Optimized Cutting Plane Algorithm for Support Vector Machines

Vojtěch Franc VOJTECH.FRANC@FIRST.FRAUNHOFER.DE
Soeren Sonnenburg SOEREN.SONNENBURG@FIRST.FRAUNHOFER.DE
Fraunhofer Institute FIRST, Kekulestr. 7, 12489 Berlin, Germany

Abstract

We have developed a new Linear Support Vector Machine (SVM) training algorithm called OCAS. Its computational effort scales linearly with the sample size. In an extensive empirical evaluation OCAS significantly outperforms current state of the art SVM solvers, like SVM$^{\text{light}}$, SVM$^{\text{perf}}$ and BMRM, achieving speedups of over 1,000 on some datasets over SVM$^{\text{light}}$ and 20 over SVM$^{\text{perf}}$, while obtaining the same precise Support Vector solution. OCAS even in the early optimization steps shows often faster convergence than the so far in this domain prevailing approximative methods SGD and Pegasos. Effectively parallelizing OCAS we were able to train on a dataset of size 15 million examples (itself about 32GB in size) in just 671 seconds — a competing string kernel SVM required 97,484 seconds to train on 10 million examples subsampled from this dataset.

1. Introduction

Many applications in e.g. Bioinformatics, IT-Security and Text-Classification come with *huge* amounts (e.g. millions) of data points, which are indeed *needed* to obtain state-of-the-art results. They therefore require computationally extremely efficient methods capable of dealing with ever growing data sizes. Support Vector Machines (SVM) e.g. (Cortes & Vapnik, 1995; Cristianini & Shwawe-Taylor, 2000) have proven to be powerful tools for a wide range of different data analysis problems. Given labeled training examples $\{(\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_m, y_m)\} \in (\mathbb{R}^n \times \{-1, +1\})^m$ and a regularization constant $C > 0$ they learn a linear classification rule $h(\boldsymbol{x}) = \operatorname{sgn}(\langle \boldsymbol{w}^*, \boldsymbol{x} \rangle + b^*)$ by solving the quadratic SVM primal optimization problem (P) or its dual formulation (D) allowing the use of *kernels*.

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

$$\textbf{(P)} \quad \min_{\boldsymbol{w},\boldsymbol{\xi},b} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i, \text{for } \boldsymbol{w} \in \mathbb{R}^n, \boldsymbol{\xi} \in \mathbb{R}^m_+, b \in \mathbb{R}$$
$$\text{s.t.} \quad y_i\left(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b\right) \geq 1 - \xi_i, \quad i = 1, \dots, m$$

$$\textbf{(D)} \max_{\boldsymbol{\alpha}\in\mathbb{R}^m} \quad \textstyle\sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$$
$$\text{s.t.}: \quad \textstyle\sum_{i=1}^m \alpha_i y_i = 0, \; 0 \leq \alpha_i \leq \frac{C}{m}, \; i = 1 \dots m$$

Due to the central importance of SVMs, many techniques have been proposed to solve the SVM problem. As in practice only limited precision solutions to (P) and (D) can be obtained they may be categorized into *approximative* and *accurate*.

Approximative Solvers make use of heuristics (e.g. learning rate, number of iterations) to obtain (often crude) approximations to the QP-solution. They have very low per-iteration cost and low total training time. Especially for large scale problems, they are claimed to be sufficiently precise while delivering the best performance vs. training time trade-off (Bottou & Bousquet, 2008), which may be attributed to the robust nature of *large margin* SVM solutions. However while they are fast in the beginning they often fail to achieve precise solution. Among the to-date most efficient solvers are Pegasos (Shwartz et al., 2007) and SGD (Bottou & Bousquet, 2008), which are based on stochastic (sub-)gradient descent.

Accurate Solvers In contrast to approximative solvers, accurate methods solve a QP up to a given precision ε, where ε commonly denotes the violation of the relaxed KKT conditions (Joachims, 1999) or the (relative) duality gap. Accurate methods often have good asymptotic convergence properties, and thus for small ε converge to very precise solutions being limited only by numerical precision. Classical examples are off-the-shelf optimizers (e.g. MINOS, CPLEX, LOQO). However it is usually infeasible to use standard optimization tools for solving the SVM training problems (D) on datasets containing more than a few thousand examples. So-called decomposition techniques as chunking (e.g. used in (Joachims, 1999)), or SMO (used in

(Chang & Lin, 2001)) overcome this limitation by exploiting the special structure of the SVM problem. The key idea of decomposition is to freeze all but a small number of optimization variables (*working set*) and to solve a sequence of constant-size problems (subproblems of the SVM dual). While decomposition based solvers are very flexible as they are working in the dual and thus allow the use of kernels they become computationally intractable with a few hundred thousand examples. This limitation can be explained as follows: Decomposition methods exploit the fact that the optimal solution of (P) does not change if inactive constraints at the optimum are removed, they are therefore only efficient if the number of active constraints is reasonably small. Unfortunately, the number of active constraints is lower bound by the portion of misclassified examples, which is proportional to the number of examples m. Thus decomposition methods are computationally prohibitive for large-scale problems (empirically about 10%-30% of the training points become active constraints).

This poses a challenging task for even current state-of-the-art SVM solvers such as SVM$^{\text{light}}$ (Joachims, 1999), Gradient Projection-based Decomposition Technique-SVM (GPDT-SVM) (Zanni et al., 2006), LibSVM (Chang & Lin, 2001). As improving training times using the dual formulation is hard, the research focus has shifted back to the original SVM primal problem. The importance of being able to efficiently solve the primal problem for large datasets is documented by a number of very recently developed methods, e.g. SVMLin (Sindhwani & Keerthi, 2007; Chapelle, 2007), LibLinear (Lin et al., 2007), SVM$^{\text{perf}}$ (Joachims, 2006) and BMRM (Teo et al., 2007).

