Journal of Machine Learning Research 10 (2009) 2157-2192 Submitted 2/09; Published 10/09

Optimized Cutting Plane Algorithm for Large-Scale Risk Minimization*

Vojtěch Franc XFRANCV@CMP.FELK.CVUT.CZ
Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Technicka 2, 166 27 Praha 6,
Czech Republic

Sören Sonnenburg SOEREN.SONNENBURG@TUEBINGEN.MPG.DE
Friedrich Miescher Laboratory
Max Planck Society
Spemannstr. 39
72076 Tübingen, Germany

Editor: Michele Sebag

Abstract

We have developed an optimized cutting plane algorithm (OCA) for solving large-scale risk minimization problems. We prove that the number of iterations OCA requires to converge to a ε precise solution is approximately linear in the sample size. We also derive OCAS, an OCA-based linear binary Support Vector Machine (SVM) solver, and OCAM, a linear multi-class SVM solver. In an extensive empirical evaluation we show that OCAS outperforms current state-of-the-art SVM solvers like SVM$^{\text{light}}$, SVM$^{\text{perf}}$ and BMRM, achieving speedup factor more than 1,200 over SVM$^{\text{light}}$ on some data sets and speedup factor of 29 over SVM$^{\text{perf}}$, while obtaining the same precise support vector solution. OCAS, even in the early optimization steps, often shows faster convergence than the currently prevailing approximative methods in this domain, SGD and Pegasos. In addition, our proposed linear multi-class SVM solver, OCAM, achieves speedups of factor of up to 10 compared to SVM$^{\text{multi-class}}$. Finally, we use OCAS and OCAM in two real-world applications, the problem of human acceptor splice site detection and malware detection. Effectively parallelizing OCAS, we achieve state-of-the-art results on an acceptor splice site recognition problem only by being able to learn from all the available 50 million examples in a 12-million-dimensional feature space. Source code, data sets and scripts to reproduce the experiments are available at `http://cmp.felk.cvut.cz/~xfrancv/ocas/html/`.

Keywords: risk minimization, linear support vector machine, multi-class classification, binary classification, large-scale learning, parallelization

1. Introduction

Many applications in, for example, bioinformatics, IT-security and text classification come with *huge* numbers (e.g., millions) of data points, which are indeed *necessary* to obtain state-of-the-

*. A large part of the work was done while VF and SS were at the Fraunhofer Institute FIRST, Kekulestr. 7, 12489 Berlin, Germany.

art results. They, therefore, require extremely efficient computational methods capable of dealing with ever growing data sizes. A wide range of machine learning methods can be described as the unconstrained regularized risk minimization problem

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}\in\Re^n} F(\mathbf{w}) := \frac{1}{2}\|\mathbf{w}\|^2 + CR(\mathbf{w})\,, \tag{1}$$

where $\mathbf{w} \in \Re^n$ denotes the parameter vector to be learned, $\frac{1}{2}\|\mathbf{w}\|^2$ is a quadratic regularization term, $C > 0$ is a fixed regularization constant and $R\colon \Re^n \to \Re$ is a non-negative convex risk function approximating the empirical risk (e.g., training error).

Special cases of problem (1) are, for example, support vector classification and regression (e.g., Cortes and Vapnik, 1995; Cristianini and Shawe-Taylor, 2000), logistic regression (Collins et al., 2000), maximal margin structured output classification (Tsochantaridis et al., 2005), SVM for multivariate performance measures (Joachims, 2005), novelty detection (Schölkopf et al., 1999), learning Gaussian processes (Williams, 1998) and many others.

Problem (1) has usually been solved in its dual formulation, which typically only uses the computation of dot products between examples. This enables the use of kernels that implicitly compute the dot product between examples in a Reproducing Kernel Hilbert Space (RKHS) (Schölkopf and Smola, 2002). On the one hand, solving the dual formulation is efficient when dealing with high-dimensional data. On the other hand, efficient and accurate solvers for optimizing the kernelized dual formulation for large sample sizes do not exist.

Recently, focus has shifted towards methods optimizing problem (1) directly in the primal. Using the primal formulation is efficient when the number of examples is very large and the dimensionality of the input data is moderate or the inputs are sparse. This is typical in applications like text document analysis and bioinformatics, where the inputs are strings embedded into a sparse high-dimensional feature space, for example, by using the bag-of-words representation. A way to exploit the primal formulation for learning in the RKHS is based on decomposing the kernel matrix and thus effectively linearizing the problem (Schölkopf and Smola, 2002).

Due to its importance, the literature contains a plethora of specialized solvers dedicated to particular risk functions $R(\mathbf{w})$ of problem (1). Binary SVM solvers employing the hinge risk $R(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i\langle \mathbf{w}, \mathbf{x}_i\rangle\}$ especially have been studied extensively (e.g., Joachims, 1999; Zanni et al., 2006; Chang and Lin, 2001; Sindhwani and Keerthi, 2007; Chapelle, 2007; Lin et al., 2007). Recently, Teo et al. (2007) proposed the Bundle Method for Risk Minimization (BMRM), which is a general approach for solving problem (1). BMRM is not only a general but also a highly modular solver that only requires two specialized procedures, one to evaluate the risk $R(\mathbf{w})$ and one to compute its subgradient. BMRM is based on iterative approximation of the risk term by cutting planes. It solves a reduced problem obtained by substituting the cutting plane approximation of the risk into the original problem (1). Teo et al. (2007) compared BMRM with specialized solvers for SVM classification, SV regression and ranking, and reported promising results. However, we will show that the implementation of the cutting plane algorithm (CPA) used in BMRM can be drastically sped up making it efficient even for large-scale SVM binary and multi-class classification problems.

