

A Fast Method for Training Linear SVM in the Primal

Trinh-Minh-Tri Do and Thierry Artières

LIP6 - Université Pierre et Marie Curie
104 avenue du président Kennedy, 75016 Paris France
Trinh-Minh-Tri.Do@lip6.fr, Thierry.Artieres@lip6.fr

Abstract. We propose a new algorithm for training a linear Support Vector Machine in the primal. The algorithm mixes ideas from non smooth optimization, subgradient methods, and cutting planes methods. This yields a fast algorithm that compares well to state of the art algorithms. It is proved to require $O(1/\lambda\epsilon)$ iterations to converge to a solution with accuracy ϵ . Additionally we provide an exact shrinking method in the primal that allows reducing the complexity of an iteration to much less than $O(N)$ where N is the number of training samples.

1 Introduction

Support Vector Machines (SVMs) are a very popular method for supervised learning tasks such as classification and regression. The standard way for solving the SVM learning problem is to introduce Lagrange multipliers, one for each constraint, and to optimize the equivalent dual problem which is an instance of quadratic programming. Since direct optimization of this quadratic problem becomes uneasy when the training set size (N) increases, some solutions have focused on efficient optimization of the dual through decomposition of the learning problem [1]. Decomposition methods like Sequential Minimal Optimization (SMO) [2], SVM-light [3], LIBSVM [4] can handle larger problems. However, their super-linear scaling behavior with N makes them intractable for very large datasets. To overcome this limitation some techniques have been investigated that mainly rely on an approximation of the dual [5,6]. However, as suggested in [7], optimizing an approximation of the dual might not be a good idea when one actually wants to optimize the primal.

Consequently, a number of recent works tackled the learning of SVM through direct optimization of the primal. By introducing the hinge loss function ($hinge(z) = \max(0, z)$), the constrained optimization problem is transformed into an unconstrained convex one. The main difficulty for optimizing this objective function lies in the non-differentiability of the hinge loss function at 0. Two main solutions have been proposed. The first one is to use a differentiable approximation of the loss (e.g. by smoothing) in which case standard optimization methods can be applied (e.g. [8,7]). [8] proposes a efficient method to solve the primal linear SVM which can be very fast if the number of features d is

small, scaling roughly as $O(Nd^2)$. The second solution is to rely on sub-gradient methods for direct optimization of the objective function [9]. The advantage of this latter approach is its simplicity, but its rate of convergence is usually much dependent on the setting of the stepsize. An exception is the Pegasos algorithm [10] which is the only one of this family that does not require the setting of this hyperparameter. Yet, note that most of those algorithms in primal [9,10] have been proposed for linear SVM.

Lastly, recent works proposed to use the cutting planes technique to build an approximated problem (i.e. a lower bound to be maximized) which is refined every iteration [11,12]. Generally speaking, the approximated problem is represented as an optimization problem with linear constraints. At each iteration, a new constraint is added to the problem and the approximated problem is solved in its dual form (it is somehow a semi dual method). Besides, one advantage of this method is that one can gain information of the approximation quality every iteration, which provides a meaningful stopping criterion.

In this paper, we propose a new algorithm for solving the SVM learning unconstrained optimization problem in the primal form (with hinge loss). Our work is a mix of subgradient methods, non smooth optimization, and cutting planes method. Our algorithm relies on the following ideas:

Simple approximation. We use the cutting planes method to iteratively build a lower bound of the primal objective function. We use a simple lower bound which can be solved very quickly. Despite its simplicity, we prove that the number of iterations required to reach the solution with accuracy ϵ is $O(1/\lambda\epsilon)$.

Efficient linesearch in the primal. Unlike many previous works, we deal with the particular shape of the objective function to derive an efficient optimal line search in a given direction from the current solution.

Shrinking in Primal. Our iterative algorithm approaches the solution iteratively. If the current solution doesn't move too far from a given iteration to the next, many constraints can be ignored in the optimization step. We propose a shrinking method that reduces the number of terms actually used in the simplified objective function to much less than N .

Section 2 presents the context and discusses related works, Section 3 recall non smooth optimization results applied to the primal objective function. Section 4 details our algorithm, the building of the lower bound, the optimal linesearch, convergence analysis, and the shrinking method. Section 5 compares experimentally our algorithm with state of the art techniques.

2 Preliminaries and Related Work

Given a training set $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$, where $x_i \in R^d$ and $y \in \{-1, +1\}$, we are interested in learning a linear classifier: $h_w(x) = \text{sign}(\langle x, w \rangle)$

where $w \in R^d$ is the model parameter set to be learned. The optimization problem can be written as follows:

$$\begin{aligned} \min_w \quad & \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{i=1..N} \xi_i \\ \text{subject to} \quad & y_i \langle x_i, w \rangle \geq 1 - \xi_i \quad \forall i \\ & \xi_i \geq 0 \quad \forall i \end{aligned} \tag{1}$$

where λ is the regularization parameter of the SVM, ξ_i are non-negative variables which represent the loss for the case where the margin constraint is violated for example x_i . Introducing the hinge loss function $hinge(z) = \max(0, z)$, we obtain the equivalent unconstrained problem:

$$\min_w f(w) \text{ with } f(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{i=1..N} \max(0, 1 - y_i \langle x_i, w \rangle) \tag{2}$$

where the second term, denoted $R_{emp}(w) = \frac{1}{N} \sum_{i=1..N} \max(0, 1 - y_i \langle x_i, w \rangle)$ is a upper bound of the empirical risk.

A simple approach to solve this problem is to use the sub-gradient method, which is proved to converge to the global minimum [13]. Pegasos [10] is an example of this type of algorithm which requires a number of iterations, independent of N , that scales with $O(1/\lambda\epsilon)$ to reach a given ϵ accuracy. However, this algorithm lacks a good stopping criterion as we observed in our experiments that the primal value may oscillate during training.

The idea of using cutting planes method for SVM-like problem was first introduced in [14,11], where the authors proposed to approximate the original problem with many linear constraints sharing a slack variable by another one with much less constraints. Their algorithm builds an approximated problem by iteratively adding the most violated constraint every iteration, until the solution of the approximate problem does not violate any constraint in the original problem more than ϵ , which means that the two problems are close enough.

