Control and Cybernetics

vol. **38** (2009) No. 1

A linear Support Vector Machine solver for a large number of training examples[*]

by

Paweł Białoń

National Institute of Telecommunications
1 Szachowa Str., Warsaw, Poland

Abstract: A new linear Support Vector Machine algorithm and solver are presented. The algorithm is in a twofold way well-suited for problems with a large number of training examples. First, unlike many optimization algorithms, it does not simultaneously keep all the examples in RAM and thus does not exhaust the memory (moreover, it smartly passes through disk files storing the data: two mechanisms reduce the computation time by disregarding some input data without a loss in solution quality). Second, it uses the analytical center cutting plane scheme, appearing as more efficient for hard parameter settings than the Kelley's scheme used in other solvers, like `SVM_perf`. The experiments with both real-life and artificial examples are described. In one of them the solver proved to be capable of solving a problem with one billion training examples. A critical analysis of the complexity of `SVM_perf` is given.

Keywords: Support Vector Machine, analytic center cutting plane method, RAM volume required.

1. Introduction

Many practical tasks yield Support Vector Machine (SVM) classification problems (see Vapnik, 1995, Musicant, 2000) with a large number n of training examples. The clear reason is the large number of records in modern databases. Increasingly often these records are produced automatically, based on the data collected from sensors, telecommunication networks control protocols, etc., at regular time intervals. And it is usually not easy to reduce the complexity of the SVM problem by a priori filtering some of the training examples: any arbitrary operation in this context may lead to a loss of the most meaningful examples, i.e., the closest to the separating hyperplane, especially when they represent some rare events.

[*]Submitted: June 2008; Accepted: October 2008

We present a specific solver, specialized for solving SVM optimization problems with large n. Many optimization solvers assume that they can simultaneously store all the data, defining the problem, in RAM. This is disputable for the application in SVM with very large n. Our solver, instead, makes many passes through large data sets; in this it is closer to machine learning algorithms than to optimization solvers. Moreover, we will introduce two mechanisms that reduce the amount of data read by our algorithm. First, it is checked whether the number of examples read in the current file pass is sufficient to gain new information about the supporting hyperplane location. This mechanism allows to finish file passes in early stages of the algorithm run. In the later stages, the second mechanism intervenes more often. It consists in neglecting many examples that are clearly far from the hyperplane (of whose position we have a good knowledge in the late stages of the algorithm run).

For an optimization problem with a large number of constraints the choice of a cutting plane-scheme is natural. Such a scheme reduces the original SVM problem to a sequence of easier optimization subproblems. These are problems of minimizing a function of N (or a bit more) variables – where N is the number of features in SVM – subject to constraints (cuts) being generated as the cutting plane algorithm runs and reads new training examples.

This work is also intended to be a voice encouraging the use of the newest concepts of the optimization science, namely, interior point methods – IPM (Goffin and Vial, 1999; Terlaky, 1996; Nesterov, 1995) in solving SVM problems.

Existing solvers that use older techniques (e.g., active set techniques, possibly with a reduction of the number of used variables with chunking) can be very efficient (see, e.g, Musicant, 2000, and numerous references therein; Platt, 1998; Joachims, 1999; Scheinberg, 2006). However (most) IPMs have a unique feature of being polynomial-time algorithms for solving linear optimization problems, and even more attractive is the practical efficiency of IPMs.

The relations between the older techniques and IPMs will be illustrated by comparing our solver with `SVM_perf` (Joachims, 2006) of a very similar application area (linear separation, large n). However, `SVM_perf` uses the old Kelley's cutting plane method (Kelley, 2006) while we use the interior point Analytic Center Cutting Plane Method – ACCPM from Nesterov (2005).

`SVM_perf` is a very interesting solver, since its author has proved its linear complexity in both the number of features and the number of examples, receiving much attention. This paper includes a critical discussion of this result and shows poor complexity in some parameter settings (e.g., final accuracy of the solution). The practical comparison of both solvers is in favor of the solver presented here for harder parameter settings.

Among other related work, the interesting solver from Bordes and Bottou (2005) passes through the data set even just once (!). However, it solves only hard-margin SVMs and its steps, involving convex hulls, are complex. The existing IPM implementations for SVM (Ferris and Munson, 2003; Vandenberghe and Comanor, 2003; Fine and Scheinberg, 2001, to name a few) not necessarily assume linear separation and large n, and not necessarily use cutting plane methods. They, instead, need special techniques, like rank-one updates, to deal with large Hessian/kernel matrices – Fine, Scheinberg (2001). Our work stands out also owing to two accelerating mechanism. The solver from Ferris and Munson (2003) uses bucketing, similarly to our 2$^{\text{nd}}$ mechanism; however, bucketing is driven there by the specific needs of sparse Cholesky matrix factorization.

In Section 2 we pose the problems solved, and in Section 3 we show their auxiliary transformations. The ACCP method used is described in Section 4, the effective computing of goal functions and constraints, i.e., our two accelerating mechanisms – in Sections 5 and 6. Section 7 contains the discussion of the efficiency of both `SVM_perf` and the author's solver. Section 8 summarizes the experiments with our solver, Section 9 gives some conclusions.

Notation. We shall identify tuples (elements of Cartesian products) with column vector, e.g $(1,2) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. The colon will be used together with parentheses to denote a concatenation of vectors (or vectors with numbers). For example, if $a = (1,4)$, $b = (10,20)$ and $c = 8$ then $(a;b) = (1,4,10,20)$ and $(a;c) = (1,4,8)$. Subscripts will be used to denote components of vectors and matrices, while superscripts – for other purposes, especially to denote the iteration number. For a matrix A, A_i will denote the ith column of A and a_i – the ith row of A. Given a convex function $f : \mathbb{R}^n \to \mathbb{R}$ and $x_0 \in \mathbb{R}^n$ we define the *subdifferential* of f at x_0 as $\partial f(x_0) = \{v \in \mathbb{R}^n : \ f(x) \geq f(x_0) + v^\top (x - x_0)\}$, and any $g \in \partial f(x_0)$ will be called a *subgradient* of f at x_0. For differentiable functions, the subgradient reduces to the mere gradient. The default norm will be the second norm.