In the following we will focus on finding *accurate* solutions of the unconstrained linear SVM primal problem[1]

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} F(\boldsymbol{w}) := \left[\tfrac{1}{2}\|\boldsymbol{w}\|^2 + CR(\boldsymbol{w})\right], \quad (1)$$

$$\text{where} \quad R(\boldsymbol{w}) = \tfrac{1}{m}\textstyle\sum_{i=1}^m \max\{0, 1 - y_i\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle\} \quad (2)$$

is a convex risk approximating the training error.

Among the up to date most efficient *accurate* SVM primal problem (1) solvers are the Cutting Plane Algorithm (CPA) based methods put forward in (Joachims, 2006; Teo et al., 2007) and implemented in SVM$^{\text{perf}}$ and BMRM. The idea of CPAs is to approximate the risk R by a piece-wise linear function defined as the maximum over a set of linear under-estimators, in CPA terminology called *cutting planes*. In (Joachims, 2006; Teo et al., 2007) it was shown that their number does not depend on the number of training examples m and that very few such cutting planes are needed in practice to sufficiently approximate (1).

[1]Note that we focus on the linear rule without a bias. The bias can be included by adding a constant feature to each training example $\boldsymbol{x}_i$.

In this work we propose a new method, called the Optimized Cutting Plane Algorithm for SVMs (OCAS). We empirically show that OCAS converges on a wide variety of large-scale datasets even considerably faster than SVM$^{\text{perf}}$, BMRM and SVM$^{\text{light}}$, achieving speedups of several orders of magnitude on some problems. We also demonstrate that OCAS even in the early optimization steps shows faster convergence than the so far in this domain dominating approximative methods. Finally we critically analyze all solvers w.r.t. classification performance in an extensive model selection study.

The report is organized as follows. CPA is described in Section 2. In Section 3, we point out a source of inefficiency of CPA and propose a new method, OCAS, to alleviate the problem and prove linear convergence. An extensive empirical evaluation is given in Section 4 and concludes the paper.

2. Cutting Plane Algorithm

Recently, the Cutting Plane Algorithm (CPA) based large-scale solvers, SVM$^{\text{perf}}$ (Joachims, 2006) and BMRM (Teo et al., 2007), have been proposed. SVM$^{\text{perf}}$ implements CPA specifically for the linear SVM problem (1). Decoupling regularizer and loss function, BMRM generalizes SVM$^{\text{perf}}$ to a wide range of losses and regularizers making it applicable to many machine learning problems, like classification, regression, structure learning etc. It should be noted that BMRM using the two norm regularizer $\|.\|_2$ and hinge loss (i.e. SVM problem (1)) coincides with SVM$^{\text{perf}}$. It was shown that SVM$^{\text{perf}}$ and BMRM by far outperform the decomposition methods like SVM$^{\text{light}}$ on large-scale problems. The rest of this section describes the idea behind CPA for the standard SVM setting (1) in more detail.

In CPA terminology, the original problem (1) is called the master problem. Using the approach of (Teo et al., 2007) one may define a reduced problem of (1) which reads

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{w}} F_t(\boldsymbol{w}) := \left[\frac{1}{2}\|\boldsymbol{w}\|^2 + CR_t(\boldsymbol{w})\right]. \quad (3)$$

Problem (3) is obtained from the master problem (1) by substituting a piece-wise linear approximation R_t for the risk R while leaving the regularization term unchanged, i.e. only the complex part of the objective F is approximated. The approximation R_t is derived as follows. Since the risk R is a convex function, it can be approximated at any point $\boldsymbol{w}'$ by a linear under estimator

$$R(\boldsymbol{w}) \geq R(\boldsymbol{w}') + \langle \boldsymbol{a}', \boldsymbol{w} - \boldsymbol{w}'\rangle, \qquad \forall \boldsymbol{w} \in \mathbb{R}^n, \quad (4)$$

where $\boldsymbol{a}'$ is any subgradient of R at the point $\boldsymbol{w}'$. We will use a shortcut $b' = R(\boldsymbol{w}') - \langle \boldsymbol{a}', \boldsymbol{w}'\rangle$ to abbreviate (4) as $R(\boldsymbol{w}) \geq \langle \boldsymbol{a}', \boldsymbol{w}\rangle + b'$. In CPA terminology, $\langle \boldsymbol{a}', \boldsymbol{w}\rangle + b' = 0$

is called a cutting plane. A subgradient $\boldsymbol{a}'$ of R at the point $\boldsymbol{w}'$ can be obtained as

$$\boldsymbol{a}' = -\frac{1}{m}\sum_{i=1}^{m} \pi_i y_i \boldsymbol{x}_i , \quad \pi_i = \begin{cases} 1 & \text{if} \quad y_i\langle \boldsymbol{w}', \boldsymbol{x}_i\rangle \leq 1 \,, \\ 0 & \text{if} \quad y_i\langle \boldsymbol{w}', \boldsymbol{x}_i\rangle > 1 \,. \end{cases} \tag{5}$$

To get a better approximation of the risk R than a single cutting plane, one may use a collection of cutting planes $\{\langle \boldsymbol{a}_i, \boldsymbol{w}\rangle + b_i = 0 \mid i = 1, \ldots, t\}$ at t distinct points $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t\}$ and take their point-wise maximum

$$R_t(\boldsymbol{w}) = \max\big\{0, \max_{i=1,\ldots,t} \big(\langle \boldsymbol{a}_i, \boldsymbol{w}\rangle + b_i\big)\big\} \,. \tag{6}$$

The zero cutting plane is added to the maximization as the risk R is always greater or equal to zero. It follows directly from (4) that the approximation R_t lower bounds R and thus also F_t lower bounds F.