Recently, [12] proposed to use a variant of bundle methods for minimizing a convex regularized risk. It is equivalent to the cutting planes method of [11] in our particular case of linear SVM learning, however its simplicity is much appealing, and our work is inspired by this one. Actually, the objective function in (2) can be lower bounded by a quadratic function with a cutting planes technique. Every iteration, one adds a new lower bound built with the (sub)gradient of the $R_{emp}(w)$ at the current solution. The approximate problem at iteration t is then:

$$\min_w \frac{\lambda}{2} \|w\|^2 + \max \{ \langle a_{j+1}, w \rangle + b_{j+1} \} \forall j = 1..t - 1 \tag{3}$$

where every term $\langle a_{j+1}, w \rangle + b_{j+1}$ is a cutting plane lower bound of R_{emp} computed at the solution at iteration j , w_j . This problem can be solved in its dual form, and its solution w_t is used for computing a_{t+1} and b_{t+1} . Note that the minimum of the lower bound increases every iteration so that the gap between the minimum observed value of the primal and the minimum of the lower bound decreases. In [12] the number of iterations required for reaching a gap less than ϵ is proved to be $O(1/\lambda\epsilon)$.

3 Primal Subgradient and Subdifferential

We present now useful properties of the primal objective that we want to minimize and discuss what the subgradient and the subdifferential of $f(w)$ (which will be used in Sect. 4) look like. Properties of interest of $f(w)$ are its convexity, its piecewise quadratic form, and its non differentiability which prevents the use of a number of smooth optimization methods.

Actually, $f(w)$ is differentiable everywhere but on hyperplanes $H^i : 1 - y_i \langle x_i, w \rangle = 0$. Every hyperplane H^i divides the parameter space into two half-spaces: $H_0^i : \{w | 1 - y_i \langle x_i, w \rangle \leq 0\}$ and $H_1^i : \{w | 1 - y_i \langle x_i, w \rangle \geq 0\}$. Hence, the N hyperplanes divide the parameter space into many polytopes: $C^k = \bigcap_{i=1..N} H_{\sigma_i^k}^i$ where $\sigma_i^k \in \{0, 1\}$. We note $I^k = \{i | \sigma_i^k = 1\}$ the set of active hyperplanes in C^k . Within any polytope C^k $f(w)$ is quadratic.

In order to optimize $f(w)$ one has to rely on the notions of subgradient and of subdifferential. Let $h : R^d \rightarrow R$ be a convex function. A vector $u \in R^d$ is called a subgradient of h at x_0 if:

$$h(x) \geq h(x_0) + \langle x - x_0, u \rangle \quad \forall x \in R^d \tag{4}$$

The set of all subgradients of h at x_0 is called the subdifferential at x_0 , and is noted $\partial h(x_0)$. The subdifferential is a nonempty, convex and compact set. The subdifferential of the objective $f(w)$ has a particular form, as we show now.

Theorem 1. *The subdifferential of the primal objective at w is defined as:*

$$\partial f(w) = \left\{ \begin{aligned} &\lambda w + \sum_{i: y_i \langle x_i, w \rangle < 1} \frac{-y_i x_i}{N} \\ &+ \sum_{i: y_i \langle x_i, w \rangle = 1} (-\beta_i \frac{y_i x_i}{N}) \end{aligned} \middle| \forall i \beta_i \in [0, 1] \right\} \tag{5}$$

where every vector $\beta = \beta_1, \dots, \beta_N$ corresponds to a particular subgradient of $f(w)$.

Proof. We base our proof on the two following results, both from [15].

Theorem 2. *Let $\{f_j : R^d \rightarrow R, j = 1, \dots, m\}$ be a set of convex and differentiable functions, then the subdifferential of $f = \max_{j=1..m} f_j$ is*

$$\partial f(x) = \text{conv} \{ \nabla f_j(x) | j \in I(x) \} \tag{6}$$

where $I(x) = \{i | f_i(x) = \max_j f_j(x)\}$, $\nabla f_j(x)$ stands for the gradient of $f_j(x)$ at x , and $\text{conv}(\cdot)$ stands for the convex hull of a vector set.

Proposition 1. *Let $\{f_j : R^d \rightarrow R, j = 1, \dots, m\}$ be a set of convex functions and let $f = \sum_{j=1..m} f_j$, then*

$$\partial f(x) = \left\{ u = \sum_{j=1}^m u_j \middle| (u_1, \dots, u_m) \in \partial f_1(x) \times \dots \times \partial f_m(x) \right\} \tag{7}$$

Algorithm 1. Global

1: **Input:** $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$, λ , ϵ
 2: $t \leftarrow 0$, $w_0 \leftarrow 0$, $\tilde{w} \leftarrow 0$, $v_0 \leftarrow 0$
 3: **repeat**
 4: $t \leftarrow t + 1$
 5: $g_t(w) \leftarrow \text{LowerBound}(w_{t-1}, \tilde{w}_{t-1}, v_{t-1})$
 6: $\tilde{w}_t \leftarrow \text{argmin}_w g_t(w)$ and $v_t \leftarrow \text{min}_w g_t(w) = g_t(\tilde{w}_t)$
 7: $w_t \leftarrow \text{linesearch}(f, \Delta_t)$ where $\Delta_t \leftarrow \tilde{w}_t - w_{t-1}$
 8: $\gamma_t \leftarrow f(w_t) - v_t$
 9: **until** $\gamma_t < \epsilon$

We begin with the definition of the subdifferential of an elementary function $\max(0, \frac{1-y_i \langle x_i, w \rangle}{N})$, which is differentiable except for w such that $y_i \langle x_i, w \rangle = 1$. For w such that $y_i \langle x_i, w \rangle \neq 1$, the function is differentiable and its subdifferential resumes to its gradient. In other cases using Theorem 2, we get:

$$\partial \left(\max(0, \frac{1 - y_i \langle x_i, w \rangle}{N}) \right) = \text{conv} \left\{ 0, -\frac{y_i x_i}{N} \right\} = \left\{ -\beta_i \frac{y_i x_i}{N} \mid \beta_i \in [0, 1] \right\} \quad (8)$$

Then the subdifferential of an elementary function is given by:

$$\partial \left(\max(0, \frac{1 - y_i \langle x_i, w \rangle}{N}) \right) = \begin{cases} 0 & \text{if } y_i \langle x_i, w \rangle > 1 \\ \left\{ -\beta_i \frac{y_i x_i}{N} \mid \beta_i \in [0, 1] \right\} & \text{if } y_i \langle x_i, w \rangle = 1 \\ -\frac{y_i x_i}{N} & \text{if } y_i \langle x_i, w \rangle < 1 \end{cases} \quad (9)$$

Next, applying Proposition 1 to $f(w)$ which is the sum of convex functions (Cf. (2)), together with the result of (9), we get the expected result. \square

4 Algorithm

The pseudo-code is given in Algorithm 1. Iteration t begins with the building of a simple lower bound $g_t(w)$ which is built based on the cutting plane method. Then the solution \tilde{w}_t minimizing this lower bound, and the minimum value v_t , are found by quadratic programming. Finally a linesearch is performed along the line from the solution in previous iteration, w_{t-1} , to \tilde{w}_t , yielding the new current solution at the t^{th} iteration, w_t . The algorithm stops once the gap $\gamma_t = f(w_t) - v_t$ is less than ϵ , which means that an ϵ -solution has been reached. The key point is that the minimum of the lower bound increases every iteration and approaches the minimum of the objective function $f(w)$. Consequently, the gap between the current value of the primal and the minimum of the lower bound decreases every iteration. The way the lower bound is built guarantees the convergence, while the linesearch procedure guarantees the decrease of $f(w_t)$.