2. Linear SVM problem

Let matrix $A_{n \times N}$ be composed of n rows a_i that represent n training examples, each one with N feature values. Each example belongs either to class $\mathcal{A}$ or class $\mathcal{B}$. We assume that n is very large. We pose the following *exact separation problem*:

$$\text{(E)} \quad \underset{\omega \in \mathbb{R}^N,\, \gamma \in \mathbb{R}}{\text{minimize}} \; f(\omega;\gamma) \equiv \frac{1}{2}\|\omega\|^2$$

subject to

$$g_j((\omega;\gamma)) \equiv \big(-d_j \cdot (a_j\omega - \gamma) + 1 \big) \leq 0 \quad \text{for } j = 1, \ldots n. \tag{1}$$

The column vector $d \in \mathbb{R}^n$ is defined as follows: $d_i = -1$ if ith example is of class $\mathcal{A}$ and $d_i = 1$ if ith example is of class $\mathcal{B}$.

As well known (Musicant, 2000), the solution $(\omega^\star; \gamma^\star)$ for this problem represents a classification hyperplane (x is of class $\mathcal{A}$ iff $\omega^\top x \geq \gamma$). Besides this hyperplane, infinite many hyperplanes parallel to it properly separate the examples. The distance between the two extreme ones of them, the separation margin, equals to $\frac{2}{\|\omega\|}$.

By allowing the violation of constraints (1) and by adding their violation to the goal function of this problem, we obtain the *inexact separation problem*:

$$
\text{(I)} \quad \underset{\omega \in \mathbb{R}^N,\, \gamma \in \mathbb{R},\, y \in \mathbb{R}^n_+}{\text{minimize}} \frac{1}{2}\|\omega\|^2 + Ce^\top y
$$

subject to

$$
-d_j \cdot (a_j\omega - \gamma) - y + 1 \leq 0 \quad \text{for } j = 1, \ldots n.
$$

Here $e^\top = (1, 1, \ldots, 1) \in \mathbb{R}^n$, y is a vector of variables representing the violations and the positive real parameter C is a user-defined weight representing the trade-off between the separation margin and the total constraint violation.

3. Transformations of the inexact separation problem

Our ACCP method can effectively solve optimization problems with very many constraints but the number of variables should be kept moderate. However, the transition from problem (E) to problem (I) introduced as many as n new variables y_k. Thus, we propose two reformulations of problem (I), yielding the same solution $(\omega; \gamma)$ and having much less variables, by including violations directly into the goal function using the max operation.

Transformation 1

$$
\text{(I1)} \quad \underset{\omega \in \mathbb{R}^N,\, \gamma \in \mathbb{R}}{\text{minimize}} f((\omega;\gamma)) \equiv \big(\frac{1}{2}\|\omega\|^2 + C \sum_{j \in \{1,\ldots,n\}} \max(0,\, -d_j \cdot (a_j\omega - \gamma) + 1)\big).
$$

Transformation 2 We first transform (I1) with an addition of a slack variable z:

$$
\underset{\omega \in \mathbb{R}^N,\, \gamma \in \mathbb{R},\, z \in \mathbb{R}}{\text{minimize}} \frac{1}{2}\|\omega\|^2 + Cz
$$

subject to

$$
\sum_{j \in \{1,\ldots,n\}} \max(0, -d_j \cdot (a_j\omega - \gamma) + 1) - z \leq 0
$$

and further:

$$
\text{(I2)} \quad \underset{\omega \in \mathbb{R}^N,\, \gamma \in \mathbb{R},\, z \in \mathbb{R}}{\text{minimize}} f((\omega;\gamma;z)) \equiv \left(\frac{1}{2}\|\omega\|^2 + Cz\right)
$$

subject to

$$
g_1((\omega;\gamma;z)) \equiv \big(\sum_{j \in \{1\}} \max(0, -d_j \cdot (a_j\omega - \gamma) + 1) - z\big) \leq 0
$$

$$g_2((\omega;\gamma;z)) \equiv \big(\sum_{j\in\{1,2\}} \max(0,-d_j\cdot(a_j\omega-\gamma)+1)-z\big) \leq 0$$

$$\dots$$

$$g_n((\omega;\gamma;z)) \equiv \big(\sum_{j\in\{1,\dots,n\}} \max(0,-d_j\cdot(a_j\omega-\gamma)+1)-z\big) \leq 0.$$

Note that in (I2) the constraints are aligned so that each successor implies its predecessor. Thus, all but the last constraints are redundant (however, as we shall see, they are desired to shorten computations within the oracle).

4. The analytic center cutting plane method

Solving the optimization problems (E), (I1) or (I2) is based on the variant of Analytic Center Cutting Plane method (scheme) with a proximal term, proposed in Nesterov (1995).

We shall describe the Nesterov scheme. It yields a chain of optimization and numerical algebra subproblems, whose solving is only shortly documented in Appendix 9. Solving them generally follows the cited paper of Nesterov, with some modifications, mainly introducing line search and other ways of rank-one updating. The most innovate part of the solver is its sophisticated *oracle*, described in more details in Sections 5 and 6.

Our solver has been developed since the work Białoń et al. (2004) and has grown to a rather complex, multilayered program with many further improvements and modifications not covered here.

The Nesterov ACCP scheme (Algorithm 1) is used in our solver to solve either of (E), (I1), (I2). We shall use a notation with which we will be able to describe either of these problems as an instance of one general optimization problem. Namely, we define $x = (\omega;\gamma)$ for problems (E), (I1) and $x = (\omega;\gamma;z)$ for problem (I2). We use the definitions of f and g given within (E), (I1) or (I2), depending on which of the three problems we are considering. We denote $\tilde{N} = \dim x^k$. Now the general problem can be written as:

$$\text{(G)}\quad \underset{x\in\tilde{N}}{\text{minimize}}\, f(x)$$
$$\text{subject to}$$
$$g_j(x) \leq 0 \qquad \text{for } j = 1,\dots,\tilde{n}$$

Later on, we shall assume that f and all g_i are convex functions, which will be satisfied in the actual problems solved by both the authors' solver and `SVM_perf`.