To select the cutting planes, CPA starts from $t = 0$ (no cutting plane) and then it iterates two steps:

1. Compute $\boldsymbol{w}_t$ by solving the reduced problem (3), which can be cast as a standard QP with t variables.
2. Add a new cutting plane $(\boldsymbol{a}_{t+1}, b_{t+1})$ to approximate the risk R at the current solution $\boldsymbol{w}_t$.

A natural stopping condition for CPA is based on evaluating the ε-optimality condition $F(\boldsymbol{w}_t) - F_t(\boldsymbol{w}_t) \leq \varepsilon$ which, if satisfied, guarantees that $F(\boldsymbol{w}_t) - F(\boldsymbol{w}^*) \leq \varepsilon$ holds. [2] (Joachims, 2006) proved that for arbitrary $\varepsilon > 0$ CPA converges to the ε-optimal solution after $\mathcal{O}(\frac{1}{\varepsilon^2})$ iterations, i.e. it does not depend on the number of examples m. An improved analysis of the CPA published recently (Teo et al., 2007) shows that the number of iterations scales only with $\mathcal{O}(\frac{1}{\varepsilon})$. More important, in practice CPA usually requires only tens of iterations to reach a sufficiently precise solution.

3. Optimized Cutting Plane Algorithm for SVMs (OCAS)

We first point out a source of inefficiency appearing in CPA and then propose a new method to alleviate the problem.

CPA selects a new cutting plane such that the reduced problem objective function $F_t(\boldsymbol{w}_t)$ monotonically increases with w.r.t. the number of iterations t. However, there is no such guarantee for the master problem objective $F(\boldsymbol{w}_t)$. Even though it will ultimately converge to the minimum $F(\boldsymbol{w}^*)$, its value can heavily fluctuate between iterations. The reason for these fluctuations is the following. CPA selects at each iteration t the cutting plane which perfectly approximates the master objective F *at the current solution* $\boldsymbol{w}_t$. However, there is no guarantee that such cutting plane will be an active constraint in the vicinity of the optimum $\boldsymbol{w}^*$, nor must the new solution $\boldsymbol{w}_{t+1}$ of the reduced problem improve the master objective. In fact it often occurs that $F(\boldsymbol{w}_{t+1}) > F(\boldsymbol{w}_t)$.

[2]An alternative stopping condition advocated in (Joachims, 2006) halts the algorithm when $R(\boldsymbol{w}_t) - R_t(\boldsymbol{w}_t) \leq \hat{\varepsilon}$. It can be seen that both the stopping conditions become equivalent if we set $\varepsilon = C\hat{\varepsilon}$.

To speed up the convergence of CPA, we propose a new method which we call the Optimized Cutting Plane Algorithm for SVMs (OCAS). Unlike standard CPA, OCAS aims at simultaneously optimizing the master and reduced problem's objective functions F and F_t, respectively. In addition, OCAS tries to select such cutting planes that have higher chance to actively contribute to the approximation of the master objective function F around the optimum $\boldsymbol{w}^*$. In particular, we propose the following three changes to CPA.

Change 1 We maintain the best so far solution $\boldsymbol{w}_t^{\mathrm{b}}$ obtained during the first t iterations, i.e. $F(\boldsymbol{w}_1^{\mathrm{b}}), \ldots, F(\boldsymbol{w}_t^{\mathrm{b}})$ forms a monotonically decreasing sequence.

Change 2 The new best so far solution $\boldsymbol{w}_t^{\mathrm{b}}$ is found by searching along a line starting at the previous best solution $\boldsymbol{w}_{t-1}^{\mathrm{b}}$ crossing the reduced problem's solution $\boldsymbol{w}_t$, i.e. ,

$$\boldsymbol{w}_t^{\mathrm{b}} = \min_{k \geq 0} F(\boldsymbol{w}_{t-1}^{\mathrm{b}}(1-k) + \boldsymbol{w}_t k) \,, \tag{7}$$

which can be solved exactly in $\mathcal{O}(m \log m)$ time (see Appendix A).

Change 3 The new cutting plane is selected to approximate the master objective F at a point $\boldsymbol{w}_t^{\mathrm{c}}$ which lies in a vicinity of the best so far solution $\boldsymbol{w}_t^{\mathrm{b}}$. In particular, the point $\boldsymbol{w}_t^{\mathrm{c}}$ is computed as

$$\boldsymbol{w}_t^{c} = \boldsymbol{w}_t^{\mathrm{b}}(1-\lambda) + \boldsymbol{w}_t\lambda \,, \tag{8}$$

where $\lambda \in (0,1]$ is a prescribed parameter. Having the point $\boldsymbol{w}_c$, the new cutting plane is computed using Equation (5) such that $F(\boldsymbol{w}_t^{\mathrm{c}}) = F_{t+1}(\boldsymbol{w}_t^{\mathrm{c}})$. Note that although the theoretical bound on the number of iterations (see Theorem 1) does not depend on λ its value has impact on the convergence speed in practice. We found that the value $\lambda = 0.1$ works consistently well in all experiments.

Algorithm 1 describes the proposed OCAS. Figure 1 shows the impact of the proposed changes to the convergence. OCAS generates a monotonically decreasing sequence of master objective values $F(\boldsymbol{w}_1^{\mathrm{b}}), \ldots, F(\boldsymbol{w}_t^{\mathrm{b}})$ and a monotonically strictly increasing sequence of reduced objective values $F_1(\boldsymbol{w}_1), \ldots, F_t(\boldsymbol{w}_t)$. Similar to CPA, a natural stopping condition for OCAS reads

$$F(\boldsymbol{w}_t^{\mathrm{b}}) - F_t(\boldsymbol{w}_t) \leq \varepsilon \,, \tag{9}$$

where $\varepsilon > 0$ is a prescribed precision parameter. Satisfying the condition (9) guarantees that $F(\boldsymbol{w}_t^{\mathrm{b}}) - F(\boldsymbol{w}^*) \leq \varepsilon$ holds.