4.1 Building a Lower Bound

In this section, we describe how we build, at the t^{th} iteration, a lower bound $g_t(w)$ of $f(w)$. Then, we describe its minimization.

Our lower bound is inspired by the recent work [12] which is based on the cutting plane method which we present first. As proposed in [12], we approximate $R_{emp}(w)$ only and let the quadratic term $\frac{\lambda}{2}\|w\|^2$ aside (Cf. (2)). This is a key issue of the technique which provides a fast rate of convergence.

Cutting Plane Technique: Whatever w_0 , the convex function $R_{emp}(w)$ can be lower bounded by using the inequality $R_{emp}(w) \geq \langle a_{w_0}, w \rangle + b_{w_0}$ where a_{w_0} is the (sub)gradient of $R_{emp}(w)$ at w_0 , b_{w_0} is the offset which can be obtained from the equality at w_0 : $\langle a_{w_0}, w_0 \rangle + b_{w_0} = R_{emp}(w_0)$. Then, the function $g_{cp}^{w_0}(w) = \frac{\lambda}{2}\|w\|^2 + \langle a_{w_0}, w_0 \rangle + b_{w_0}$ may be seen as a cutting plane approximation of $f(w)$ (with equality at w_0) which is accurate for w lying in the vicinity of w_0 . Figure 1-a shows the cutting plane approximation of $R_{emp}(w)$ at a particular w_0 and Fig. 1-d shows the corresponding quadratic lower bound $g_{cp}^{w_0}(w)$ of $f(w)$. If $R_{emp}(w)$ is not differentiable at w_0 , one can use any (e.g. random) subgradient (Fig. 1-b). However the quality of the approximation may be poor since equality $g_{cp}^{w_0}(w) = f(w)$ holds for w_0 only (Fig. 1-e).

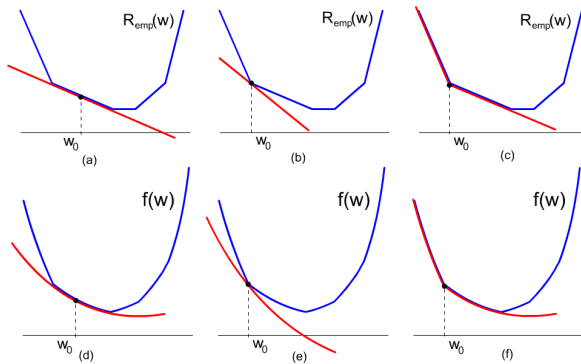


Fig. 1. Building a Lower Bound (LB, in red) of $R_{emp}(w)$ (top) and of $f(w)$ (bottom) based on cutting planes. Left ((a) and (d)): LB of the risk, $\langle a_{w_0}, w_0 \rangle + b_{w_0}$, and of the objective, $g_{cp}^{w_0}(w)$, using the gradient in the differentiable case. Middle ((b) and (e)): Same as left where $g_{cp}^{w_0}(w)$ is defined using one subgradient, in the non differentiable case. Right ((c) and (f)): LB of $R_{emp}(w)$ using the maximum over all cutting planes built from all subgradients (c) and corresponding lower bound of $f(w)$ using (12) (f).

The lower bound built from one cutting plane only $g_{cp}^{w_0}(w)$ is not a good approximation of $f(w)$. A solution is to build iteratively an increasingly accurate lower bound by adding, every iteration, a new cutting plane at the current solution [12] (Cf. (3)). At iteration t one uses the lower bound:

$$g_t(w) = \frac{\lambda}{2} \|w\|^2 + \max_{j=1..t-1} \{ \langle a_{w_j}, w \rangle + b_{w_j} \} \tag{10}$$

where w_j stands for the solution at iteration j .

Lower Bound: The main difference between our lower bound and the one in (10) is that we build the lower bound from only three elementary lower bounds $g_t(w) = \max(g_t^1(w), g_t^2(w), g_t^3(w))$. Hence we get an optimization problem that is very fast to solve. We detail now these three lower bounds.

The first lower bound is a cutting plane approximation at \tilde{w}_{t-1} , the minimum of the lower bound in previous iteration:

$$g_t^1(w) = g_{cp}^{\tilde{w}_{t-1}}(w) \stackrel{not}{=} \frac{\lambda}{2} \|w\|^2 + \langle a_t^1, w \rangle + b_t^1 \tag{11}$$

where, as before, a_t^1 is a (sub)gradient of $R_{emp}(w)$ at \tilde{w}_{t-1} , and $b_t^1 = R_{emp}(\tilde{w}_{t-1}) - \langle a_t^1, \tilde{w}_{t-1} \rangle$. The idea behind this is that we want to improve the approximation quality around \tilde{w}_{t-1} (In particular: $g_t^1(\tilde{w}_{t-1}) = f(\tilde{w}_{t-1})$).

The second lower bound is a simplification of the lower bound $g_{t-1}(w)$ of the previous iteration. The role of this lower bound, together with $g_t^1(w)$, is to guarantee a minimum improvement of the lower bound, thus providing a fast convergence rate of the algorithm (as we will see in Section 4.3). For this, We use a quadratic function $g_t^2(w) = \frac{\lambda}{2} \|w\|^2 + \langle a_t^2, w \rangle + b_t^2$, which is minimized at \tilde{w}_{t-1} , and takes the same minimum value than $v_{t-1} = g_{t-1}(\tilde{w}_{t-1})$. These conditions determine uniquely a_t^2 and b_t^2 . Note that by doing so $g_t^2(w) \leq g_{t-1}(w) \forall w$, so that $g_t^2(w)$ is also a lower bound of $f(w)$.