Algorithm 1 The Analytic Center Cutting Plane Method

```
   Variables: starting point $x^0 = 0$, the upper bound $R$ for the solution norm, initial potential function $F^0 : \mathbb{R}^{\tilde{N}} \mapsto \mathbb{R}$, $F^0(x) = 4\|x\|^2/R^2$, iteration counter $k$
1: $k$:=0
2: loop
3:    $\gamma^k$:=oracle($x^k$). Set new potential function $F^k : \mathbb{R}^{\tilde{N}} \mapsto \mathbb{R}$:
      $F^{k+1}(x) = F^k(x) - \ln({\gamma^k}^\top(x - x^k)) + \frac{1}{2R^2}\|x\|^2$
4:    Compute new iterate: $x^{k+1}$:=$\arg\min_{x \in \mathbb{R}^{\tilde{N}}} F^{k+1}(x)$
5:    $k$:=$k + 1$
6: end loop
```

Let us explain the main ideas behind this algorithm.

To find the solution, the method implicitly constructs consecutive *localization sets* $L^0 \subset L^1 \subset \ldots$, each of them containing the solution.

Nesterov shows that all the iterates x^k of this method are in the interior of the ball $B = \{x : \|x - x^0\| \le 2R\}$. Since we even assumed that solution lies in $\{x : \|x - x^0\| \le R\}$, we put $L^0 = B$. Then the `oracle` subroutine[1] is used to find a *proper cut* at x^k, e.g., a linear inequality of the form ${\gamma^k}^T x^k \le {\gamma^k}^T$, allowing for the problem solution. Technically, the oracle only returns the „gradient" γ^k of the cut. Each consecutive cut narrows the localization set, i.e. we have $L^k = L^{k-1} \cap \{x : {\gamma^k}^T x \le {\gamma^k}^T x^k\}$.

As we can easily see, $F^k = \frac{k}{2R} + \sum_{i=0}^{k}(-\ln \gamma^i(x - x^i))$. Its first component is the Nesterov regularizing term, needed in the convergence analysis, while the second term is a potential function pushing x^{k+1}, the minimum point of F^k, away from the cuts, thus away from the boundaries of L^k. Since, as we have said, x^{k+1} lies in the interior of B, we see that x^{k+1} lies in the interior of the localization set L^k. This substantiates the classification of this method as an interior point method.

The practical strength of the method lies in the quick contraction of localization sets by cuts. The minimum points of F^k, called *analytic centers* turn out in practice to be good centers of localization sets with respect to a quick narrowing of these sets (not as good as, for example, the center of gravity, but good enough and, unlike the center of gravity, relatively easily computable).

5. The oracle with early returns

The oracle returns the „gradient" of a valid cut at the current iterate $x^k \equiv (\gamma^k; \omega^k)$ (problems (E) and (I1) or $x^k \equiv (\gamma^k; \omega^k; z^k)$ (problem (I2). More specifically, the oracle returns the subgradient of one of the unsatisfied constraints

[1]The term *oracle* in general denotes a part of the solver that gives the core algorithm information about the particular instance of the optimization problem being solved.

(e.g, an element of $\partial g_l(x)$ for one of the ls such that $g_l(x) > 0$) or – if there is no unsatisfied constraint – it returns the subgradient of the goal function (an element of $\partial f(x)$).

Our oracle will sometimes be able to early (without having passed through the whole data) return from a call: this will be our ”first accelerating mechanism”.

The realization of a call `oracle`(x^k) in Algorithm 1 will be now shown separately for optimizations in problems (E), (I1), (I2). We assume for a while that $J = \{1, \ldots, n\}$. The `totalgetandputcounter` is set to zero at the beginning of optimization (this variable will count the overall number or reads/writes of rows a_j from files). The operation ”take a new j from J” will not substract anything from the set J but will only set variable j. Moreover, we assume that j are taken in the increasing order.

In the first oracle variant, we return after meeting the first unsatisfied constraint; this may happen very quickly in the early stages of optimization.

Algorithm 2 Oracle call for (E)

```
1: while $J$ not exhausted do
2:    Take a new $j$ from $J$. Increase totalgetandputcounter by 1.
3:    if $-d_j(\omega^k a_j - \gamma^k) + 1 > 0$ then
4:       return $-d_j \cdot (a_j^\top; -1)$ {return subgradient of $g_j$}
5:    end if
6: end while
7: return $\omega^k$ {return goal function subgradient}
```

In the following variant we shall need to look through the whole data.

Algorithm 3 Oracle call for (I1)

```
$r{:=}(\omega^k; 0)$
while $J$ not exhausted do
   Take a new $j$ from $J$. Increase totalgetandputcounter by 1.
   if $-d_j(a_j\omega^k - \gamma^k) + 1 > 0$ then
      $r{:=}r - C \cdot d_j \cdot (a_j^\top; -1)$
   end if
end while
return $r$ {return goal function subgradient}
```

In the last variant, we cheaply (incrementally) construct the consecutive constraints from problem (I2) and also can return early.

Algorithm 4 Oracle call for (I2)

```
$r$:=0 $\in \mathbb{R}^{N+1}$ {$r$ corresponds to $g_i$ in 3 with $i = k$}.
$s$:=0 $\in \mathbb{R}$
while $J$ not exhausted do
  Take a new $j$ from $J$. Increase totalgetandputcounter by 1.
  if $-d_j \cdot (\omega_j a_j - \gamma^k) + 1 > 0$ then
    $r$:=$r - d_j \cdot (a_j^\top; -1)$
    $s$:=$s - d_j \cdot (a_j^\top \omega^k - \gamma^k) + 1$
    if $s - z > 0$ then
      return  $(r; -1)$  {return subgradient of $g_j$; an early return}
    end if
  end if
end while
return  $(\omega; 0; C)$ {return goal function subgradient}
```

6. Disregarding some data in the late stages

We have learned to effectively skip some file data by early finding violated constraints. This technique will be not effective when we get close to the problem solution: then most of the constraints are satisfied.

Then, however, we have another possibility of skipping some file data. If we are close to the solution, x^k does not change much from iteration to iteration. Thus, roughly speaking, satisfaction of a particular constraint in $(k+s)$th iteration will usually be assured by its satisfaction in kth iteration (with $s \geq 0$). Thus, an explicit examination of the satisfaction in the $(k+s)th$ iteration, together with the reading of a_j, the jth row of A, from a file, is not necessary.

