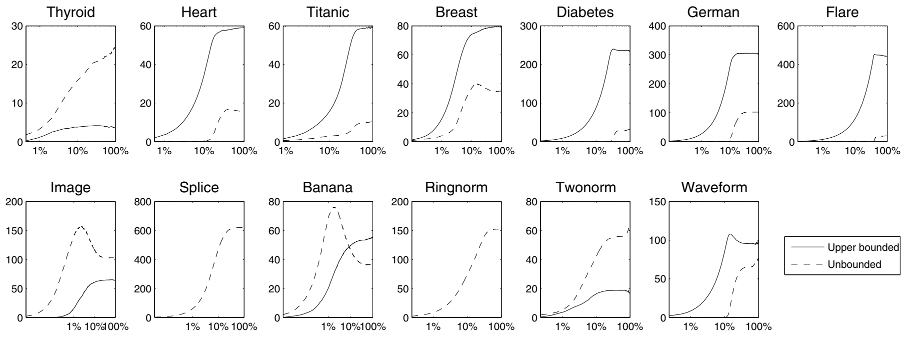


Fig. 1. Evolution of the number of bounded and unbounded α coefficients for every dataset. The x-axis represents the percentage of iterations performed by the algorithm (in logarithmic scale), while the y-axis stands for the number of upper bounded or unbounded coefficients.

Algorithm 1. Accelerated SMO

```
initialize $\alpha = 0$, $\nabla f(\alpha) = 0p$, $\mathcal{Q} = \varnothing$ ;
while (stopping condition == FALSE) do
  find $(L, U)$ second (4) order SMO rules ;
  if pair $(L, U)$ is found in $\mathcal{Q}$ then
    build accelerating direction $v$ ;
    if $v$ is feasible and $\partial_2 < \partial_1$ then
      compute $R$, optimal unbounded stepsize $\lambda^o$ using (5) ;
      clip $\lambda^o$ to meet constraints $\rightarrow \lambda^*$ ;
      $\alpha = \alpha + \lambda^* v$, $\nabla f(\alpha) = \nabla f(\alpha) + \lambda^* R$, $\mathcal{Q} = \varnothing$ ;
    else
      remove $(L, U)$ and previous updates from $\mathcal{Q}$ ;
      perform standard SMO update using $(L, U)$, add $(L, U)$ to $\mathcal{Q}$ ;
    end if
  else
    perform standard SMO update using $(L, U)$, add $(L, U)$ to $\mathcal{Q}$ ;
  end if
end while
```

the number of upper bounded and unbounded multipliers becomes stable, and only slight changes in their numbers are made until the end of the algorithm. Notice also that some datasets differ from this behaviour. In the case of Heart, Diabetes and Flare datasets unbounded multipliers are only generated after a number of iterations have been completed. On the other hand, in Splice and Ringnorm datasets no upper bounded coefficients appear at all. As we will later see, these datasets present no improvement under our procedure.

While we cannot give a rigorous argument for the generalized two–phase regime, notice that at the early stages of SMO any pair L, U would be eligible and the gain will be large when we have $X_L \simeq X_U$ but $y_L \neq y_U$, as $Z_{L,U} \simeq 0$ and $\Delta_{L,U} = y_U \nabla f(\alpha)_U - y_L \nabla f(\alpha)_L = W \cdot X_U - y_U - (W \cdot X_L - y_L) \simeq -2y_U$. If this is the case, either X_L or X_U will not be correctly classified and the corresponding multiplier will be set to C and it is likely that it will stay there. On the other hand, if we set $\mathcal{O}(\alpha) = \{i : \alpha_i = 0\}$ and $\mathcal{C}(\alpha) = \{i : \alpha_i = C\}$, it can be shown [9] that for a large enough t_0 we will have $\mathcal{O}(\alpha^t) \subset \mathcal{O}(\alpha^*)$ and $\mathcal{C}(\alpha^t) \subset \mathcal{C}(\alpha^*)$ for $t \geq t_0$, with α^* the optimal multiplier vector. Thus, the stabilization of the number of 0 and C bounded multipliers is to be expected (this is also the reason why shrinking works).