The third lower bound is an approximation of $f(w)$ at the solution in previous iteration, w_{t-1} . The way this solution is found (it is the result of the line search step described in the next section) makes it very often that $R_{emp}(w)$ (hence $f(w)$) is not differentiable at w_{t-1} . In theory, we could use a cutting plane approximation with any subgradient (e.g. steepest descent subgradient). However, we observed experimentally that this strategy may fail when dealing with high dimensional data (it may happen that the linesearch direction is not a descent direction of $f(w)$). To get a better approximation that equals $f(w)$ on a neighbourhood of w_{t-1} , we rather exploit the whole subdifferential. The idea is to use the maximum of all cutting plane approximations $\langle a, w \rangle + b_a$ (note that the offset b_a depends on a) built from all subgradients in the subdifferential. One then gets the following lower bound:

$$g_t^3(w) = \frac{\lambda}{2} \|w\|^2 + \max_{a \in \partial R_{emp}(w_{t-1})} (\langle a, w \rangle + b_a) \tag{12}$$

This is a better lower bound of $f(w)$ as shown in Figure 1 where a pair of (extreme) cutting plane approximations of $R_{emp}(w)$ are shown in Figure 1-c while the lower bound found by exploiting all cutting plane approximations built with all subgradients is shown in Fig. 1-f. We exploit this idea but we use a faster way to build the above approximation. The idea is to split the non-differentiable and differentiable parts of $R_{emp}(w)$ at w_{t-1} . Let $LA = \{i | \langle y_i x_i, w_{t-1} \rangle = 1\}$ denote the set of index of non-differentiable terms in $R_{emp}(w)$ at w_{t-1} (note that we omit the dependency of LA on w_{t-1} for clarity). Then:

$$R_{emp}(w) = \sum_{i \notin LA} \max(0, \frac{1 - \langle y_i x_i, w_{t-1} \rangle}{N}) + \sum_{i \in LA} \max(0, \frac{1 - \langle y_i x_i, w_{t-1} \rangle}{N}) \tag{13}$$

Let consider the cutting plane lower bound of the differentiable part of $R_{emp}(w)$ at w_{t-1} , $\langle a_t^3, w \rangle + b_t^3$, with $a_t^3 = \frac{1}{N} \sum_{i: \langle y_i x_i, w_{t-1} \rangle < 1} (-y_i x_i)$ being the gradient of the differentiable part, and $b_t^3 = \frac{1}{N} \sum_{i: \langle y_i x_i, w_{t-1} \rangle < 1} 1$. Then our third lower bound is defined as:

$$g_t^3(w) = \frac{\lambda}{2} \|w\|^2 + \langle a_t^3, w \rangle + b_t^3 + \sum_{i \in LA} \max(0, \frac{1 - \langle y_i x_i, w_{t-1} \rangle}{N}) \tag{14}$$

Putting all together, our lower bound in iteration t , $g_t(w)$ is defined as:

$$g_t(w) = \frac{\lambda}{2} \|w\|^2 + \max(\langle a_t^1, w \rangle + b_t^1, \langle a_t^2, w \rangle + b_t^2, \langle a_t^3, w \rangle + b_t^3 + \sum_{i \in LA} \max(0, \frac{1 - \langle y_i x_i, w_{t-1} \rangle}{N})) \tag{15}$$

Minimizing the Lower Bound: To minimize $g_t(w)$, we rewrite the problem in a constrained form with slack variables then we solve this constrained minimization problem in its dual form by quadratic programming. The size of this dual problem is very small with respect to the original problem, there are only $K + 3$ variables where K is the number of hyperplanes that cross w_{t-1} .

The constrained minimization problem of $g_t(w)$ can be written as:

$$\begin{aligned} \min_{w, \xi, \xi_i} & \frac{\lambda}{2} \|w\|^2 + \xi \\ \text{s.t.} & \langle a^1, w \rangle + b^1 \leq \xi \\ & \langle a^2, w \rangle + b^2 \leq \xi \\ & \langle a^3, w \rangle + b^2 + \frac{1}{N} \sum_{i \in LA'} \xi_i \leq \xi \\ & 1 - \langle y_i x_i, w_{t-1} \rangle \leq \xi_i \quad \forall i \in LA \\ & \xi_i \geq 0 \quad \forall i \in LA \\ & \xi \geq 0 \end{aligned} \tag{16}$$

Following standard derivation, this optimization problem can be solved by writing the Lagrangian then noticing that the solution is given by a saddle point of the Lagrangian, that must be minimized wrt. parameters w, ξ, ξ_i and maximized wrt. Lagrange multipliers. Omitting details, one can get the dual form:

$$\begin{aligned} \min_{\alpha} & \frac{1}{2\lambda} \beta^T A_t^T A_t \beta + \beta^T B \\ \text{s.t.} & \alpha_i \geq 0 \quad \forall i \in LA \\ & \gamma_i \geq 0 \\ & \gamma_1 + \gamma_2 + \gamma_3 \leq 1 \\ & \alpha_i \leq \frac{\gamma_3}{N} \quad \forall i \in LA \end{aligned} \tag{17}$$

where $A = [...(x_i y_i) ..., -a_1, -a_2, -a_3]$, $B = [...1 ..., -b_1, -b_2, -b_3]$ and $\beta = [...\alpha_i ..., \gamma_1, \gamma_2, \gamma_3]$ is the vector of Lagrange multipliers. Solving this optimization problem resumes to a quadratic programming problem of limited size ($K + 3$) (usually K is less than 20). Once the problem is solved in dual, the primal solution may be obtained by using the equality at saddle point: $\tilde{w}_t = \frac{\sum_{i \in LA} \alpha_i x_i y_i - \gamma_1 a_1 - \gamma_2 a_2 - \gamma_3 a_3}{\lambda}$.

It is straightforward to see that if w_{t-1} is not the optimum of $f(w)$ then the search direction given by the solution \tilde{w}_t minimizing the lower bound is a

descent direction of $f(w)$ at w_{t-1} . Actually, because $g_t(w)$ is convex, the direction from any non optimum point (for instance w_{t-1}) to \tilde{w}_t is a descent direction. Furthermore, since $g_t(w)$ matches exactly $f(w)$ on a neighbourhood of w_{t-1} the direction is also a descent direction of $f(w)$.