More precisely, we shall use the above fact not exactly for the three problem constraints but for a certain inequality.

Looking at Algorithms 2 through 4, we see that a_j is used in computations only if the inequality

$$-d_j(a_j\omega^k - \gamma^k) + 1 > 0 \tag{2}$$

is satisfied. Thus, instead of using the original $J = \{1, \ldots n\}$, in Algorithms 2-4, we can use any set J such that for $j = 1, \ldots, n$ $\;(2) \Rightarrow j \in J$ and we will not change the return value of the oracle call. Reducing the size of J could correspond to disregarding some input data for the problem.

We note the following fact:

$$\begin{aligned} -d_j(a_j\omega^k - \gamma^k) + 1 \leq -\|a_j\| \cdot c \wedge \|(\omega^k; \gamma^k) - (\omega^{k+s}; \gamma^{k+s})\| \leq c \Rightarrow \\ -d_j(a_j\omega^{k+s} - \gamma^{k+s}) + 1 \leq 0 \end{aligned} \tag{3}$$

for $s, k \geq 0$, $c \geq 0$. This says that if the inequality (2) was satisfied in kth iteration with some margin $\|a_j\| \cdot c$, and the current iterate moved by less than c

between kth and $(k + s)$th iteration, the same inequality for the $(k + s)$th iteration, i.e., with k replaced by $k + s$, will also hold. Formally, (3) could be shown using the triangle inequality.

We make a modification to our ACCP algorithm.

1. At the beginning of each kth iteration of Algorithm 1, we decide whether to perform a "bucket dispatching" (by which we mean the first phase of bucket sorting – dispatching elements into buckets).
2. If we decided to do so, we dispatch our input data records (mainly the rows a_j) into $\mathfrak{r}$ buckets according to the value $v_j \equiv -\frac{-d_j(a_j\omega^k-\gamma^k)+1}{\|a_j\|}$; $\mathfrak{r}$ is an algorithm parameter. Physically, it means that we dispatch all the data in $\mathfrak{r}$ temporary files.
3. Since the dispatching took n reads and n writes of an A row, we increase `totalgetandputcounter` by $2n$.
4. We prepare a set J narrower than $\{1, \ldots, n\}$ for the oracle calls from now till the next bucket dispatching. Namely, in iteration $k + s$, we put

$$J = \{1, \ldots, n\} \setminus \{j : \ v_j \geq \|(\omega^{k+s}; \gamma^{k+s}) - (\omega^k; \gamma^k)\|\}. \tag{4}$$

   One easily verifies, using (3) with $c = v_j$, that now $-d_j(a_j\omega^{k+s} - \gamma^{k+s}) + 1 > 0$ implies $j \in J$, as desired. Technically, we do not remember all (v_j)s nor have any explicit data structure for J. Instead, the oracle reads a_js only from selected buckets (files)[2].

Remark 1 *We should not make "bucket dispatching" too often, since each one involves $2n$ gets/puts. Thus we could slow down our algorithm, instead of accelerating it. If we require that* `totalgetandputcounter` *increases by at least $2Kn$ between the end of one and the beginning of another dispatching (i.e., by the oracle activities themselves), we assure that "bucket dispatching" slows down our algorithm at most $(K + 1)/K$ times. With K equal to, say, 3, this is not much and the gain from narrowing J will be probably bigger.*

7. Comments on the efficiency of the solvers

The purpose of this section is to investigate the very interesting relations between the performance characteristics of both the authors' solver and the `SVM_perf` of Joachims'. It is shown how the linear complexity of `SVM_perf` in both n and d (which is a surprisingly good result for an optimization solver!) can be explained with a poor complexity with respect to the solution accuracy. The character of our solver is complementary to that: our solver is not afraid of hard accuracy settings but for loose accuracy settings it reacts stronger to problem sizes.

Regarding the ACCP method, we shall only recall some main complexity results from Nesterov (1995). Therein is included a detailed complexity analysis

[2]This is only an approximation (quantization); we actually obtain a slightly bigger J than the one defined by (4).

of his ACCP method. The Nesterov's proofs are rather involved, they operate simultaneously on various levels of the algorithm, and use two notions introduced by him (*self-concordant functions* and *compatible matrices*). Thus, reproducing the proofs for the case of our solver is out of the scope of this paper.

7.1. The Kelley's scheme in `SVM_perf`

We shall show the operations made by `SVM_perf` (see Algorithm 1 in Joachims, 2006) in terms of the general problem (4), in order to have a common view of both solvers. However, his actual problem resembles our (3) in its structure, in particular, he uses a slack variable like z to represent the total classification error and, most importantly $\tilde{N}$ is similar to N and $\tilde{n}$ is large.

The Kelley's method used in `SVM_perf` is outlined as Algorithm 5.

Algorithm 5 The Kelley's method

```
   Variables: starting point $x^0 = 0$, iteration counter $k = 0$.
1: loop
2:   Find a new cut $\gamma^k$:=oracle($x^k$).
3:   Compute new iterate: $x^{k+1}:=\arg\min_{x \in \mathbb{R}^{\tilde{N}}} f(x)$ subject to $\gamma^{k\top} x \leq \gamma^{k\top} x^k$ for $i := 0, \dots k$
4:   $k := k + 1$
5: end loop
```

Both methods: Kelley's (Algorithm 5) and Nesterov's (Algorithm 1) are cutting plane methods, as they obtain proper cuts from the oracle, and they both yield optimization subproblems. However, while in the Kelley's method the goal function of the subproblem is the original problem goal function, in the Nesterov's method it is the potential function. In consequence, in the Kelley method, an iterate x^k can lie at the hyperplane of some cut, in ACCPM it is always distanced from all the cut hyperplanes.

7.2. Comments on the efficiency of `SVM_perf`

Joachims analyses his algorithm in the following steps

1. He establishes the maximum number of iterations of the upper-level scheme necessary to achieve accuracy[3] ϵ as $\max\left\{\frac{2}{\epsilon}, \frac{8CR^2}{\epsilon^2}\right\}$ (see Lemma 2 in Joachims, 2006), which in practice (when ϵ is sufficiently small) reduces to

$$\frac{8CR^2}{\epsilon^2}. \tag{5}$$

[3]His accuracy is defined as difference between the current and the optimal value of the goal function.