Turning now our attention to our method, before performing any comparisons we should note that this method introduces a new parameter τ into the SVM training, which stands for the maximum length of the circular queue $\mathcal{Q}$. Small values of τ might overlook some (L, U) pair repetitions, while large τ values might detect lengthy, spurious cycles which provide small improvement at a high computational cost. To analyse the influence of this parameter we run our method for a range of τ values from 1 to 100 and measure the percentage of reduction in KOs when compared against the standard second order procedure, computed as $p = 100\frac{KOs(AccSO)}{KOs(SO)}$. Results are plotted in figure 2 for all

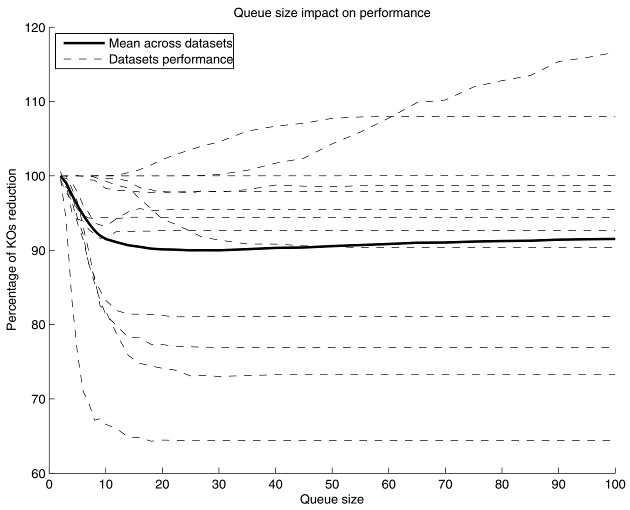


Fig. 2. Percentage of reduction achieved as a function of the queue size. Reductions for each dataset are plotted as dashed lines, while the solid line stands for an average reduction across datasets.

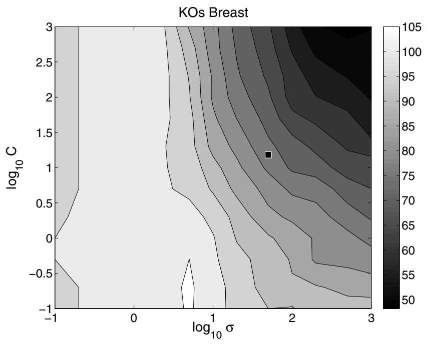
the datasets, along a global reduction value averaged across datasets. It can be observed that in most of the datasets AccSO obtains an improvement in the number of KOs, although there is no reduction or even a worsening in performance in some cases. Note also that on average a good choice of τ seems to be any value in the interval [20 – 35]. However, it should be pointed out that the

Table 1. Average and std. deviation values of the number of KOs (in thousands) and execution times (in milliseconds) by our second order SMO code (SO) and its accelerated version (AccSO) ; the reduction in % also is given.

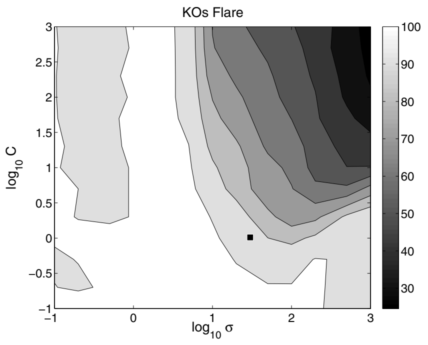
	KOs			Running time		
Dataset	SO	AccSO	Red.	SO	AccSO	Red.
Banana	13576 ± 8377	8751 ± 4813	64,5 √	456,3 ± 246,88	306,74 ± 146,09	67,22 √
Image	56304 ± 8776	41728 ± 5336	74,1 √	1666,87 ± 258,06	1605,91 ± 205,26	96,34 √
Breast	703 ± 270	543 ± 181	77,3 √	27,87 ± 10,7	20,76 ± 6,89	74,49 √
Heart	131 ± 36	106 ± 23	81,2 √	5,39 ± 1,53	4,31 ± 0,94	79,96 √
Flare	1130 ± 594	1047 ± 420	92,6 √	49,18 ± 26,58	46,51 ± 19,22	94,57 √
German	2595 ± 269	2444 ± 229	94,2 √	107,23 ± 16,8	106,03 ± 14,33	98,88 √
Titanic	50 ± 9	48 ± 7	94,4 √	1,89 ± 0,35	1,8 ± 0,28	95,24 √
Thyroid	64 ± 20	61 ± 18	95,4 √	2,28 ± 0,74	2,15 ± 0,65	94,3 √
Twonorm	365 ± 49	356 ± 46	97,7 √	15,15 ± 3,16	14,37 ± 3	94,85 √
Diabetes	451 ± 58	441 ± 52	97,9 √	18,27 ± 2,56	17,72 ± 2,2	96,99 √
Splice	9613 ± 399	9613 ± 399	100,0 ≈	312,37 ± 37,86	308,01 ± 34,34	98,6 √
Ringnorm	487 ± 41	487 ± 41	100,0 ≈	21,46 ± 3,38	21,57 ± 3,41	100,51 ×
Waveform	340 ± 39	347 ± 39	102,2 ×	13,7 ± 2,6	13,9 ± 2,59	101,46 ×