In this paper, we develop an *efficient and general* algorithm to solve the regularized risk minimization problem (1). Building on the work of Teo et al. (2007), we propose an *optimized cutting plane algorithm* (OCA) that extends the standard CPA in two ways. First, unlike CPA, we use the solution of the reduced problem as a direction in the line-search to directly minimize the original

(master) problem (1). Second, we introduce a new cutting plane selection strategy that reduces the number of cutting planes required to achieve the prescribed precision and thus significantly speeds up convergence. An *efficient* line-search procedure for the optimization of (1) is the *only additional requirement* of OCA compared to the standard CPA (or BMRM).

While our proposed method (OCA) is applicable to a wide range of risk terms R, we will—due to their importance—discuss in more detail two special cases: learning of the binary (two-class) and multi-class SVM classifiers. We show that the line-search procedure required by OCA can be solved exactly for both the binary and multi-class SVM problems in $O(m \log m)$ and $O(m \cdot Y^2 + m \cdot Y \log(m \cdot Y))$ time, respectively, where m is the number of examples and Y is the number of classes. We abbreviate OCA for binary SVM classifiers with OCAS and the multi-class version with OCAM.

We perform an extensive experimental evaluation of the proposed methods, OCAS and OCAM, on several problems comparing them with the current state of the art. In particular, we would like to highlight the following experiments and results:

- We compare OCAS with the state-of-the-art solvers for binary SVM classification on previously published data sets. We show that OCAS significantly outperforms the competing approaches achieving speedups factors of more than 1,200.
- We evaluate OCAS using the large-scale learning challenge data sets and evaluation protocols described in Sonnenburg et al. (2009). Although OCAS is an implementation of a general method for risk minimization (1), it is shown to be competitive with dedicated binary SVM solvers, which ultimately won the large-scale learning challenge.
- We demonstrate that OCAS can be sped up by efficiently parallelizing its core subproblems.
- We compare OCAM with the state-of-the-art implementation of the CPA-based solver for training multi-class SVM classifiers. We show that OCAM achieves speedups of an order of magnitude.
- We show that OCAS and OCAM achieve state-of-the-art recognition performance for two real-world applications. In the first application, we attack a splice site detection problem from bioinformatics. In the second, we address the problem of learning a malware behavioral classifier used in computer security systems.

The OCAS solver for training the binary SVM classification was published in our previous paper (Franc and Sonnenburg, 2008a). This paper extends the previous work in several ways. First, we formulate OCA for optimization of the general risk minimization problem (1). Second, we give an improved convergence proof for the general OCA (in Franc and Sonnenburg 2008a the upper bound on the number of iterations as a function of precision ε scales with $O(\frac{1}{\varepsilon^2})$, while in this paper the bound is improved to $O(\frac{1}{\varepsilon})$). Third, we derive a new instance of OCA for training the multi-class SVM classifiers. Fourth, the experiments are extended by (i) including the comparison on the challenge data sets and using the challenge protocol, (ii) performing experiments on multi-class classification problems and (iii) solving two real-world applications from bioinformatics and computer security.

The remainder of this paper is organized as follows. The standard cutting plane algorithm (CPA) to solve (1) is reviewed in Section 2. In Section 3, we point out a source of inefficiency

of CPA and propose a new optimized cutting plane algorithm (OCA) to drastically reduce training times. We then develop OCA for two special cases linear binary SVMs (OCAS, see Section 3.1) and linear multiclass SVMs (OCAM, see Section 3.2). In Section 4, we empirically show that using OCA, training times can be drastically reduced on a wide range of large-scale data sets including the challenge data sets. Finally, we attack two real-world applications. First, in Section 5.1, we apply OCAS to a human acceptor splice site recognition problem achieving state-of-the art results by training on all available sequences—a data set of 50 million examples (itself about 7GB in size) using a 12 million dimensional feature space. Second, in Section 5.2, we apply OCAM to learn a behavioral malware classifier and achieve a speedup of factor of 20 compared to the previous approach and a speedup of factor of 10 compared to the state-of-the-art implementation of the CPA. Section 6 concludes the paper.

2. Cutting Plane Algorithm

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

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

(2) is obtained from master problem (1) by substituting a piece-wise linear approximation R_t for the risk R. Note that only the risk term R is approximated while the regularization term $\frac{1}{2}\|\mathbf{w}\|^2$ remains unchanged. The idea is that in practise one usually needs only a small number of linear terms in the piece-wise linear approximation R_t to adequately approximate the risk R around the optimum $\mathbf{w}^*$. Moreover, it was shown in Teo et al. (2007) that the number of linear terms needed to achieve arbitrary precise approximation does not depend on the number of examples.