2. Interesting is the complexity of a single step of the upper level Kelley scheme. In kth iteration of Algorithm 5, he needs to solve the minimization subproblem in step 3. However, this subproblem has a very simple, quadratic goal function and k constraints. Thus he solves, instead, the dual of this problem, that has k variables.[4] The essence of the efficiency `SVM_perf` consists in an assumption that k is essentially smaller than $\tilde{N}$, the number of the variables of the primal subproblem. Later, we shall argue against this assumption, but let us continue the complexity calculations (as in Lemma 1 of Joachims, 2006):
   (a) Joachims assesses the cost of transforming between the primal and the dual as $O(nNk^2)$ (where N and n are understood as before, i.e., as in (I)).
   (b) He says nothing of the cost of solving the dual, we shall denote this cost as $O(\theta(k))$. Function $\theta(x)$ can be in the best case the polynomial $x^{3.5}$ as no better optimization algorithm exists, even for linear problems.
   (c) The cost of the oracle call is also $O(nN)$.

Putting together (5) and the above components of the cost of a step of Algorithm 5 we obtain the following complexity estimate:

$$O\left(\sum_{i=1}^{\lceil \frac{8CR^2}{\epsilon^2} \rceil} (\theta(i) + nNi^2)\right), \tag{6}$$

a rather poor estimate both in C and ϵ. Joachims, however, exposes (6) as the following claim:

For constant C and ϵ the complexity is $O(nN)$.

(compare Theorem 4 in Joachims, 2006). The so expressed result has received much attention in the data mining community.

Joachims in a natural way defines his ϵ and hopes he will not need small ϵ and/or large C in solving practical problems. He tries to support this assumption with experiments.

In the opinion of the author of this paper the assumption is at least risky. First of all, Joachims assumes he will need ϵ and C set up to some levels, which are independent of the number of features N. In the light of (5), this would mean also that the number of iterations, made by Algorithm 5 is independent of N. This already contradicts the common geometric intuition, which says

[4]It is easy to imagine that the geometry of the "primal" subproblem is determined by relative alignments of the cuts. Thus, the essence of its geometry can be included in the k^2 dot products of the k^2 possible pairs of the cut normals. It is then not surprising that the dual problem of k variables, having k^2 elements in the Jacobian matrix of constraints, is equivalent to the primal.

that the number of cuts generated to isolate the solution should depend on the dimensionality N of the space of decision variables. Let us express this with an example.

EXAMPLE 1 *Assume that the number of cuts generated by Algorithm 5 is at least DN, where D is a positive constant. Then only the last iteration of this algorithm costs*

$$O\left(\theta(DN) + nD^2N^3\right) \tag{7}$$

(due to components 2a, 2b, 2c), which is far from linearity in N.

The strong aspect of the Joachims' algorithm is, however, the ability to fully exploit the sparsity of data (matrix A in (2)). If $s\%$ of the coefficients of A are zeros then the complexity of the algorithm (for constant C and ϵ) reduces by $s\%$.

Actually, there are a number of methods in optimization that can give good estimates of complexity in the problem sizes, even like number of constraints times number of variables. Then, however, the theoretical dependence of final accuracy turns out to be poor, also like $1/\epsilon^2$. Such are for example *projection methods* (see, e.g., Bauschke and Borwein, 1996; Kiwiel, 1996). However, their practical dependence on the accuracy is not much better than the theoretical one. (This does not mean they are not useful. For example, their great advantage is lack of the necessity of inverting large matrices, and in large problems, e.g. in image recognition, they may remain the only feasible approach).

7.3. Efficiency of the author's proposal

Since the theoretical complexity bounds of the Nesterov method are far from practical ones and since we have introduced some modifications that could break his sophisticated reasoning, our algorithm so far is difficult to firmly analyze theoretically. Below we give only some indications of how it can behave.

The main tool to analyze Algorithm 1 is the following theorem.

THEOREM 1 *Assume there exist a convex function $\phi : \mathbb{R}^{\tilde{n}} \mapsto \mathbb{R}$ with a unique minimum point $x^\star \in \mathbb{R}$ satisfying $\|x^\star\| \leq R$, L-Lipschitz on $\{x \in \mathbb{R}^{\tilde{n}} : \|x\| < R\}$, such that for every k $\texttt{oracle}(x^k) \in \partial\phi(x^k)$. Then*

$$\phi(x^k) - \phi(x^\star) \leq \frac{c \exp(4/(k+1))LR}{\sqrt{10}} \tag{8}$$

where $c = \frac{\exp(\sqrt{2}-1)}{\sqrt{2}(3-2\sqrt{2}-4/27)} \approx 45.7.$

Proof. See Theorem 4.1 in Nesterov (1995) with $\sigma = 8$, $x_0 = 0$. ■

Remark 2 *Some obstacle is the fact that the Nesterov method was designed for unconstrained optimization, while problem (G) in general has constraints.*

More formally, among problems (E), (I1), (I2) only (I1) gives a clear indication of what function ϕ could be: namely, we can put $\phi = f$. The remaining two problems have constraints, whose subgradients can be returned by the appropriate oracles together with the subgradients of the goal functions. It would be still possible to reconstruct a necessary ϕ, given the sequence x^k and γ^k generated, but such a ϕ would be very artificial and probably the difference $\phi(x^k) - \phi(x^\star)$ would be a poor indicator of how close we are to the solution of our problem. Thus, formally, we can use the Nesterov's results only for problem (3). However, in practice, the variant of our algorithm with problem (3) is even faster than the variant with problem (3), probably due to the earlier oracle returns.

Corollary 1 *One easily verifies that for $\epsilon \equiv \phi - \phi^\star$ the Nesterov method takes at most*

$$O\left(\frac{LR}{\epsilon^2}\right) \tag{9}$$

to achieve accuracy ϵ.

Assumed $\|a_j\|$ in problem (I1) were uniformly bounded by some constant, the Lipschitz constant L in (9) is proportional to C. Thus, the estimates (5) and (9) are similar.