4.2 Optimal Line Search

The line search procedure (line 7 in Algorithm 1) is used to find an optimum solution along a line from the solution in previous iteration, w_{t-1} , to the minimum \tilde{w}_{t-1} of the current lower bound $g_t(w)$. To improve readability we consider the line search when starting from a point w_s and along a particular direction Δ . This yields the one-dimensional minimization problem:

$$\eta^* = \operatorname{arg\,min}_{\eta} g(\eta) \tag{18}$$

where $g(\eta) = f(w_s + \eta\Delta)$. Let imagine that we examine $w = w_s + \eta\Delta$ for increasing η , then w will successively cross hyperplanes H^i that are at frontiers between polytopes. Without loss of generality (by renumbering hyperplanes) assume that w crosses successively hyperplanes $H^1, H^2, H^3 \dots$ (see Fig. 2). The intersection with hyperplane $H^i : 1 - y_i \langle x_i, w \rangle = 0$ occurs (if it exists) for a particular value of η , denoted η^i , which may be computed according to:

$$\begin{aligned} 1 - y_i \langle x_i, (w_s + \eta^i \Delta) \rangle &= 0 \\ \iff \eta^i &= \frac{1 - y_i \langle x_i, w_s \rangle}{y_i \langle x_i, \Delta \rangle} \end{aligned} \tag{19}$$

$g(\eta)$ is a piecewise quadratic function (see figure 2), which is not differentiable at η^n . Within any segment $[\eta^n, \eta^{n+1}]$, $g(\eta)$ is quadratic and equals:

$$g^n(\eta) = \frac{\lambda}{2} \|w_s + \eta\Delta\|^2 + \frac{1}{N} \sum_{i \in I^{k(n)}} (1 - y_i \langle x_i, (w_s + \eta\Delta) \rangle) \tag{20}$$

where $k(n)$ stands for the index of the polytope corresponding to the n^{th} segment (and $I^{k(n)}$ is defined as in Section 3). To determine η^* , one sets $\frac{\partial g^n(\eta)}{\partial \eta} = 0$. In the n^{th} segment this yields an optimal stepsize η_{opt}^n :

$$\eta_{opt}^n = \frac{\sum_{i \in I^{k(n)}} \frac{y_i \langle x_i, \Delta \rangle}{N} - \lambda \langle w_s, \Delta \rangle}{\lambda \langle \Delta, \Delta \rangle} \tag{21}$$

Two cases may arise. Either $g(\eta)$ is differentiable at η^* or it is not. In the first case, there exists one particular n such that $\eta^n \leq \eta_{opt}^n \leq \eta^{n+1}$ and η_{opt}^n is the solution of (18). Otherwise, $\eta^* = \eta^{\hat{n}}$ and whatever n , η_{opt}^n does not belong to the n^{th} segment. In this case it is easy to show that the solution of (18) satisfies $\eta_{opt}^n \leq \eta^{\hat{n}} \leq \eta_{opt}^{n-1}$. With this discussion in mind, our algorithm examines successively the existence of η^* in segments $[0, \eta^1), [\eta^1, \eta^2), \dots, [\eta^{L-1}, \eta^L), [\eta^L, +\infty)$ until the solution (i.e. one of the two cases arises) is found.

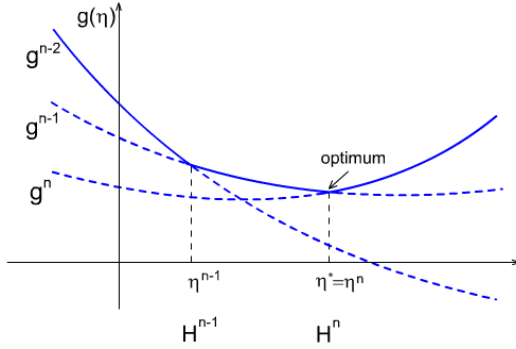


Fig. 2. Line search

4.3 Convergence Analysis

In this section, we analyse the convergence rate of the proposed algorithm based on the improvement of the lower bound.

Theorem 3. *Let G be an upper bound of the norm of the subgradient then the Algorithm 1 reaches a gap of at most ϵ after $\left\lceil \log_2 \left(\frac{\lambda}{4G^2} \right) + \frac{8G^2}{\lambda\epsilon} - \frac{8G^2}{\lambda} \right\rceil$ iterations.*

Let $\gamma_t = f(w_t) - v_t$ be the gap between the current objective value and the minimum of the lower bound at iteration t . We want to find the maximum number of iterations to reach a gap γ_t less than ϵ . For simplicity, we consider $g_t^*(w) = \max(g_t^1(w), g_t^2(w))$ (hence $v_t = \min_w g_t(w) \geq \min_w g_t^*(w)$) and study how the minimum of the lower bound v_t behaves w.r.t. $\min_w g_t^*(w)$. Note that by construction $g_t^1(w)$ and $g_t^2(w)$ have the same quadratic component, and that $g_t^1(\tilde{w}_{t-1}) > g_t^2(\tilde{w}_{t-1})$.

Let note $w^* = \operatorname{argmin}(g_t^*(w))$. By definition of our lower bounds only two cases may happen: either w^* minimizes $g_t^1(w)$ (cf. Fig. 3a), or $g_t^1(w^*) = g_t^2(w^*)$ (cf Fig. 3c), the limit case is illustrated in Fig. 3b where $\min g_t^1(w) = \frac{g_t^1(\tilde{w}_{t-1}) + g_t^2(\tilde{w}_{t-1})}{2}$. We examine now the two cases in more details.

In the first case, w^* minimizes $g_t^1(w)$ then $\min_w g_t^1(w) \geq \frac{g_t^1(\tilde{w}_{t-1}) + g_t^2(\tilde{w}_{t-1})}{2}$, and:

$$\begin{aligned} v_t &\geq \min g_t^1(w) \geq \frac{g_t^1(\tilde{w}_{t-1}) + g_t^2(\tilde{w}_{t-1})}{2} \\ &\geq v_{t-1} + \frac{g_t^1(\tilde{w}_{t-1}) - v_{t-1}}{2} \geq v_{t-1} + \frac{f(\tilde{w}_{t-1}) - v_{t-1}}{2} \end{aligned} \tag{22}$$

where we have used the equality $g_t^1(\tilde{w}_{t-1}) = f(\tilde{w}_{t-1})$ by construction of g_t^1 , and that $v_{t-1} = g_t^2(\tilde{w}_{t-1})$ by construction of g_t^2 .