We now derive the approximation to R. Since the risk term R is a convex function, it can be approximated at any point $\mathbf{w}'$ by a linear under estimator

$$R(\mathbf{w}) \geq R(\mathbf{w}') + \langle \mathbf{a}', \mathbf{w} - \mathbf{w}' \rangle, \qquad \forall \mathbf{w} \in \Re^n, \tag{3}$$

where $\mathbf{a}' \in \partial R(\mathbf{w}')$ is any subgradient of R at the point $\mathbf{w}'$. We will use a shortcut $b' = R(\mathbf{w}') - \langle \mathbf{a}', \mathbf{w}' \rangle$ to abbreviate (3) as $R(\mathbf{w}) \geq \langle \mathbf{a}', \mathbf{w} \rangle + b'$. In CPA terminology, $\langle \mathbf{a}', \mathbf{w} \rangle + b' = 0$ is called a cutting plane.

To approximate the risk R better than by using a single cutting plane, we can compute a collection of cutting planes $\{\langle \mathbf{a}_i, \mathbf{w} \rangle + b_i = 0 \mid i = 1, \ldots, t\}$ at t distinct points $\{\mathbf{w}_1, \ldots, \mathbf{w}_t\}$ and take their point-wise maximum

$$R_t(\mathbf{w}) = \max\left\{0, \max_{i=1,\ldots,t} \left(\langle \mathbf{a}_i, \mathbf{w} \rangle + b_i\right)\right\}. \tag{4}$$

The zero cutting plane is added to the maximization as the risk R is assumed to be non-negative. The subscript $_t$ denotes the number of cutting planes used in the approximation R_t. It follows directly from (3) that the approximation R_t is exact at the points $\{\mathbf{w}_1, \ldots, \mathbf{w}_t\}$ and that R_t lower bounds R, that is, that $R(\mathbf{w}) \geq R_t(\mathbf{w}), \forall \mathbf{w} \in \Re^n$ holds. In turn, the objective function F_t of the reduced problem lower bounds the objective F of the master problem.

Having readily computed R_t, we may now use it in the reduced problem (2). Substituting (4) with (2), the reduced problem can be expressed as a linearly constrained quadratic problem

$$(\mathbf{w}_t, \xi_t) := \operatorname*{argmin}_{\mathbf{w} \in \Re^n, \xi \in \Re} \left[\frac{1}{2}\|\mathbf{w}\|^2 + C\xi\right], \tag{5}$$

subject to

$$\xi \geq 0\,, \quad \xi \geq \langle \mathbf{a}_i, \mathbf{w} \rangle + b_i\,, \quad i = 1, \ldots, t\,.$$

The number of constraints in (5) equals the number of cutting planes t which can be drastically lower than the number of constraints in the master problem (1) when expressed as a constrained QP task. As the number of cutting planes is typically much smaller than the data dimensionality n, it is convenient to solve the reduced problem (5) by optimizing its dual formulation, which reads

$$\alpha_t := \operatorname*{argmax}_{\alpha \in \mathcal{A}_t} D_t(\alpha) := \Big[\sum_{i=1}^{t} \alpha_i b_i - \frac{1}{2} \Big\| \sum_{i=1}^{t} \mathbf{a}_i \alpha_i \Big\|^2 \Big]\,, \tag{6}$$

where $\mathcal{A}_t$ is a convex feasible set containing all vectors $\alpha \in \Re^t$ satisfying

$$\sum_{i=1}^{t} \alpha_i \leq C\,, \qquad \alpha_i \geq 0\,, i = 1, \ldots, t\,.$$

The dual formulation contains only t variables bound by $t+1$ constraints of simple form. Thus task (6) can be efficiently optimized by standard QP solvers. Having (6) solved, the primal solution can be computed as

$$\mathbf{w}_t = -\sum_{i=1}^{t} \mathbf{a}_i [\alpha_t]_i\,, \quad \text{and} \quad \xi_t = \max_{i=1,\ldots,t} \left(\langle \mathbf{w}_t, \mathbf{a}_i \rangle + b_i \right).$$

Solving the reduced problem is beneficial if we can effectively select a small number of cutting planes such that the solution of the reduced problem is sufficiently close to the master problem. CPA selects the cutting planes using a simple strategy described by Algorithm 1.

Algorithm 1 Cutting Plane Algorithm (CPA)

1: $t := 0$.
2: **repeat**
3: Compute $\mathbf{w}_t$ by solving the reduced problem (5).
4: Add a new cutting plane to approximate the risk R at the current solution $\mathbf{w}_t$, that is, compute $\mathbf{a}_{t+1} \in \partial R(\mathbf{w}_t)$ and $b_{t+1} := R(\mathbf{w}_t) - \langle \mathbf{a}_{t+1}, \mathbf{w}_t \rangle$.
5: $t := t + 1$
6: **until** a stopping condition is satisfied.

The algorithm is very general. To use it for a particular problem one only needs to supply a formula to compute the cutting plane as required in Step 4, that is, formulas for computing the subgradient $\mathbf{a} \in \partial R(\mathbf{w})$ and the objective value $R(\mathbf{w})$ at given point $\mathbf{w}$.

It is natural to stop the algorithm when

$$F(\mathbf{w}_t) - F_t(\mathbf{w}_t) \leq \varepsilon \tag{7}$$

holds. Because $F_t(\mathbf{w}_t)$ is the lower bound of the optimal value, $F(\mathbf{w}^*)$, it follows that a solution $\mathbf{w}_t$ satisfying (7) also guarantees $F(\mathbf{w}_t) - F(\mathbf{w}^*) \leq \varepsilon$, that is, the objective value differs from the optimal one by ε at most. An alternative stopping condition advocated in Joachims (2006) stops the algorithm when $R(\mathbf{w}_t) - R_t(\mathbf{w}_t) \leq \hat{\varepsilon}$. It can be seen that the two stopping conditions become equivalent if we set $\varepsilon = C\hat{\varepsilon}$. Hence we will consider only the former stopping condition (7).