Algorithm 1 Optimized Cutting Plane Algorithm

```
1: Set $t = 0$ (i.e. there is no cutting plane at the beginning) and $\boldsymbol{w}_0^{\rm b} = \boldsymbol{0}$.
2: repeat
3:     Compute $\boldsymbol{w}_t$ by solving the reduced problem (3).
4:     Compute a new best so far solution $\boldsymbol{w}_t^{\rm b}$ using the line-search (7).
5:     Add a new cutting plane which approximates the risk $R$ at the point $\boldsymbol{w}_t^{\rm c}$ given by (8), i.e. ,
           $\boldsymbol{a}_{t+1} = -\frac{1}{m}\sum_{i=1}^m \pi_i y_i \boldsymbol{x}_i$ ,
           $b_{t+1} = R(\boldsymbol{w}_t^{\rm c}) - \langle \boldsymbol{a}_{t+1}, \boldsymbol{w}_t^{\rm c}\rangle$ ,
           where $\pi_i = \begin{cases} 1 & \text{if} \quad y_i\langle \boldsymbol{w}_t^{\rm c}, \boldsymbol{x}_i\rangle \leq 1 \,, \\ 0 & \text{if} \quad y_i\langle \boldsymbol{w}_t^{\rm c}, \boldsymbol{x}_i\rangle > 1 \,. \end{cases}$
6:     $t := t + 1$
7: until a stopping condition is satisfied
```

Theorem 1 *For any $\varepsilon > 0$, $C > 0$, $\lambda \in (0, 1]$, and any training set $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_m, y_m)\}$, Algorithm 1 satisfies the stopping condition (9) after at most*

$$\max\left\{\frac{2C}{\varepsilon}, \frac{8C^3Q^2}{\varepsilon^2}\right\} , \tag{10}$$

iterations where $Q = \max_{i=1,\dots,m}\|\boldsymbol{x}_i\|$.

Proof The proof is along the lines of the convergence analysis of the standard CPA (Joachims, 2006). First, it can be shown that violated condition (9) guarantees that adding a new cutting plane $(\boldsymbol{a}_t, b_t)$ leads to an improvement of the reduced objective $\Delta_t = F_{t+1}(\boldsymbol{w}_{t+1}) - F_t(\boldsymbol{w}_t)$ which is not less than $\min\left\{\frac{\varepsilon}{2}, \frac{\varepsilon^2}{8Q^2}\right\}$. Second, by exploiting that $0 \leq F_t(\boldsymbol{w}_t) \leq F(\boldsymbol{w}^*)$ and $F(\boldsymbol{w}^*) \leq F(\boldsymbol{0}) = C$ one can conclude that the sum of improvements $\sum_{i=0}^t \Delta_t$ cannot be greater than C. Combining these two results gives immediately the bound (10). For more details we refer to our technical report (Franc & Sonnenburg, 2007).

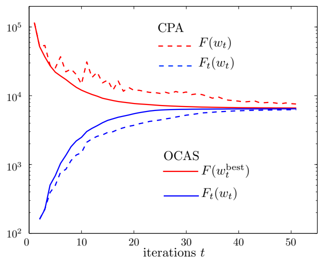

Figure 1. Convergence behaviour of the standard CPA vs. OCAS.

The bound on the maximal number of iterations of OCAS coincides with the bound for CPA given in (Joachims, 2006). Despite the same theoretical bounds, in practice OCAS converges significantly faster compared to CPA (cf. Table 2 in the experiments section).

3.1. Time Complexity and Parallelization

By Theorem 1 the number of iterations of OCAS does not depend on the number of examples m. Hence the overall time complexity is given by the effort required per iteration which is $\mathcal{O}(mn + m\log m) \approx \mathcal{O}(mn)$ (in practice $\log(m) \ll n$, where n is the dimensionality of the data). The per-iteration complexity of the subtasks and the way how they can be effectively parallelized is detailed below:

Output computation involves computation of the dot products $\langle \boldsymbol{w}_t, \boldsymbol{x}_i\rangle$, $i = 1, \dots, m$, which requires $\mathcal{O}(s)$ time, where s equals the number of non-zero elements in the training examples. Distributing the computation equally to p processor leads to $\mathcal{O}(\frac{s}{p})$ time.

Line-search The dominant part is sorting $|K|$ numbers ($K \leq m$, see Appendix A for details) which can be done in $\mathcal{O}(|K|\log|K|)$. A speedup can be achieved by parallelizing the sorting function to using p processors, reducing complexity to $\mathcal{O}\big(\frac{|K|\log|K|}{p}\big)$. Note that our implementation of OCAS uses quicksort, whose worst case complexity is $\mathcal{O}(|K|^2)$, although its *expected* run-time is $\mathcal{O}(|K|\log|K|)$.

Cutting plane computation The dominant part requires computing the sum $-\frac{1}{m}\sum_{i=1}^m \pi_i y_i \boldsymbol{x}_i$ which can be done in $\mathcal{O}(s_\pi)$, where s_π is the number of non-zero elements in the training examples for which π_i is non-zero. Using p processors leads to $\mathcal{O}(\frac{s_\pi}{p})$ time.

Reduced problem The size of the reduced problem (3) is upper bound by the number of iterations which is invariant against the dataset size, hence it requires $\mathcal{O}(1)$ time. Though solving the reduced problem cannot be easily parallelized, it does not constitute the bottleneck as the number of iterations required in practice is small (cf. Table 2).

4. Experiments

We now compare current state-of-the-art SVM solvers (SGD, Pegasos, SVM$^{\rm light}$, SVM$^{\rm perf}$, BMRM[3] on a variety of datasets with the proposed method (OCAS) using 5 carefully crafted experiments measuring:

1. Training time and objective for optimal C
2. Speed of convergence (time vs. objective)
3. Time to perform a full model selection
4. Scalability w.r.t. dataset size
5. Effects of parallelization

To this end we implemented OCAS and the standard CPA[4] in C. We use the very general compressed sparse column

[3]SGD version 1.1 (svmsgd2) `http://leon.bottou.org/projects/sgd`, SVM$^{\rm light}$ 6.01 and SVM$^{\rm perf}$ 2.1 `http://svmlight.joachims.org`, pegasos `http://ttic.uchicago.edu/~shai/code/`, BMRM version 0.01 `http://users.rsise.anu.edu.au/~chteo/BMRM.html`.