Furthermore, the linesearch at iteration t makes that $f(w_t) \leq f(w_{t-1})$, and the one at iteration $t - 1$ makes that $f(w_{t-1}) \leq f(\tilde{w}_{t-1})$, and hence:

$$\begin{aligned} \gamma_t = f(w_t) - v_t &\leq f(w_t) - v_{t-1} - \frac{f(\tilde{w}_{t-1}) - v_{t-1}}{2} \\ &\leq f(w_{t-1}) - v_{t-1} - \frac{f(w_{t-1}) - v_{t-1}}{2} \\ &\leq \gamma_{t-1} - \frac{\gamma_{t-1}}{2} \end{aligned} \tag{23}$$

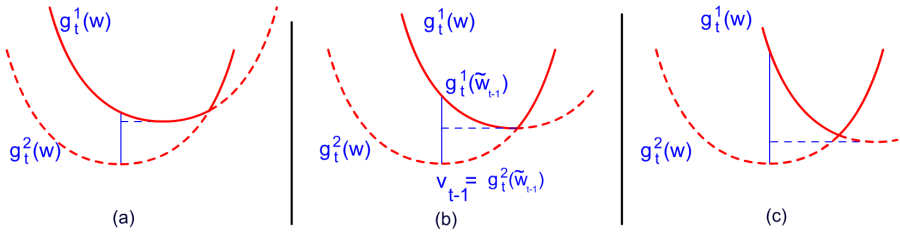


Fig. 3. Minimizing g_t^*

Let examine briefly the second case now. We do not provide the full proof since it is simple but long. We rather give hints. We know that w^* satisfies $g_t^1(w^*) = g_t^2(w^*)$, which defines a hyperplane $\langle a_t^1, w \rangle + b_t^1 = \langle a_t^2, w \rangle + b_t^2$ (since $g_t^1(w)$ and $g_t^2(w)$ have the same quadratic component). Furthermore, one can easily show that w^* must lie on the segment line delimited by the two minimum points, $\frac{-a_t^1}{\lambda}$ and $\frac{-a_t^2}{\lambda}$, of $g_t^1(w)$ and $g_t^2(w)$. After simple manipulations, one gets that the minimum $g_t^*(w)$ is given by $g_t^*(w^*) = g_t^2(\tilde{w}_{t-1}) + \frac{\lambda}{2} \frac{(g_t^1(\tilde{w}_{t-1}) - g_t^2(\tilde{w}_{t-1}))^2}{(a_1 - a_2)^2} = v_{t-1} + \frac{(f(\tilde{w}_{t-1}) - v_{t-1})^2}{(a_1 - a_2)^2}$. This may be used to build an upper bound on γ_t since:

$$\begin{aligned}
 \gamma_t &= f(w_t) - v_t \leq f(w_t) - v_{t-1} - \frac{\lambda}{2} \frac{(f(\tilde{w}_{t-1}) - v_{t-1})^2}{(a_2 - a_3)^2} \\
 &\leq f(w_{t-1}) - v_{t-1} - \frac{\lambda}{2} \frac{(f(w_{t-1}) - v_{t-1})^2}{(a_2 - a_3)^2} \\
 &\leq \gamma_{t-1} - \frac{\lambda}{8G^2} \gamma_{t-1}^2
 \end{aligned}
 \tag{24}$$

where G is an upper bound on the norm of gradient.

Putting all together, we get $\gamma_t \leq \frac{\gamma_{t-1}}{2} \min(1, \frac{\lambda}{4G^2} \gamma_{t-1})$. We see that if $\gamma_{t-1} \geq \frac{4G^2}{\lambda}$ then the inequality shows that $\gamma_t \leq \frac{\gamma_{t-1}}{2}$ and the gap is at least divided by two. Then the condition $\gamma_{t-1} \geq \frac{4G^2}{\lambda}$ happens for at most $T_0 = \log_2(\frac{\lambda}{4G^2})$ steps because $\gamma_0 = 1$. Then, we have $\gamma_t \leq \frac{\lambda}{8G^2} \gamma_{t-1}^2$. To estimate the number of iterations required to reach $\gamma_t \leq \epsilon$, we introduce a function $\gamma(t)$ which is its upper bound. Solving differential equation $\gamma'(t) = -\frac{\lambda}{8G^2} \gamma^2(t)$ with boundary condition $\gamma(T_0) = 1$ gives us $\gamma(t) = \frac{8G^2}{\lambda(t + \frac{8G^2}{\lambda} - T_0)}$ is an upper bound of γ_t . Since $\gamma(t) \leq \epsilon \iff t \geq \frac{8G^2}{\lambda \epsilon} + T_0 - \frac{8G^2}{\lambda}$, the solution is reached with accuracy ϵ within $\lceil \log_2(\frac{\lambda}{4G^2}) + \frac{8G^2}{\lambda \epsilon} - \frac{8G^2}{\lambda} \rceil$ iterations.

4.4 Complexity and Shrinking

The algorithmic complexity of our algorithm has three main components, the computation of the subdifferential (5), the quadratic programming problem, and the linesearch procedure (Sect. 4.2). However, the complexity of the quadratic programming problem is rather small, and the subdifferential is iteratively updated along the linesearch (whenever a hyperplane is crossed). Finally, the algorithmic complexity is dominated by the complexity of the linesearch. Estimation

of η^n requires 2 dot products, and sorting positive η^n is upper bounded by $N \log_2 N$. Then the overall algorithm cost is about $2Nd + N \log_2 N$, where d is the dimension of the data. In case of sparse data the complexity is related to the average number of non null components rather than to d .

In practice, w_{t-1} , w_t and \tilde{w}_t are usually close so that the set of active hyperplanes may be expected not to change much between two iterations. The idea of shrinking is to take this into account to design a more efficient algorithm by considering a limited number of hyperplanes only. Let consider a convex region Q around w_{t-1} (e.g. a ball) and let H_I denote the set of inactive hyperplanes with respect to Q (i.e. hyperplanes that don't cross Q), and let $H_A = \{1, \dots, N\} \setminus H_I$ the set of active hyperplanes. Then $f(w)$ coincides with $f^Q(w)$ on Q , where:

$$f^Q(w) = \frac{\lambda}{2} w^2 + \sum_{i \in H_A} \max(0, 1 - \langle y_i x_i, w \rangle) - \langle A_{H_I}, w \rangle + C_{H_I} \quad (25)$$