queue management also implies a computational burden scaling proportionally to the queue size. This extra cost, although small in comparison to the cost of computing KOs, cannot be neglected. Hence, we shall use a value of $\tau = 20$ for the rest of our experiments.

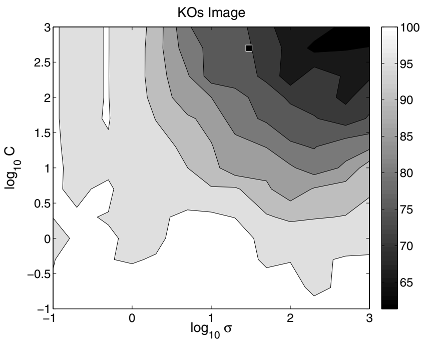
Table 1 shows the detailed results in KOs for that selection of τ: the table's datasets are sorted with respect to the percentage of reduction achieved. Additionally, a $\surd$ symbol denotes a significant improvement in a Wilcoxon rank–sum test at a 10% level, whereas $\times$ stands for significant worsening and $\approx$ for no significant difference. AccSO requires more KOs for the Waveform dataset, ties with SO over Splice and Ringnorm and wins in the other ten datasets. We can thus conclude that AccSO may lead to sizeable savings in KOs when compared with SO and, most likely, with the state–of–the–art SMO packages. Moreover, when this is not the case, AccSO does not seem to add such a great complexity burden as to discourage its use. Additionally, also in table 1 we provide the corresponding execution times, where we can check that the amount of reduction is roughly the same as the observed in KOs for most of the datasets.



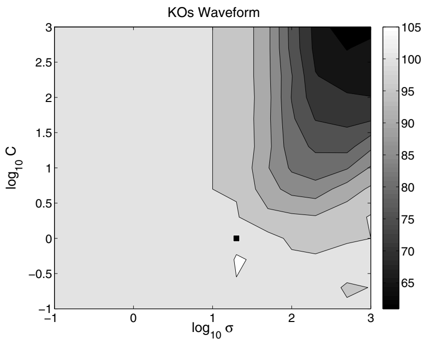
(a)



(b)



(c)



(d)

Fig. 3. Percentage of KOs reduction in AccSO for different settings of C and σ parameters. The squared dots represent the values recommended in [8] for the dataset.

Finally, it is of interest to test whether different values for the SVM parameters C and σ would provide a different degree of improvement. To do this we measure the performance of SO and AccSO for a grid of C and σ values in the range $[0.1, 1000]$. We depict the percentage of reduction achieved for this range of values as contour maps in figure 3 for the datasets Breast, Flare, Image and Waveform, the rest of the datasets showing similar behaviours. It can be observed that the degree of reduction achieved by the method depends heavily on the SVM parameters, the best results being obtained when both of them have large values. Note that C and σ are normally selected through a cross–validation procedure, and so their values will depend on the problem at hand. So, AccSO might be able to provide larger improvements in performance depending on the dataset. On the other hand, note that for most of the parameter space either a notable reduction or no reduction at all is obtained. Worsenings only appear in small areas. Therefore, it is advisable to apply AccSO over SO regardless of the dataset, as generally no increase in computational cost will take place. Also, due to these same reasons, the method could be specially useful to improve running times of a cross-validation procedure that requires training the SVM for a large number of points in the parameter space.