[4]To not measure implementation specific effects (solver, dot-product computation) etc.

(CSC) representation to store the data. Here each element is represented by an index and a value (each 64bit). To solve the reduced problem (3), we use our implementation of improved SMO (Fan et al., 2005).

4.1. Experimental Setup

The datasets used throughout the experiments are summarized in Table 1. We augmented the Cov1, CCAT, Astro datasets from (Joachims, 2006) by the MNIST, a artificial dense and two larger bioinformatics splice datasets for worm and human. The artificial dataset was generated from two Gaussians with different diagonal covarience matrices of multiple scale. If not otherwise stated experiments were performed on a 2.4GHz AMD Opteron Linux machine. We disabled the bias term in the comparison. As stopping conditions we use the defaults: $\varepsilon_{light} = \varepsilon_{gpdt} = 0.001$, $\varepsilon_{perf} = 0.1$ and $\varepsilon_{bmrm} = 0.001$. For OCAS we used the same stopping condition which is implemented in SVM$^{\text{perf}}$, i.e., $\frac{F(\boldsymbol{w})-F_t(\boldsymbol{w})}{C} \leq \frac{\varepsilon_{perf}}{100} = 10^{-3}$. Note that these ε have a very different meaning denoting the maximum KKT violation for SVM$^{\text{light}}$, the maximum tolerated violation of constraints for SVM$^{\text{perf}}$ and for the BMRM the relative duality gap. For SGD we fix the number of iterations to 10 and for Pegasos we use $100/\lambda$, as suggested in (Shwartz et al., 2007). For the regularization parameter C and λ we use the following relations: $\lambda = 1/C$, $C_{perf} = C/100$, $C_{bmrm} = C$ and $C_{light} = Cm$. Throughout experiments we use C as a shortcut for C_{light}.[5]

Dataset	Examples	Dim	Sp	Split
MNIST	70,000	784	19	77/09/14
Astro	99,757	62,369	0.08	43/05/52
Artificial	150,000	500	100	33/33/33
Cov1	581,012	54	22	81/09/10
CCAT	804,414	47,236	0.16	87/10/03
Worm	1,026,036	804	25	80/05/15
Human	15,028,326	564	25	

Table 1. Datasets used in the experimental evaluation. Sp denotes the average number of non-zero elements of a dataset in percent. Split describes the size of the train/validation/test sets in percent. Datasets are available from the following urls: MNIST `http://yann.lecun.com/exdb/mnist/`, Cov1 `http://kdd.ics.uci.edu/databases/covertype/covertype.html`, CCAT `http://www.daviddlewis.com/resources/testcollections/rcv1/`, Worm and Human `http://www.fml.tuebingen.mpg.de/raetsch/projects/lsmkl`

[5]The exact cmdlines are: `svm_perf_learn -l 2 -m 0 -t 0 --b 0 -e 0.1 -c` C_{perf}, `pegasos -lambda` λ `-iter` $100/\lambda$ `-k 1`, `svm_learn -m 0 -t 0 -b 0 -e 1e-3 -c` C_{light}, `bmrm-train -r 1 -m 10000 -i 999999 -e 1e-3 -c` C_{bmrm}, `svmsgd2 -lambda` λ `-epochs 10`

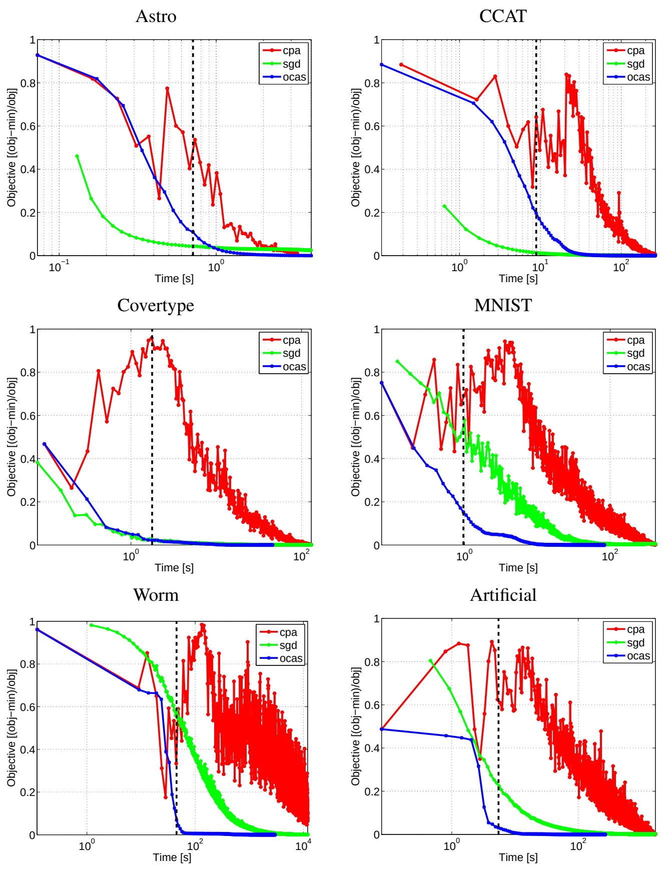

Figure 2. Objective value vs. Training time of CPA (red), SGD (green) and OCAS (blue) measured for a different number of training examples.The dashed line shows the time required to run SGD for 10 iterations. OCAS was stopped when the precision fell below 10^{-6} or the training time for CPA was achieved. In all cases OCAS achieves the minimal objective value and is even on half of the datasets already in the beginning outperforming all other methods including SGD.