where $A_{H_I} = \sum_{i \in H_I, y_i x_i w_t < 1} y_i x_i$ and $C_{H_I} = \sum_{i \in H_I, y_i x_i w_t < 1} 1$. We may consider the simplified line search problem $\min_{\eta} f^Q(w_t + \eta \Delta_t)$ which can be solved with the algorithm in Sect. 4.2. If the solution $w_{opt} \in Q$ then w_{opt} is a local minimum of f because the approximation is correct within Q . Hence w_{opt} is a global minimum of f since it is convex. A simple method for implementing these ideas consists in starting with a ball centered at w_{t-1} and with a small radius so that number of active hyperplanes is small enough (e.g 10). If the obtained solution w_{opt} belongs to the ball then we have our solution $w_t = w_{opt}$. Otherwise, we increase the radius of the ball (i.e. the number of active hyperplanes) and start a new linesearch until we find a solution which is correct with respect to the ball considered. This approach is described in Algorithm 2. To simplify the presentation, we assume that all the distances from w_{t-1} to all hyperplanes are precomputed and sorted in ascending order so that: $dist(w_{t-1}, H_{h_1}) \leq dist(w_{t-1}, H_{h_2}) \leq \dots \leq dist(w_{t-1}, H_{h_N})$. Also we consider here that N is a power of 2 and that we use $\log(N)$ balls, the k^{th} one corresponding to 2^k active hyperplanes.

Algorithm 2 outputs w_{t+1} without considering all hyperplanes in the linesearch, but it requires to compute the distance from the current solution w_t to the N hyperplanes. The number of dot products to be calculated is then still linear with N , while the complexity of the linesearch ($M \log_2 M$) now depends of the average number of active hyperplanes M (possibly N). It is possible to go further by maintaining balls with different centers. Imagine that at iteration t , we define a ball $B_k = Ball(w_t, R_k)$ that is large enough to include all forthcoming solutions $w_{t+1}, w_{t+2}, \dots, w_{final}$. Then all hyperplanes that are inactive with respects to B_k could be completely ignored in the following iterations. This means we may avoid computing all distances each iteration and instead reuse balls from previous iterations up to a certain extent. We do not detail this algorithm here because we lack room for that, we only give the idea. The main difference with the previous simpler shrinking algorithm is that the balls that are maintained during linesearch are not all centered on w_{t-1} , but may be centered on w_{t-2}, w_{t-3}, \dots . Once a solution of the reduced line search is found (with the set of active

Algorithm 2. line search one center

```

1: Input:  $w, \Delta, H_1, \dots, H_N$ 
2:  $NB = \log_2 N$ 
3: Compute distances from  $w$  to all hyperplanes and renumber hyperplanes with in-
   creasing distance.
4: for  $k=1:NB$  do
5:   Set  $R_k =$  distance to hyperplane number  $2^k$ .
6:   Set  $L =$  number of active hyperplanes in  $Ball(w, R_k)$ .
7:   Compute  $\eta^1, \dots, \eta^L, g^1, \dots, g^{L+1}$  with active hyperplanes  $H_1, \dots, H_{2^k}$ .
8:    $\eta_{opt} = LineSearch(\eta^1, \dots, \eta^L, g^1, \dots, g^{L+1})$ 
9:    $w_{opt} = w + \eta_{opt}\Delta$ 
10:  if  $w_{opt} \in Ball(w, R_k)$  return  $w_{opt}$ 
11: end for

```

hyperplanes of a ball B_j), one has to check if this solution is valid, that is $w_{opt} \in \cap_{k=j}^{NB} B_k$. If this is not the case, one has to recompute distances with hyperplanes in the smallest ball including the solution and restart the procedure.

5 Experimental Results

We performed binary classification experiments on 6 datasets (Table 1) to compare our algorithm (named Hyperpass) with two reference methods, the sub-gradient algorithm Pegasos (on-line version) of [10] and the Cutting Planes algorithm svmperf of [11]. We used our own implementation of Pegasos where we set the parameter K (number of examples uses to compute sub-gradients) to 1000 whereas we used the available code for svmperf¹. Cross validation over the full datasets is used to determine the parameter λ for each dataset ($\lambda = 10^{-3}, 10^{-4}, \dots, 10^{-8}$). To fairly compare the behaviour of the three algorithms w.r.t. the number of learning iterations, one iteration of Pegasos stands for $\frac{N}{K}$ subiterations (using K examples). The complexity of an iteration is $O(N)$ whatever the algorithm.

Figure 4 shows the evolution of the Primal objective as a function of the number of iterations. Actually, it plots the gap obtained for the current solution $gap(w) = f(w) - v^*$ where v^* stands for the best lower bound value observed during training (in Hyperpass and Cutting Plane runs). On the one hand, on low-dimensional datasets (first row of plots), one can see that Hyperpass both requires much less iterations than Pegasos and svmperf to reach a good ϵ -solution for all these datasets and that it converges to a more accurate solution. On the other hand, for high-dimensional datasets, Hyperpass may be slower than Pegasos in the first 20 to 100 iterations, then its convergence is faster and more accurate. Whatever the dataset Hyperpass is much faster and accurate than svmperf. Similar to these above situations, we observed that Hyperpass attends the minimum error rate faster than Pegasos in low-dimensional settings (roughly

¹ http://svmlight.joachims.org/svm_perf.html

Table 1. Datasets

DATASET	ADULT9	COVERTYPE	WEB8	CCAT	C11	REALSIM
#EXAMPLES	48842	581012	64700	804414	697641	72309
DIMENSION	123	54	300	47236	47236	20958

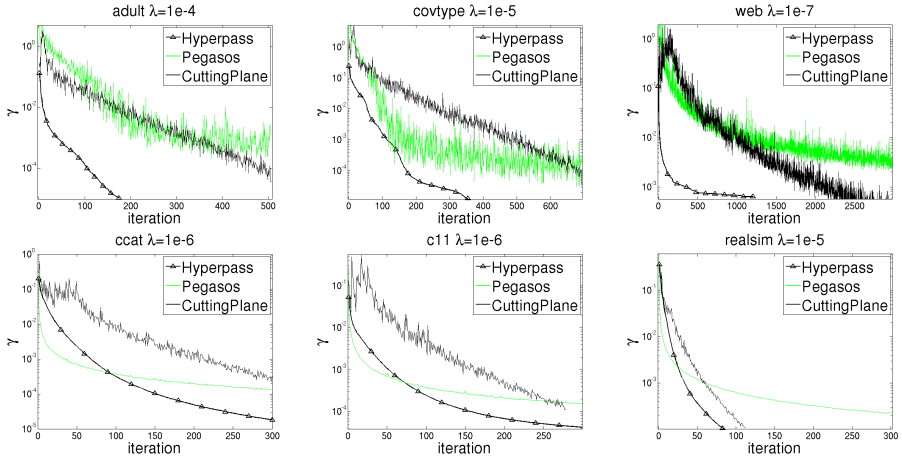


Fig. 4. Evolution of the gap (Primal- v^*) as a function of the number of iterations

when the gap is about 1% of the primal value); for high-dimensional data, Pegasos achieves the minimal error rate with less iterations. The three methods reach same performance after convergence (svmperf being the slowest one in any case).