The result (9), of the class $1/\epsilon^2$, is relatively weak. What is interesting about it is its independence of the number of variables $\tilde{N}$. However, we expect a much better practical result. This is because analytic centers proved experimentally to quickly shrink the localization set. Moreover, some other analytic center cutting plane methods, though more sophisticated, give better bounds even in theory. The method of Atkinson and Vaidya (1995) has the theoretical estimate of $O(\tilde{N}\log(1/\epsilon)^2)$.

Assuming that the upper level scheme makes k iterations throughout the optimization run, Nesterov gives the following estimate of the necessary number of arithmetic operations:

$$O(\tilde{N}^2 k + \tilde{N}^{\frac{3}{2}} k^2).$$

If the Nesterov's result were directly applicable to our case (regardless of the modifications we did, like introducing linesearch), we we could expect, taking into account the oracle call cost, which is $O(\tilde{N}\tilde{n})$, the overall number of operations of

$$O(\tilde{N}^2 k + \tilde{N}^{\frac{3}{2}} k^2 + \tilde{N}\tilde{n}) = O(N^2 k + N^{\frac{3}{2}} k^2 + Nn) = O\left(\frac{N^2 LR}{\varepsilon^2} + \frac{N^{\frac{3}{2}} L^2 R^2}{\varepsilon^4} + Nn\right).$$

The disadvantage of the current version of the solver is its difficulty with treating sparse problems. Since the oracles for an inexact separation problems add rows of A when calculating a cut, the cut is usually dense. Even though the algebra used in the solver (in particular to update the inverses of the Hessian) is sparse, the dense cuts quickly make the Hessian inverse representation dense. In practice, this results in intractability of tasks with large number N of features. However, many practical tasks have a moderate number of features: in data bases there are not many attributes in the tables, many tasks regarding signal processing (like some examples in the experiments below) may lead to a moderate N. Many of them (like in signal precessing, network traffic data in automatic failure detection, like problems constructed using data tables with numeric attributes) are dense problems and sparse techniques of any solver would not not help. However, we left the sparse algebra in the implementation to ease the future development of the solver. There are in general two possible ways for effectively invert the Hessian:

1. Use low-rank approximations of the Hessian. This approach, taken, e.g., in Fine and Scheinberg (2001), yields a risk of slowing down the solution of the optimization subproblem in step 4 of Algorithm 1 by spoiling the mechanism of minimizing a self-concordant function because of inaccurate knowledge of the Hessian.
2. Change the reformulations (I1), (I2) for oracles to generate more sparse cuts. In a cutting plane method we can shallow the cuts even by artificially imposing zeros on some their components. This is the direction of the further author's research.

8. Experiments

Numerous experiments were performed in order to investigate various aspects of the algorithm. We used an own artificial example, the real-life examples from UCI KDD archives[5]: `covtype`, `physics`, `biology`, `eeglarge`, suitably adjusted, and an own real example `switch` regarding automatic recognition of failures in computer networks (see Białoń et al., 2004). All the problems were fairly or completely dense. The experiments were performed on AMD Opteron 880 Dual Core, 2390 MHz, 1024 KB cache, with 8 GB RAM. Here we give only a flavor of the results.

Memory economy. To validate our algorithm as a „real data processing algorithm" that does not simultaneously store all the data in RAM, we successfully solved an artificial, randomly generated, fully separable problem with $n = 1,000,000,000$ and $N = 10$. It took 20h 03m. The attained high accuracy of 99.9917% on the training set shows solver robustness.

Effectiveness of accelerating mechanisms. We used a similar (but not fully separable) artificial example with $n = 20000$, $N = 20$. We put $C = 10$. Switching on the accelerating mechanism 1 means using the algorithm variant

[5]http://kdd.ics.uci.edu/

with (I2), switching it off – with (I1). Switching off the accelerating mechanism 2 was realized by using only one bucket and giving up bucket dispatching.

Table 1. Efficiency of the accelerating mechanisms

Mechanisms	1 on, 2 on	1 off, 2 on	1 on, 2 off	1 off, 2 off
real time (s)	56	423	104	371
accuracy (training set)	86.50%	88.64%	88.27%	88.64%

As we see in Table 1, the mechanisms together were able to speed solving even several times over (in all other experiments – not less than two times). However, mechanism 1 with its complex reformulation of the SVM problem may introduce some inaccuracy in the final solution, possibly resulting in slightly lowering the separation accuracy. This does not always happen (especially it does not for very well separable problems).

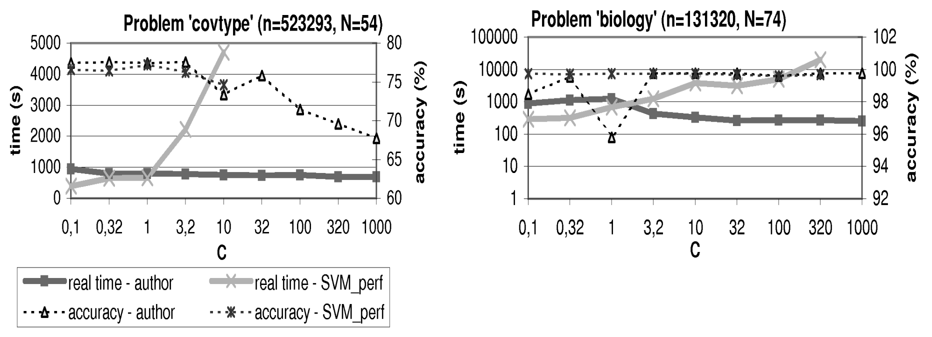

Figure 1. Reaction of the solvers to an increase in C

Real examples, comparison to `SVM_perf`, and influence of input parameters. We compared the behavior of our solver and `SVM_perf` on real life examples. In our solver we emulated bucket files in RAM, to have a more fair competition. We did experiments for various settings of C. The final solution accuracy ϵ for `SVM_perf` was varied during experiments. However, in the author's solver there is no direct access to ϵ. The stopping criterion uses, instead, the sum of minus logarithms of distances of current iterate from cuts divided by the number k of current iteration. Thus the accuracies are hardly comparable, but in the authors' solver the solution accuracy was kept constant but tight[6], believably tighter than any settings used for `SVM_perf`.