4.2. Evaluation

In the following paragraphs we run and evaluate the aforementioned experiments 1–5.

Training time and objective for optimal C We trained all methods on all except the human splice dataset using the training data and measured training time (in seconds) and computed the unconstrained objective value $F(\boldsymbol{w})$

The obtained results are displayed in Table 2. The proposed method – OCAS – consistently outperforms all its competitors of the *accurate solver* category on all benchmark datasets in terms of training time while obtaining a comparable (often the best) objective value. BMRM and SVM$^{\text{perf}}$ implement the same CPA algorithm but due to implementation specific details results can be different. Our implementation of CPA gives very similar results (not shown).[6] Note that for SGD, Pegasos (and SVM$^{perf2.0}$ –

[6]In contrast to SVM$^{\text{perf}}$, BMRM and our implementation of

	astro		ccat		cov1		mnist		worm		artificial	
svmlight	**2.0939e+03**		**8.1235e+04**		**2.5044e+06**		**6.7118e+05**					
	2972	22	77429	5295	1027310	41531	622391	2719		-		
svmperf	**2.1180e+03**		**8.1744e+04**		**2.5063e+06**		**6.7245e+05**		**3.2224e+04**		**1.3186e+02**	
	38	2	228	228	520	152	1295	228	2029	4436	709	162
bmrm	**2.1152e+03**		**8.1682e+04**		**2.5060e+06**		**6.7250e+05**					
	42	2	327	248	678	225	2318	4327		-		
ocas	**2.1103e+03**		**8.1462e+04**		**2.5045e+06**		**6.7158e+05**		**3.1920e+04**		**1.3172e+02**	
	21	1	48	25	80	10	137	10	125	258	76	13
pegasos	**2.1090e+03**		**8.1564e+04**		**2.5060e+06**		**Error**		**4.6212e+04**		**1.3120e+03**	
	2689K	4	70M	127	470M	460	270M	647	82M	213	25K	1
sgd	**2.2377e+03**		**8.2963e+04**		**2.6490e+06**		**1.3254e+06**		**2.1299e+05**		**1.8097e+02**	
	10	1	10	4	10	1	10	1	10	9	10	2

Table 2. Comparison of OCAS against other SVM solvers. ”-” means not converged, blank not attempted. Shown in bold is the unconstrained SVM objective value Eq. (1). The two numbers below the objective value denote the number of iterations (left) and the training time in seconds (right). Lower timing and objective values mean “better.” All methods solve the unbiased problem. As convergence criteria the standard settings described in Section 4.1 are used. On MNIST pegasos ran into numerical problems. OCAS clearly outperforms all of its competitors in the *accurate solver* category by a large margin achieving similar and often lowest objective value. The objective value obtained by SGD and Pegasos is often far away from the optimal solution, cf. text for a further discussion.

not shown) the objective value sometimes deviates significantly from the true objective. As a result the learned classifier may differ substantially from the optimal parameter $\boldsymbol{w}^*$. However as training times for SGD are significantly below all others it remains unclear whether SGD achieves the same precision using less time when run for further iterations. An answer to this question is given in the next paragraph.

Speed of convergence (time vs. objective) To address this problem we re-ran the best methods CPA, OCAS and SGD, recording *intermediate* progress, i.e. while optimization record time and objective for several time points. The results are shown in Figure 2. Ocas was stopped when reaching the maximum time or a precision of $1 - F(\boldsymbol{w}^*)/F(\boldsymbol{w}) \leq 10^{-6}$ and was in all cases achieving the minimum objective. In three of the six datasets OCAS not only as expected at a later time point achieves the best objective but already from the very beginning. Further analysis made clear that OCAS wins over SGD in cases where *large C* were used and thus the optimization problem is more difficult. Still plain SGD outcompetes even CPA. One may argue that practically the true objective is not the unconstrained SVM-primal value (1), but the performance on a validation set, i.e. optimization is stopped when the validation error won't change.

One should however note that one in this case does not obtain an SVM but *some classifier* instead. Then a comparison should not be limited to SVM solvers but should be open to any other large scale approach, like on-line algorithms (e.g. perceptrons) too. We argue that to compare *SVM solvers* in a fair way one needs to compare objective values. As it is still interesting to see how the methods perform w.r.t. classification performance we analyze them under this criterion in the next paragraph.

Time to perform a full model selection When using SVMs in practice, their C parameter needs to be tuned in model selection. We therefore train all methods using different settings[7] for C on the training part of all datasets, evaluate them on the validation set and choose the best model to do predictions on the test set. As performance measure we use the area under the receiver operator characteristic curve (auROC) (Fawcett, 2003). Again among the *accurate methods* OCAS outperforms its competitors by a large margin, followed by SVM$^{\mathrm{perf}}$. Note that for all *accurate methods* the performance is very similar and has little variance. Except for the artificial dataset plain SGD is clearly fastest while achieving a similar accuracy. However the optimal parameter settings for accurate SVMs and SGD are different. Accurate SVM solvers use a larger C constant than SGD. For lower C the objective function is dominated by the regularization term $\|w\|$. A potential explanation is that SGDs update rule puts more emphasize on the regularization term and SGD when not run for a large number of iterations does imply early stopping.

Scalability w.r.t. Dataset Size In this section, we investigate how computational time of OCAS, CPA and SGD scales with the number of examples on the worm splice dataset, for sizes 100 to $1,026,036$. We again use our implementation of CPA that shares essential sub-routines with OCAS. Results are shown and discussed in Figure 3.

CPA did not converge for large C on worm even after 5000 iterations. Most likely the core solver of SVM$^{\mathrm{perf}}$ is more robust.