5 Discussion and Conclusions

While decomposition methods for SVM training result in less iterations as the size of the working set grows, this does not translate automatically in a smaller number of kernel operations (KOs), that in fact may increase for larger working sets. The practical consequence of this is that second order SMO, as implemented for instance in the LIBSVM packages, is often the best option to build SVMs on problems with moderately large sample sizes. In any case, as training advances, the convergence speed of second order SMO decreases, something that is usually accompanied by the repeated appearance of some descent directions.

In this work we have numerically shown how there is a further speed gain in second order SMO if its standard descent directions are replaced, when appropriate, by the combination of the successive descent steps between two appearances of a repeated index pair. We thus arrive at a simple procedure to accelerate second order SMO training.

The question that remains is the reason for this faster convergence. While we do not have a full answer at this moment, there are some facts that may partially explain why this is so. As pointed out above, SMO uses the $d_{L,U} = e_U - y_U y_L e_L$ vectors as descent directions. Thus, it can be seen as a kind of coordinate descent on the $d_{i,j}$ meta–coordinate system. If we define the $N - 1$ vectors $\chi_i = d_{i,1}$, $2 \leq i \leq N$, they are clearly linearly independent and we have $d_{L,1} = \chi_L$ and $d_{L,U} = \chi_L - \chi_U$. Thus the subspace spanned by the $d_{i,j}$ directions used in SMO is at most $N - 1$ dimensional but if we center our attention on SMO's second training phase, the subspace dimension would be much smaller, as we replace N by the number of unbounded support vectors.

On the coordinates associated to these directions, first order SMO can be seen as a kind of Gauss–Southwell (GS) minimization method, as the $d_{L,U}$

coordinate chosen is precisely the one associated to the largest negative gradient component. Second order SMO becomes then an improved GS variant. The GS method is essentially an improvement on basic cyclic coordinate descent, where one sequentially explores the coordinate descent directions (see [6], chapter 8). A frequent observation on these methods is that their simple descent directions can be sequentially combined to obtain a new direction that leads to a faster convergence. Examples of this are the acceleration step for cyclic coordinate descent or the Hooke–Jeeves (H–J) method [6] that for a D–dimensional space combines D standard coordinate descent steps with a single step on a certain combination of previously taken directions. As such, it cannot be applied in an SMO setting, as the dimension D might be too large, but our method detects direction cycles and combines the previously taken directions in a way not too far away from those in the H–J algorithm and, as it is the case with H–J, that leads to a convergence speed up.

In any case, further work is required to obtain insights into the method, that may also suggest other ways to improve second order SMO performance.

References

1. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond. In: Machine Learning. MIT Press, Cambridge (2002)
2. Joachims, T.: Making Large-Scale Support Vector Machine Learning Practical. In: Advances in Kernel Methods: Support Vector Learning, pp. 169–184. MIT Press, Cambridge (1999)
3. Keerthi, S.S., Shevade, S.K., Bhattacharyya, C., Murthy, K.R.K.: Improvements to Platt's SMO Algorithm for SVM Classifier Design. Neural Computation 13(3), 637–649 (2001)
4. Collobert, R., Bengio, S.: Svmtorch: Support vector machines for large-scale regression problems. Journal of Machine Learning Research 1, 143–160 (2001)
5. Chang, C.C., Lin, C.J.: LIBSVM: a Library for Support Vector Machines (2001)
6. Bazaraa, M., Sherali, D., Shetty, C.: Nonlinear Programming: Theory and Algorithms. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, Chichester (1992)
7. Fan, R.E., Chen, P.H., Lin, C.J.: Working Set Selection using Second Order Information for Training Support Vector Machines. Journal of Machine Learning Research 6, 1889–1918 (2005)
8. Rätsch, G.: Benchmark Repository (2000), Datasets available at `http://ida.first.fhg.de/projects/bench/benchmarks.htm`
9. Stefano, L., Laura, P., Risi, A., Sciandrone, M.: A convergent hybrid decomposition algorithm model for svm training. IEEE Transactions on Neural Networks 20(6), 1055–1060 (2009)