Theorem 1 by Teo et al. (2007) guarantees convergence of the CPA algorithm in $O(\frac{1}{\varepsilon})$ time for a broad class of risk functions:

Theorem 1 *(Teo et al., 2007) Assume that $\|\partial R(\mathbf{w})\| \leq G$ for all $\mathbf{w} \in \mathcal{W}$, where $\mathcal{W}$ is some domain of interest containing all $\mathbf{w}_{t'}$ for $t' \leq t$. In this case, for any $\varepsilon > 0$ and $C > 0$, Algorithm 1 satisfies the stopping condition (7) after at most*

$$\log_2 \frac{F(\mathbf{0})}{4C^2G^2} + \frac{8C^2G^2}{\varepsilon} - 2$$

iterations.

3. Optimized Cutting Plane Algorithm (OCA)

We first point out a source of inefficiency 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 $F_t(\mathbf{w}_t)$ monotonically increases w.r.t. the number of iterations t. However, there is no such guarantee for the master problem objective $F(\mathbf{w}_t)$. Even though it will ultimately converge arbitrarily close to the minimum $F(\mathbf{w}^*)$, its value can heavily fluctuate between iterations (Figure 1). The reason for these

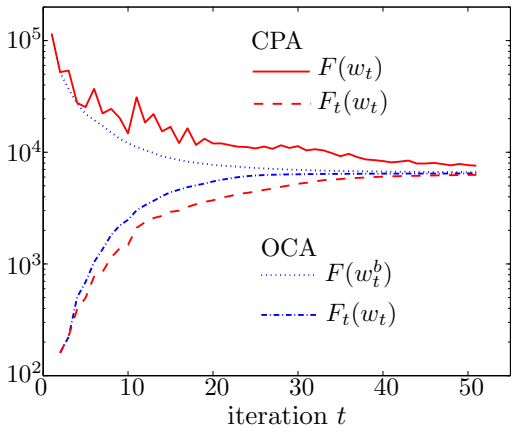

Figure 1: Convergence behavior of the standard CPA vs. the proposed OCA.

fluctuations is that at each iteration t, CPA selects the cutting plane that perfectly approximates the master objective F *at the current solution* $\mathbf{w}_t$. However, there is no guarantee that such a cutting plane will be an active constraint in the vicinity of the optimum $\mathbf{w}^*$, nor must the new solution $\mathbf{w}_{t+1}$ of the reduced problem improve the master objective. In fact, it often occurs that $F(\mathbf{w}_{t+1}) > F(\mathbf{w}_t)$. As a result, a lot of the selected cutting planes do not contribute to the approximation of the master objective around the optimum which, in turn, increases the number of iterations.

To speed up the convergence of CPA, we propose a new method which we call the *optimized cutting plane algorithm* (OCA). Unlike standard CPA, OCA aims at simultaneously optimizing the master and reduced problems' F and F_t objective functions, respectively. In addition, OCA tries to select cutting planes that have a higher chance of actively contributing to the approximation of the master objective function F around the optimum $\mathbf{w}^*$. In particular, we propose the following three changes to CPA.

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

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

$$\mathbf{w}_t^{\mathrm{b}} = \min_{k\geq 0} F(\mathbf{w}_{t-1}^{\mathrm{b}}(1-k)+\mathbf{w}_t k)\,. \tag{8}$$

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

$$\mathbf{w}_t^c = \mathbf{w}_t^{\mathrm{b}}(1-\mu)+\mathbf{w}_t\mu\,, \tag{9}$$

where $\mu\in(0,1]$ is a prescribed parameter. Having the point $\mathbf{w}_t^c$, the new cutting plane is given by $\mathbf{a}_{t+1}\in\partial R(\mathbf{w}_t^c)$ and $b_{t+1}=R(\mathbf{w}_t^c)-\langle\mathbf{a}_{t+1},\mathbf{w}_t^c\rangle$.

Algorithm 2 describes the proposed OCA. Figure 1 shows the impact of the proposed changes to the convergence. OCA generates a monotonically decreasing sequence of master objective values and a monotonically and strictly increasing sequence of reduced objective values, that is,

$$F(\mathbf{w}_1^{\mathrm{b}})\geq\ldots\geq F(\mathbf{w}_t^{\mathrm{b}})\,,\quad\text{and}\quad F_1(\mathbf{w}_1)<\ldots<F_t(\mathbf{w}_t)\,.$$

Note that for CPA only the latter is satisfied. Similar to CPA, a natural stopping condition for OCA reads

$$F(\mathbf{w}_t^{\mathrm{b}})-F_t(\mathbf{w}_t)\leq\varepsilon\,, \tag{10}$$

where $\varepsilon>0$ is a prescribed precision parameter. Satisfying the condition (10) guarantees that $F(\mathbf{w}_t^{\mathrm{b}})-F(\mathbf{w}^*)\leq\varepsilon$ holds.