[7]For Worm and Artificial we used $C = 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5,$ for CCAT, Astro, Cov1 $C = 0.1, 0.5, 1, 5, 10$ and for MNIST $C = 1, 5, 10, 50, 100$.

	astro		ccat		cov1		mnist		worm		artificial	
avg svm perf	**98.15 ± 0.00**		**98.51 ± 0.01**		**83.92 ± 0.01**		**95.86 ± 0.01**		**99.45 ± 0.00**		**86.38 ± 0.02**	
svmlight	1	152	1	124700	10	282703	10	9247	-			
svmperf	1	13	1	1750	5	781	10	887	1	22983	0.005	24520
bmrm	1	17	1	2735	10	1562	10	20278	-			
ocas	1	4	1	163	50	51	10	35	0.1	1438	0.005	6740
pegasos	**98.15**		**98.51**		**83.89**		**95.84**		**99.27**		**78.35**	
	1	59	1	2031	5	731	5	2125	5	1438	5	201
sgd	**98.13**		**98.52**		**83.88**		**95.71**		**99.43**		**80.88**	
	0.5	1	1	20	1	5	1	3	0.005	69	0.005	7

Table 3. Comparison of OCAS against other SVM solvers. ”-” means not converged, blank not attempted. Shown in bold is the area under the receiver operator characteristic curve (auROC) obtained for the best model obtained via model selection over a wide range of regularization constants C. In each cell, numbers on the left denote the optimal C, numbers on the right the training time in seconds to perform the whole model selection. As there is little variance, for accurate SVM solvers only the mean and standard deviation of the auROC are shown. SGD is clearly fastest achieving similar performance for all except for the artificial dataset. However often a smaller C than the ones chosen by accurate SVMs is selected — an indication that the learned decision function is only remotely SVM-like. Among the accurate solvers OCAS clearly outperforms its competitors. It should be noted that training times for all accurate methods are dominated by training for large C (see Table 2 for training times for the optimal C). For further discussion see the text.

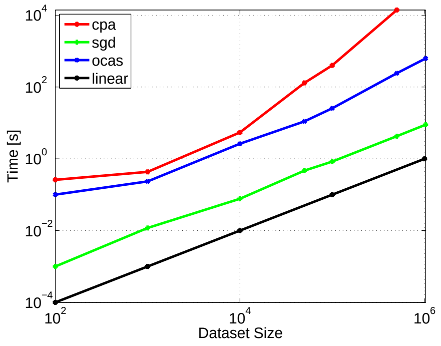

Figure 3. This figure displays how the methods scale with dataset size on the worm splice dataset. The slope of the “lines” in this figure denotes the exponent e in $\mathcal{O}(m^e)$, where the black line denotes linear effort $\mathcal{O}(m)$. Both OCAS and SGD scale about linearly. Note that SGD is much faster (as it runs for a fixed number of iterations and thus does early stopping).

CPUs	1	2	4	8	16
speedup	1	1.77	3.09	4.5	4.6
line search (s)	238	184	178	139	117
$\boldsymbol{a}_t$ (s)	270	155	80	49	45
output (s)	2476	1300	640	397	410
total (s)	3087	1742	998	684	671

Table 4. Speedups due to parallelizing OCAS achieved on 15 million human splice dataset.

Effects of Parallelization As OCAS training times are very low on the above datasets, we also apply OCAS to the 15 million human splice dataset. Using a 2.4GHz 16-Core AMD Opteron Linux machine we run OCAS using $C = 0.0001$ on 1 to 16 CPUs and show the accumulated times for each of the subtasks, the total training time and the achieved speedup w.r.t. the single CPU algorithm in Table 4. Also shown is the time accumulated for each of the threads. As can be seen — except for the line search — computations distribute nicely. Using 8 CPU cores the speedup saturates at a factor of 4.5, most likely as memory access becomes the bottleneck (for 8 CPUs output computation creates a load of 28GB/s just on memory reads).

5. Conclusions

We have developed a new Linear SVM solver called OCAS, which outperforms current state of the art SVM solvers by several orders of magnitude. OCAS even in the early optimization steps shows often faster convergence than the so far in this domain dominating approximative methods. By parallelizing the subtasks of the algorithm, OCAS gained additional speedups of factors up to 4.6 on a multi-core multiprocessor machine. Using OCAS we were able to train on a dataset of size 15 million examples (itself about 32GB in size) in just 671 seconds. As extensions to one and multi-class are straight forward, we plan to implement them in the near future. Furthermore OCAS can be extended to work with a bias term. Finally it will be future work to investigate how the kernel framework can be incorporated into OCAS and how the $\mathcal{O}(\frac{1}{\varepsilon})$ result of (Teo et al., 2007) can be applied to OCAS. An implementation of OCAS is available within the shogun toolbox `http://www.shogun-toolbox.org` and as a separate library from `http://ida.first.fraunhofer.de/~franc/ocas`.

Acknowledgements

The authors gratefully acknowledge partial support from the PASCAL Network of Excellence (EU 506778). VF was supported by Marie Curie Intra-European Fellowship grant SCOLES (MEIF-CT-2006-042107). We thank A. Zien, G. Rätsch and G. Blanchard for great discussions.

References

Bottou, L., & Bousquet, O. (2008). The tradeoffs of large scale learning. In *NIPS 20*. MIT Press.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for svms*. http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chapelle, O. (2007). Training a Support Vector Machine in the Primal. *Neural Comp.*, *19*, 1155–1178.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Cristianini, N., & Shwawe-Taylor, J. (2000). *An introduction to support vector machines*. Cambridge, UK: CUP.

Fan, R.-E., Chen, P.-H., & Lin, C.-J. (2005). Working set selection using second order information for training svm. *Journal of Machine Learning Research*, *6*, 1889–1918.

Fawcett, T. (2003). *Roc graphs: Notes and practical considerations for data mining researchers*. Technical Report HPL-2003-4). HP Laboratories, Palo Alto, CA, USA.