[6]More precisely, the stop test was $(F^k(x^k) - \frac{k}{2R})/k \geq 12$, with the additional remark that cuts generated by oracle are normalized. So this expression is a good measure of the average distance of x^k from the cuts hyperplanes.

The most interesting behavior of solvers under increasing C and with $\epsilon = 0.1$ is shown in Fig. 1.

The main conclusion from the experiments is as follows. Our solver is usually slower for small and well-conditioned tasks (e.g., with low C). However, for tighter parameter settings the situation dramatically changes. `SVM_perf` practically gets stuck even with a moderate increase of C (and so gets with a moderate decrease of ϵ). Then, our solver clearly outperforms `SVM_perf` or even remains the only one capable of solving the problem. Some 'stucking' settings were obviously not too heavy (they did not essentially deteriorate the attained classification accuracy). Other examples gave a similar behavior. More experiment results are given in Table 2.

9. Conclusions

Below we list the conclusions from our work.

1. The main goal of the work, which is the tractability of a large number of examples, turned out to be achieved in the experimental investigation.
2. Also, the accelerating mechanisms seem to be useful.
3. The Nesterov method seems to have much better practical behavior in terms of the number of iterations than its theoretical bounds.
4. The theoretical anxiety about the reaction of `SVM_perf` on large C an low ϵ is apparently supported by experiments. Still, the experiments could not clearly determine whether such hard settings of C and ϵ are necessary to obtain better classifiers. However, Example 1 gives a strong argument that they may be necessary (since the necessary tightness of their settings may depend on the number of features N).
5. Since for loose settings (and also for large N in sparse problems) `SVM_perf` outperforms our solver, both solvers can be used as complementary, in diverse situations.
6. Describing the `SVM_perf` algorithm as linear in N and n seems to be a clear simplification. Actually there appears to be a very general trade-off in optimization: either an algorithm has a poor complexity in the accuracy and is good in some other properties (low complexity in problem sizes, easy algebra and sparsity handling) or vice versa. In `SVM_perf` the dependence on ϵ and C seems to be strong.
7. The possible areas of application of our solver, determined mainly by the desired sizes n and N are: mining in databases, automatic failure detection in computer networks, some tasks of signal processing.
8. The most urgent further work for our solver is to explore sparsity by enforcing sparse cuts.
9. A broader use of interior point methods in machine learning/data mining would be beneficial. Though more sophisticated, they allow to gain an honestly quick convergence, preserving robustness to some parameter settings.

Table 2. Real life experiment results.

	covtype (n=523293, N=54)								
	C=0.1				C=10			C=1000	
	author	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.1)*
real time (s)	1572	3831	2224	384	1510	>39135	4708	1453	2739
iterations	774	302	236	122	747	> 1477	310	716	183
file gets+puts	4,9E+08	-	-	-	4,8E+08	-	-	4,6E+08	-
acc, test set	77.58%	76.55%	76.55%	76.56%	77.57%	*	74.64%	77.57%	63.54%
acc, train set	77.40%	76.48%	76.49%	76.54%	77.42%	*	74.65%	77.40%	63.69%

	biology (n=131320, N=74)								
	C=0.1				C=10			C=1000	
	author	*SVM_perf (ε=0.0001)*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.1)*
real time (s)	1866	>2h	1118	287	1817	>2h	3854	1840	7915
iterations	1673	*	1027	551	1604	-	1247	1632	2203
file gets+puts	2,5E+08	-	-	-	2,4E+08	-	-	2,5E+08	-
acc, test set	99.75%	*	99.71%	99.72%	99.75%	*	99.74%	99.74%	83.79%
acc, train set	99.73%	*	99.67%	99.68%	99.73%	*	99.72%	99.73%	83.72%

	physics (n=49460, N=78)								
	C=0.1				C=10			C=1000	
	author	*SVM_perf (ε=0.0001)*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.1)*
real time (s)	495	>2h	4479	485	530	>2h	>2h	519	>2h
iterations	1011	*	420	83	1023	*	*	1017	*
file gets+puts	6,2E+07	-	-	-	6,2E+07	-	-	6,2E+07	-
acc, test set	70.93%	*	67.59%	68.33%	71.30%	*	*	71.11%	*
acc, train set	70.85%	*	69.63%	68.83%	70.82%	*	*	70.88%	*

	switch (n=1570, N=26)								
	C=0.1				C=10			C=1000	
	author	*SVM_perf (ε=0.0001)*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.1)*	*SVM_perf (ε=0.001)*	*author*	*SVM_perf (ε=0.1)*
real time (s)	78.42	4.29	2.69	0.22	116.05	0.28	1.31	87.84	0.24
iterations	720	55	46	25	719	72	141	674	100
file gets+puts	7,3E+05	-	-	-	8,0E+05	-	-	6,5E+05	-
acc, test set	99.49%	99.06%	99.06%	99.06%	99.81%	99.82%	99.81%	99.81%	99.82%
acc, train set	99.43%	98.66%	98.66%	98.66%	100%	100%	100%	100%	100%

	eeglarge (n=600, N=600)								
	C=0.1				C=10			C=1000	
	author	*SVM_perf (ε=0.0001)*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.001)*	*SVM_perf (ε=0.1)*	*author*	*SVM_perf (ε=0.1)*
real time (s)	12533	>67117	26417	878.5	22020	>43536	27389	20140	>78485
iterations	4946	> 1163	1070	754	5080	>1561	1464	5106	>979
file gets+puts	3,7E+06	-	-	-	3,7E+06	-	-	3,6E+06	-
acc, test set	55.3333	*	55.17	54.83	57.1667	*	57.5	57.8333	*
acc, train set	79	*	75.33	74.17	89.1667	*	86.33	91.1667	*

Appendix: Lower optimization levels

Here we summarize the activities in the lower levels of our solver (below the ACCPM scheme – Algorithm 1). In step 4 of Algorithm 1 we need to solve the following minimization subproblem:

$$\text{(G)}\quad \underset{x\in\tilde{N}}{\text{minimize}}\, F^k(x) \quad \text{subject to} \quad x \in \text{dom}F^k \tag{10}$$

This is done by Algorithm 6.