Algorithm 2 Optimized Cutting Plane Algorithm (OCA)

1: Set $t:=0$ and $\mathbf{w}_0^{\mathrm{b}}:=\mathbf{0}$.
2: **repeat**
3: Compute $\mathbf{w}_t$ by solving the reduced problem (5).
4: Compute a new best-so-far solution $\mathbf{w}_t^{\mathrm{b}}$ using the line-search (8).
5: Add a new cutting plane: compute $\mathbf{a}_{t+1}\in\partial R(\mathbf{w}_t^c)$ and $b_{t+1}:=R(\mathbf{w}_t^c)-\langle\mathbf{a}_{t+1},\mathbf{w}_t^c\rangle$ where $\mathbf{w}_t^{\mathrm{c}}$ is given by (9).
6: $t:=t+1$
7: **until** a stopping condition is satisfied

Theorem 2 *Assume that $\|\partial R(\mathbf{w})\|\leq G$ for all $\mathbf{w}\in\mathcal{W}$, where $\mathcal{W}$ is some domain of interest containing all $\mathbf{w}_{t'}$ for $t'\leq t$. In this case, for any $\varepsilon>0$, $C>0$ and $\mu\in(0,1]$, Algorithm 2 satisfies the stopping condition (10) after at most*

$$\log_2\frac{F(\mathbf{0})}{4C^2G^2}+\frac{8C^2G^2}{\varepsilon}-2$$

iterations.

Theorem 2 is proven in Appendix A. Finally, there are two relevant remarks regarding Theorem 2:

Remark 1 Although Theorem 2 holds for any μ from the interval $(0,1]$ its particular value has impact on the convergence speed in practice. We found experimentally (see Section 4.1) that $\mu = 0.1$ works consistently well throughout experiments.

Remark 2 Note that the bound on the maximal number of iterations of OCA given in Theorem 2 coincides with the bound for CPA in Theorem 1. Despite the same theoretical bounds, in practice OCA converges significantly faster compared to CPA, achieving speedups of more than one order of magnitude as will be demonstrated in the experiments (Section 4). In the convergence analysis (see Appendix A) we give an intuitive explanation of why OCA converges faster than CPA.

In the following subsections we will use the OCA Algorithm 2 to develop efficient binary linear and multi-class SVM solvers. To this end, we develop fast methods to solve the problem-dependent subtasks, the line-search step (step 4 in Algorithm 2) and the addition of a new cutting plane (step 5 in Algorithm 2).

3.1 Training Linear Binary SVM Classifiers

Given an example set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \in (\Re^n \times \{-1,+1\})^m$, the goal is to find a parameter vector $\mathbf{w} \in \Re^n$ of the liner classification rule $h(\mathbf{x}) = \operatorname{sgn}\langle \mathbf{w}, \mathbf{x} \rangle$. The parameter vector $\mathbf{w}$ is obtained by minimizing

$$F(\mathbf{w}) := \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\}\,, \tag{11}$$

which is a special instance of the regularized risk minimization problem (1) with the risk

$$R(\mathbf{w}) := \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\}\,. \tag{12}$$

It can be seen that (12) is a convex piece-wise linear approximation of the training error $\frac{1}{m}\sum_{i=1}^{m} [\![h(\mathbf{x}_i) \neq y_i]\!]$.

To use the OCA Algorithm 2 for solving (12), we need the problem-dependent steps 4 and 5. First, we need to supply a procedure performing the line-search (8) as required in Step 4. Section 3.1.1 describes an efficient algorithm solving the line-search exactly in $O(m \log m)$ time. Second, Step 5 requires a formula for computing a subgradient $\mathbf{a} \in \partial R(\mathbf{w})$ of the risk (12) which reads

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

Both the line-search and computation of the subgradient can be sped up via the parallelization described in Section 3.1.2. We call the resulting algorithm the *optimized cutting plane algorithm for SVM* (OCAS).

3.1.1 LINE-SEARCH FOR LINEAR BINARY SVM CLASSIFIERS

The line-search (8) requires minimization of a univariate convex function

$$F(\mathbf{w}_{t-1}^{\mathrm{b}}(1-k)+\mathbf{w}_t k) = \frac{1}{2}\|\mathbf{w}_{t-1}^{\mathrm{b}}(1-k)+\mathbf{w}_t k\|^2 + CR(\mathbf{w}_{t-1}^{\mathrm{b}}(1-k)+\mathbf{w}_t k)\,, \tag{13}$$

with R defined by (12). Note that the line-search very much resembles the master problem (1) with one-dimensional data. We show that the line-search can be solved exactly in $O(m\log m)$ time.

We abbreviate $F(\mathbf{w}_{t-1}^{\mathrm{b}}(1-k)+\mathbf{w}_t k)$ by $f(k)$ which is defined as

$$f(k) := f_0(k) + \sum_{i=1}^{m} f_i(k) = \frac{1}{2}k^2 A_0 + kB_0 + C_0 + \sum_{i=1}^{m} \max\{0, kB_i + C_i\}\,,$$

where

$$\begin{array}{rclcrcl} A_0 &=& \|\mathbf{w}_{t-1}^{\mathrm{b}} - \mathbf{w}_t\|^2 & & & & \\ B_0 &=& \langle \mathbf{w}_{t-1}^{\mathrm{b}}, \mathbf{w}_t - \mathbf{w}_{t-1}^{\mathrm{b}}\rangle\,, & B_i &=& \frac{C}{m} y_i \langle \mathbf{x}_i, \mathbf{w}_{t-1}^{\mathrm{b}} - \mathbf{w}_t\rangle\,, & i=1,\ldots,m\,, \\ C_0 &=& \frac{1}{2}\|\mathbf{w}_{t-1}^{\mathrm{b}}\|^2\,, & C_i &=& \frac{C}{m}(1 - y_i\langle \mathbf{x}_i, \mathbf{w}_{t-1}^{\mathrm{b}}\rangle)\,, & i=1,\ldots,m\,. \end{array} \tag{14}$$