Franc, V., & Sonnenburg, S. (2007). *Optimized cutting plane algorithm for SVMs*. Research Report; Electronic Publication 1). Fraunhofer Institute FIRST.

Joachims, T. (1999). Making large–scale SVM learning practical. *Advances in Kernel Methods — Support Vector Learning* (pp. 169–184). Cambridge, MA, USA: MIT Press.

Joachims, T. (2006). Training linear svms in linear time. *KDD'06*.

Lin, C.-J., Weng, R. C., & Keerthi, S. S. (2007). Trust region newton methods for large-scale logistic regression. *ICML '07* (pp. 561 – 568). ACM New York.

Shwartz, S.-S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. *ICML '07* (pp. 807–814). ACM Press.

Sindhwani, V., & Keerthi, S.-S. (2007). Newton methods for fast solution of semi-supervised linear svms. In *Large scale kernel machines*. MIT Press.

Teo, C. H., Le, Q., Smola, A., & Vishwanathan, S. (2007). A scalable modular convex solver for regularized risk minimization. *KDD'07*.

Zanni, L., Serafini, T., & Zanghirati, G. (2006). Parallel software for training. *JMLR*, *7*, 1467–1492.

A. Computing Line-search efficiently

The line-search (7) is an essential procedure of OCAS which is called at every iteration. We show that the line-search can be solved exactly in $\mathcal{O}(m \log m)$ time. First, we introduce a more compact notation for the objective function of the line-search problem (7) $F(\boldsymbol{w}^{\rm b}_{t-1}(1-k) + \boldsymbol{w}_t k)$ by $G(k) = g_0(k) + \sum_{i=1}^m g_i(k)$ where $g_0(k) = \frac{1}{2}k^2 A_0 + kB_0 + C_0$, $g_i(k) = \max\{0, kB_i + C_i\}$, $A_0 = \|\boldsymbol{w}^{\rm b}_{t-1} - \boldsymbol{w}_t\|^2$, $B_0 = \langle \boldsymbol{w}^{\rm b}_{t-1}, \boldsymbol{w}_t - \boldsymbol{w}^{\rm b}_{t-1}\rangle$, $C_0 = \dfrac{1}{2}\|\boldsymbol{w}^{\rm b}_{t-1}\|^2$, $B_i = \dfrac{C}{m} y_i \langle \boldsymbol{x}_i, \boldsymbol{w}^{\rm b}_{t-1} - \boldsymbol{w}_t\rangle$ and $C_i = \dfrac{C}{m}(1 - y_i\langle \boldsymbol{x}_i, \boldsymbol{w}^{\rm b}_{t-1}\rangle)$. Hence the line-search (7) involves solving $k^* = \operatorname{argmin}_{k\geq 0} G(k)$ and computing $\boldsymbol{w}^{\rm b}_t = \boldsymbol{w}^{\rm b}_{t-1}(1-k^*) + \boldsymbol{w}_t k^*$. As function G is convex the unconstrained minimum of G is attained at the point k^* at which the subdifferential $\partial G(k)$ contains zero, i.e. $0 \in \partial G(k^*)$. The subdifferential of G is $\partial G(k) = kA_0 + B_0 + \sum_{i=1}^m \partial g_i(k)$,

$$\partial g_i(k) = \begin{cases} 0 & \text{if} \quad kB_i + C_i < 0\,, \\ B_i & \text{if} \quad kB_i + C_i > 0\,, \\ [0, B_i] & \text{if} \quad kB_i + C_i = 0\,. \end{cases}$$

Note that the subdifferential is not a function as there exist k for which $\partial G(k)$ is an interval.

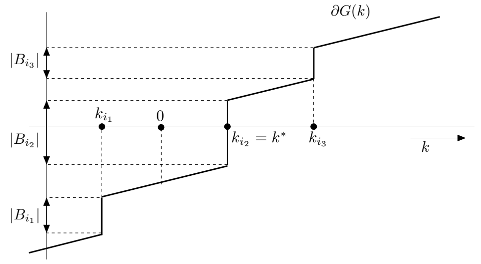

Figure 4. Illustration of the subgradient $\partial G(k)$ of the objective function $G(k)$ minimized in the line-search.

The first term of the subdifferential $\partial G(k)$ is an ascending linear function $kA_0 + B_0$ since A_0 must be greater than zero (A_0 is zero only if the algorithm has converged but then the line-search is not invoked). The term $\partial g_i(k)$ is either constantly zero, if $B_i = 0$, or it is a step-like jump whose value changes at the point $k_i = -\frac{C_i}{B_i}$. The value of $\partial g_i(k)$ w.r.t. k is summarized in Table 5. Hence the subdifferential $\partial G(k)$ is a monotonically increasing function as is illustrated in Figure 4.

	$k < k_i$	$k = k_i$	$k > k_i$
$B_i = 0$	0	0	0
$B_i < 0$	B_i	$[B_i, 0]$	0
$B_i > 0$	0	$[0, B_i]$	B_i

Table 5. The value of $\partial g_i(k)$ with respect to k.

To solve $k^* = \operatorname{argmin}_{k\geq 0} G(k)$ we proceed as follows: If $\max(\partial G(0))$ is strictly greater than zero then the unconstrained minimum $\operatorname{argmin}_k G(k)$ is at a point less or equal to 0. Thus the constrained minimum is attained at the point $k^* = 0$.

If $\max(\partial G(0))$ is less then zero then the optimum k^* corresponds to the unconstrained optimum $\operatorname{argmin}_k G(k)$ attained at the intersection between the graph of $\partial G(k)$ and the x-axis. This point can be found efficiently by sorting $K = \{k_i \mid k_i > 0, i = 1, \ldots, m\}$ and checking the condition $0 \in G(k)$ for $k \in K$ and for k in the intervals which split the domain $(0, \infty)$ in the points K. These computation are dominated by sorting the numbers K which takes $\mathcal{O}(|K| \log |K|)$.