Algorithm 6 Solving the minimization subproblem

```
   Given: starting point $x^k = 0$, the last generated cut gradient $\gamma^k$, the
   current approximation $\hat{H}$ of the Hessian inverse of $F^k$ at $x^k$.
1: (find a feasible point) $x^k := \arg\min_{t>0,\ (x^k - t\hat{H}\gamma^k)\in \text{dom}F^k} F^k(x^k - t\hat{H}\gamma^k)$.
   Update the Hessian inverse approximation:
   update($\hat{H}$).
2: repeat
3:     Compute new iterate: $x^k := \arg\min_{t\geq 0,\ (x^k - t\hat{H}\nabla F^k(x^k))\in \text{dom}F^k} F^k(x^k - t\hat{H}\nabla F^k(x^k))$
4:     update($\hat{H}$)
5: until $\nabla {F^k}^\top \hat{H} \nabla F^k \leq 0.1$
```

A line search is used in steps 1 and 3, contrary to majority of interior-point implementations that use a pure damped Newton method. It is done by an authors' line search procedure. This is an experiment to more accurately minimize the function in order to avoid the influence of possible round-off errors under very tightly aligned cuts and thus – to increase the solver robustness and ability to give very accurate solutions.

The stop test is the commonly used *approximate analytic center condition* (see Nesterov, 1995).

The procedure `update` is the commonly used procedure (e.g. in Karmarkar, 1981; Nesterov, 1995) of approximating the Hessian inverse, by lowering the number of rank-one matrix updates/downdates.

The rank-one update or downdate (i.e, computing $(A + \xi\xi^\top)^{-1}$ or $(A - \xi\xi^\top)^{-1}$ having the inverse of a positively definite $m \times m$ matrix A and a column vector $\xi \in \mathbb{R}^m$) in the work of Nesterov is done by the Sherman-Morrison-Woodbury formula. We, instead, use the CHOLMOD package[7] with its pro-

[7] We also have an own procedure inverting matrices, alternative to CHOLMOD. It uses the representation of the Hessian matrix as a product of dilation matrices. This procedure was used for experiments given in Fig. 1, whereas in other shown experiments CHOLMOD was used

cedure based on supernodal sparse Cholesky decomposition (see Davis and Hager, 1999, 2007). However, so far the potential of sparse matrix operations in CHOLMOD is not fully exploited, since the cuts generated by the oracle for problems (I1) and (I2) tend to be dense even if the matrix A in (I) is sparse. Exploiting the sparsity of A is the subject of current author's research.

The vector storage, oracle and the bucketing procedure, the data and bucket files are implemented as sparse.

References

Atkinson, D.S. and Vaidya, P.M. (1995) A cutting plane algorithm that uses analytic centers. *Mathematical Programming*, Series B, **69**, 1-43.

Bauschke, H. and Borwein, J(1996) On projection algorithms for solving convex feasibility problems. *SIAM Review* **38** (3), 367-426.

Białoń, P., Chudzian, C., Florek, J. and Majdan, M. (2004) Przetwarzanie informacji i modelowanie w środowiskach rozproszonych (in Polish). Technical Report 06.30.002.4, National Institute of Telecommunications, Warsaw, Poland.

Bordes, A. and Bottou, L. (2005) The Huller: a simple and efficient online SVM. *Machine Learning: ECML 2005, Lecture Notes in Artificial Intelligence*. Springer Verlag, 505-512.

Davis, T.A. and Hager W.W. (1999) Modifying a sparse Cholesky Factorization. *SIAM J. Matrix Anal. Appl.* **20** (3), 606-627.

Davis, T.A. and Hager W.W. (2007) *User Guide for CHOLMOD: a sparse Cholesky factorization and modification package.* Dept. of Computing and Information Science and Engineering, Univ.- of Florida, Gainesville, FL, Version 1.5.

Ferris, M. and Munson, T. (2003) Interior-point methods for massive support vector machines. *SIAM J. Optimization*, **13** (3), 783-804.

Fine, S. and Scheinberg, K. (2001) Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research* **2**, 243-264.

Goffin J.L. and Vial, J.P. (1999) Convex Nondifferentiable optimization: a survey focused on the analytic center cutting plane method. HEC/Logilab Technical Report 99.02.

Joachims, T. (1998) Making large-scale support vector machine learning practical. In: B. Scholköpf et al., eds., *Advances in Kernel Methods Support Vector Learning.* MIT Press, Cambridge, 148-168.

Joachims, T. (2006) Training linear SVMs in linear time. *Proceedings of the 12th ACM Conference on Knowledge Discovery and Data Mining (KDD)*, Philadelphia, USA. ACM Press, New York, 217–226.

Karmarkar, N. (1984) A new polynomial-time algorithm for linear programming. *Combinatorica* **4**, 373-395.

Kelley, J. (1960) The cutting plane method for solving convex programs. *Journal of the Society for Industrial Applied Mathematics* **8**, 703-712.

KIWIEL, K.C. (1996) The efficiency of subgradient projection methods for convex optimization, Part I: General level methods. *SIAM Control Optim.*, **34** (2), 660-676.

MUSICANT, D.R. (2000) Data Mining via Mathematical Programing and Machine Learning. Ph.D. thesis. University of Wisconsin, Madison.

NESTEROV, YU. (2005) Complexity estimates of some cutting plane methods based on the analytic barrier. *Mathematical Programming* **69**, 149-176.

PLATT, J. (1998) Fast training support vector machines using sequential minimal optimization. In: B. Scholköpf et al., eds., *Advances in Kernel Methods Support Vector Learning.* MIT Press, 185-208.

SCHEINBERG, K. (2006) An efficient implementation of an active set method for SVMs. *Journal of Machine Learning Research* **7**, 2237-2257.

TERLAKY, T. (ed.) (1996) *Interior Point Methods of Mathematical Programming.* Kluwer, Dordrecht.

VANDENBERGHE, L. and COMANOR, K. (2003) A sequential analytic centering approach to the support vector machine. *Proceedings of SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, San Diego, USA. SPIE - International Society for Optical Engineering, 209-218.

VAPNIK, V.N. (1995) *The Nature of Statistical Learning Theory.* Springer, New York.