Hence the line-search (8) involves solving $k^* = \mathrm{argmin}_{k\geq 0} f(k)$ and computing $\mathbf{w}_t^{\mathrm{b}} = \mathbf{w}_{t-1}^{\mathrm{b}}(1-k^*) + \mathbf{w}_t k^*$. Since $f(k)$ is a convex function, its unconstrained minimum is attained at the point k^*, at which the sub-differential $\partial f(k)$ contains zero, that is, $0 \in \partial f(k^*)$ holds. The subdifferential of f reads

$$\partial f(k) = kA_0 + B_0 + \sum_{i=1}^{m} \partial f_i(k)\,, \quad \text{where} \quad \partial f_i(k) = \begin{cases} 0 & \text{if} \quad kB_i + C_i < 0\,, \\ B_i & \text{if} \quad kB_i + C_i > 0\,, \\ [0, B_i] & \text{if} \quad kB_i + C_i = 0\,. \end{cases}$$

Note that the subdifferential is not a function because there exist k's for which $\partial f(k)$ is an interval. The first term of the subdifferential $\partial f(k)$ is an ascending linear function $kA_0 + B_0$ since A_0 must be greater than zero. Note that $A_0 = \|\mathbf{w}_{t-1}^{\mathrm{b}} - \mathbf{w}_t\|^2$ equals 0 only if the algorithm has converged to the optimum $\mathbf{w}^*$, but in this case the line-search is not invoked. The term $\partial f_i(k)$ appearing in the sum is

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

Table 1: The value of $\partial f_i(k)$ with respect to k.

either constantly zero, if $B_i = 0$, or it is a step-like jump whose value changes at the point $k_i = -\frac{C_i}{B_i}$. In particular, the value of $\partial f_i(k)$ w.r.t. k is summarized in Table 1. Hence the subdifferential $\partial f(k)$ is a monotonically increasing function as is illustrated in Figure 2. To solve $k^* = \mathrm{argmin}_{k\geq 0} f(k)$ we proceed as follows:

1. We compute the maximal value of the subdifferential $\partial f(k)$ at point 0:

$$\max(\partial f(0)) = B_0 + \sum_{i=1}^{m} [\![(B_i < 0 \wedge k_i > 0) \vee (B_i > 0 \wedge k_i \leq 0)]\!] B_i\,.$$

2. If $\max(\partial f(0))$ is strictly greater than zero, we know that the unconstrained minimum $\operatorname{argmin}_k f(k)$ is attained at a point less than or equal to 0. Thus, the constrained minimum $k^* = \operatorname{argmin}_{k\geq 0} f(k)$, that is, the result of the line-search, is attained at the point $k^* = 0$.

3. If $\max(\partial f(0))$ is less than zero, then the optimum $k^* = \operatorname{argmin}_{k\geq 0} f(k)$ corresponds to the unconstrained optimum $\operatorname{argmin}_k f(k)$. To get k^* we need to find an intersection between the graph of $\partial f(k)$ and the x-axis. This can be done efficiently by sorting points $K = \{k_i \mid k_i > 0, i = 1,\ldots,m\}$ and checking the condition $0 \in \partial f(k)$ for $k \in K$ and for k in the intervals which split the domain $(0,\infty)$ in the points K. These computations are dominated by sorting the numbers K, which takes $O(|K|\log|K|)$ time.

Computing the parameters (14) of the function $f(k)$ requires $O(mn)$ time, where m is the number of examples and n is the number of features. Having the parameters computed, the worst-case time complexities of the steps 1, 2 and 3 are $O(m)$, $O(1)$ and $O(m\log m)$, respectively.

3.1.2 PARALLELIZATION

Apart from solving the reduced problem (2), all subtasks of OCAS can be efficiently parallelized:

Output computation. This involves computation of the dot products $\langle \mathbf{w}_t, \mathbf{x}_i \rangle$ for all $i = 1,\ldots,m$, which requires $O(s)$ time, where s equals the number of all non-zero elements in the training examples. Distributing the computation equally to p processors reduces the effort to $O(\frac{s}{p})$. Note that the remaining products with data required by OCAS, that is, $\langle \mathbf{w}_t^{\mathrm{b}}, \mathbf{x}_i \rangle$ and $\langle \mathbf{w}_t^{\mathrm{c}}, \mathbf{x}_i \rangle$, can be computed from $\langle \mathbf{w}_t, \mathbf{x}_i \rangle$ in time $O(m)$.