Furthermore the complexity of an iteration in Hyperpass may be drastically reduced using shrinking as shown experimentally. Figures 5-a,b plot the same quantity as in Figure 4 but as a function of the number of dot products normalized by N (it is a number equivalent to a number of iterations without shrinking) for two low dimensional datasets ². The curves do not change for Pegasos and svmperf while we see that the total number of operations actually required by Hyperpass is actually much reduced. For instance with the *web* dataset, there are less than $250 * N$ dot product computations in about 1500 iterations. Figure 5-c gives more insight about the efficiency of the shrinking method by plotting the number of dot products actually computed every iteration. For visibility reason, we only present the result for three datasets *adult*, *web8* and *covtype*. As can be seen the number of operations in one iteration may be up to 50 times lower than N . It is interesting to note that although *covtype* is 10 times larger than *web8* the complexity of an iteration of *covtype* may be less than the one of *web8*.

² These plots compare the computational complexity of the three algorithms. Please note that a comparison based on CPU time would be unfair because the implementations might not be equally optimized.

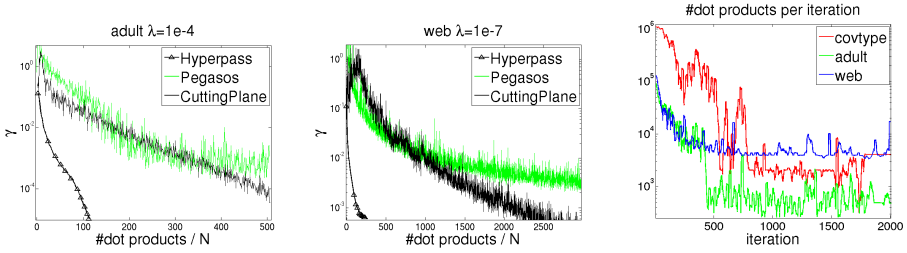


Fig. 5. Evolution of the gap (Primal- v^*) as a function of $\frac{\# \text{dot products}}{N}$

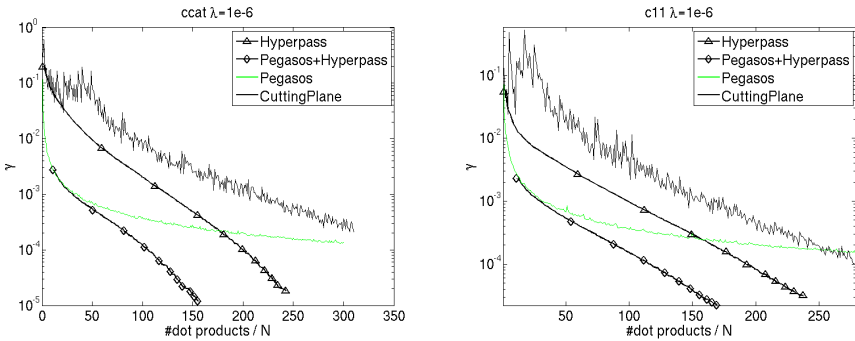


Fig. 6. Evolution of the gap (Primal- v^*) as a function of $\frac{\# \text{dot products}}{N}$

For some high dimensional datasets as we said previously, Pegasos may converge faster at the beginning of the learning. But then, it requires a larger number of iterations than Hyperpass, to reach a solution that is less accurate. While Hyperpass may be seen to outperform Pegasos in the long run (i.e. to reach a very accurate solution) it may benefit from initialization, i.e. a bootstrap, using Pegasos. We investigate the use of Pegasos as a smart starter for Hyperpass for high dimensional datasets in Fig. 6. This Figures plot the evolution of the gap as a function of the number of dot products for four methods, the three already compared and a fourth one which is based on Hyperpass with an initialization given by Pegasos after 10 iterations. As may be seen on these plots, this combined Pegasos-Hyperpass algorithm performs much better than any of the three others, and converges at the same rate (the fastest one) than Hyperpass.

6 Conclusion

We presented an original algorithm for training SVM in the primal, for which we provided convergence rate analysis. We proposed an efficient shrinking technique and showed experimentally that our algorithm converges much faster than state

of the art algorithms on number of benchmark datasets, wrt. to the number of iterations and to the number of operations.

References

1. Osuna, E., Freund, R., Girosi, F.: Training support vector machines: An application to face detection. In: CVPR 1997 (1997)
2. Platt, J.C.: Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research (1998)
3. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge (1999)
4. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001)
5. Bordes, A., Ertekin, S., Weston, J., Bottou, L.: Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research* 6, 1579–1619 (2005)
6. Tsang, I.W., Kwok, J.T., Cheung, P.M.: Core vector machines: Fast svm training on very large data sets. *J. Mach. Learn. Res.* 6, 363–392 (2005)
7. Chapelle, O.: Training a support vector machine in the primal. *Neural Computation* 19(5), 1155–1178 (2007)
8. Mangasarian, O.L., Musicant, D.R.: Lagrangian support vector machines. *Journal of Machine Learning Research* 1, 161 (2001)
9. Zhang, T.: Solving large scale linear prediction problems using stochastic gradient descent algorithms. In: *ICML 2004*, vol. 116. ACM Press, New York (2004)
10. Shalev-Shwartz, S., Singer, Y., Srebro, N.: Pegasos: Primal estimated sub-gradient solver for SVM. In: *ICML 2007*, pp. 807–814. ACM Press, New York (2007)
11. Joachims, T.: Training linear SVMs in linear time. In: *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pp. 217–226 (2006)
12. Teo, C.H., Le, Q.V., Smola, A., Vishwanathan, S.V.: A scalable modular convex solver for regularized risk minimization. In: *KDD 2007*, pp. 727–736 (2007)
13. Nedic, A., Bertsekas, D.: Convergence rate of incremental subgradient algorithms. In: Uryasev, S., Pardalos, P.M. (eds.) *Stochastic Optimization: Algorithms and Applications*. Kluwer Academic, Dordrecht (2000)
14. Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: *ICML 2004*, pp. 104–112 (2004)
15. Bertsekas, D.P., Nedic, A., Ozdaglar, A.E.: *Convex analysis and optimization*. Athena Scientific, Belmont (2003)