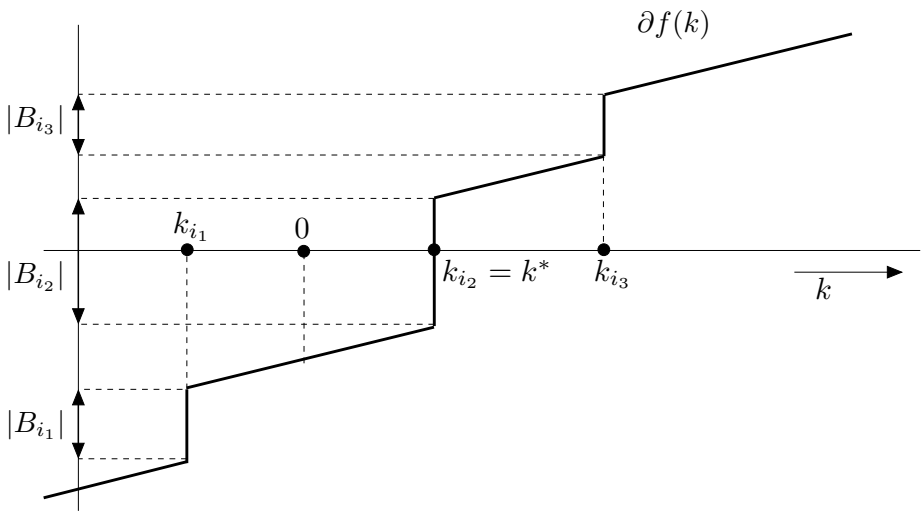

Figure 2: Graph depicting the subdifferential $\partial f(k)$ of the objective function $f(k)$. The line-search requires computing $k^* = \min_{k\geq 0} f(k)$ which is equivalent to finding the intersection k^* between the graph of $\partial f(k)$ and the positive part of the x-axis.

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

Cutting plane computation. The dominant part requires the sum $-\frac{1}{m}\sum_{i=1}^{m}\pi_i y_i \mathbf{x}_i$, which can be done in $O(s_\pi)$ time, where $s_\pi = |\{i|\pi_i \neq 0,\ \forall i = 1,\ldots,m\}|$ is the number of non-zero π_i. Using p processors results in a time complexity of $O(\frac{s_\pi}{p})$.

It is worth mentioning that OCAS usually requires a small number of iterations (usually less than 100 and almost always less than 1000). Hence, solving the reduced problem, which cannot be parallelized, is not the bottleneck, especially when the number of examples m is large.

3.2 Training General Linear Multi-Class SVM Classifiers

So far we have assumed that (i) the ultimate goal is to minimize the probability of misclassification, (ii) the input observations $\mathbf{x}$ are vectors from $\Re^n$ and (iii) the label y can attain only two values $\{-1,+1\}$. In this section, we will consider the regularized risk minimization framework applied to the learning of a general linear classifier (Tsochantaridis et al., 2005).

We assume that the input observation x is from an arbitrary set $\mathcal{X}$ and the label y can have values from $\mathcal{Y} = \{1,\ldots,Y\}$. In addition, let $\delta\colon \mathcal{Y}\times\mathcal{Y}\to\Re$ be an arbitrary loss function which satisfies $\delta(y,y) = 0$, $\forall y\in\mathcal{Y}$, and $\delta(y,y') > 0$, $\forall (y,y')\in\mathcal{Y}\times\mathcal{Y}$, $y\neq y'$. We consider the multi-class classification rule $h\colon \mathcal{X}\to\mathcal{Y}$ defined as

$$h(x;\mathbf{w}) = \operatorname*{argmax}_{y\in\mathcal{Y}} \langle \mathbf{w}, \Psi(x,y)\rangle\,,$$

where $\mathbf{w}\in\Re^d$ is a parameter vector and $\Psi\colon \mathcal{X}\times\mathcal{Y}\to\Re^d$ is an arbitrary map from the input-output space to the parameter space. Given example set $\{(x_1,y_1),\ldots,(x_m,y_m)\}\in(\mathcal{X}\times\mathcal{Y})^m$, learning the parameter vector $\mathbf{w}$ using the regularized risk minimization framework requires solving problem (1) with the empirical risk $R_{\text{emp}}(h(\cdot;\mathbf{w})) = \frac{1}{m}\sum_{i=1}^{m}\delta(h(x_i),y_i)$. Tsochantaridis et al. (2005) propose two convex piece-wise linear upper bounds on risk $R_{\text{emp}}(h(\cdot;\mathbf{w}))$. The first one, called the *margin re-scaling* approach, defines the proxy risk as

$$R(\mathbf{w}) := \frac{1}{m}\sum_{i=1}^{m}\max_{y\in\mathcal{Y}}\big(\delta(y,y_i) + \langle\Psi(x_i,y) - \Psi(x_i,y_i),\mathbf{w}\rangle\big)\,. \tag{15}$$

The second one, called the *slack re-scaling* approach, defines the proxy risk as

$$R(\mathbf{w}) := \frac{1}{m}\sum_{i=1}^{m}\max_{y\in\mathcal{Y}}\delta(y,y_i)\big(1 + \langle\Psi(x_i,y) - \Psi(x_i,y_i),\mathbf{w}\rangle\big)\,. \tag{16}$$

In the rest of this section we will derive the OCA solver for minimization of the margin re-scaling risk (15). Note that modification of the solver to optimize the slack re-scaling risk (16) is straightforward and that both variants have exactly the same computational complexity. Note also that for the special case when $\delta(y,y')$ is the 0/1-loss, both (15) and (16) become equivalent.

To use the OCA Algorithm 2 for the regularized minimization of (15), we need, first, to derive a procedure performing the line-search (8) required in Step 4 and, second, to derive a formula for the computation of the subgradient of the risk R as required in Step 5. Section 3.2.1 describes an efficient algorithm solving the line-search exactly in $O(m \cdot Y^2 + m \cdot Y \log(m \cdot Y))$ time. The formula for computing the subgradient $\mathbf{a} \in \partial R(\mathbf{w})$ of the risk (15) reads

$$\mathbf{a} = \frac{1}{m} \sum_{i=1}^{m} \left(\Psi(x_i, \hat{y}_i) - \Psi(x_i, y_i) \right),$$

where

$$\hat{y}_i = \operatorname*{argmax}_{y \in \mathcal{Y}} \left(\delta(y_i, y) + \langle \Psi(x_i, y) - \Psi(x_i, y_i), \mathbf{w} \rangle \right).$$

We call the resulting method the *optimized cutting plane algorithm for multi-class SVM* (OCAM). Finally, note that the subtasks of OCAM can be parallelized in a fashion similar to the binary case (see Section 3.1.2).

3.2.1 LINE-SEARCH FOR GENERAL MULTI-CLASS LINEAR SVM CLASSIFIERS

In this section, we derive an efficient algorithm to solve the line-search (8) for the margin re-scaling risk (15). The algorithm is a generalization of the line-search for the binary SVM described in Section 3.1.1. Since the core idea remains the same we only briefly describe the main differences.

The goal of the line-search is to minimize a univariate function $F(\mathbf{w}_{t-1}^b(1-k) + \mathbf{w}_t k)$ defined by (13) with the risk R given by (15). We can abbreviate $F(\mathbf{w}_{t-1}^b(1-k) + \mathbf{w}_t k)$ by $f(k)$ which reads

$$f(k) := f_0(k) + \sum_{i=1}^{m} f_i(k) = \frac{1}{2} k^2 A_0 + k B_0 + C_0 + \sum_{i=1}^{m} \max_{y \in \mathcal{Y}} \left(k B_i^y + C_i^y \right),$$

where the constants A_0, B_0, C_0, $(B_i^y, C_i^y), i = 1, \ldots, m, y \in \mathcal{Y}$ are computed accordingly. Similar to the binary case, the core idea is to find an explicit formula for the subdifferential $\partial f(k)$, which, consequently, allows solving the optimality condition $0 \in \partial f(k)$. For a given $f_i(k)$, let $\hat{\mathcal{Y}}_i(k) = \{ y \in \mathcal{Y} \mid k B_i^y + C_i^y = \max_{y' \in \mathcal{Y}} \left(k B_i^{y'} + C_i^{y'} \right) \}$ be a set of indices of the linear terms which are active at the point k. Then the subdifferential of $f(k)$ reads

$$\partial f(k) = k A_0 + B_0 + \sum_{i=1}^{m} \partial f_i(k) \quad \text{where} \quad \partial f_i(k) = \operatorname{co}\{ B_i^y \mid y \in \hat{\mathcal{Y}}_i(k) \}. \tag{17}$$

The subdifferential (17) is composed of a linear term $kA_0 + B_0$ and a sum of maps $\partial f_i \colon \Re \to I$, $i = 1, \ldots, m$, where I is a set of all closed intervals on a real line. From the definition (17) it follows that ∂f_i is a step-function (or staircase function), that is, ∂f_i is composed of piece-wise linear horizontal and vertical segments. An explicit description of these linear segments is crucial for solving the optimality condition $0 \in \partial f(k)$ efficiently. Unlike the binary case, the segments cannot be computed directly from the parameters $(B_i^y, C_i^y), y \in \mathcal{Y}$, however, they can be found by the simple algorithm described below.

First, we introduce an equivalent representation of ∂f_i. Unlike (17), the new representation explicitly defines intervals where $\partial f_i(k)$ is a constant and the points for which the constant value of

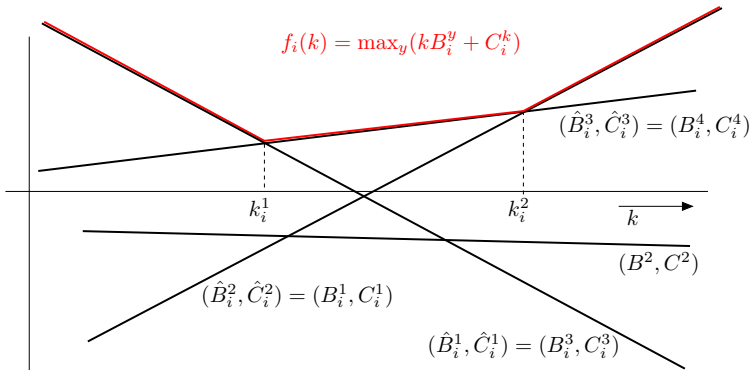

Figure 3: Figure shows an example of the function $f_i(k)$ which is defined as the point-wise maximum over linear terms $kB_i^y + C_i^y$, $y = 1,\ldots,4$. The parameters $(\hat{B}_i^z, \hat{C}_i^z)$, $z = 1,\ldots,3$, and points k_i^z, $z = 1,2$ found by Algorithm 3 are also visualized.

$\partial f_i(k)$ changes. Let $Z \in \{1,\ldots,Y-1\}$ be a given integer and $k_i^1,\ldots,k_i^{Z-1}$ be a strictly increasing sequence of real numbers. Then we define a system of Z open intervals $\{I_i^1,\ldots,I_i^Z\}$ such that

$$I_i^1 = (-\infty, k_i^1), \quad I_i^Z = (k_i^{Z-1}, \infty), \quad \text{and} \quad I_i^z = (k_i^{z-1}, k_i^z), \forall 1 < z < Z.$$

It can be seen that there exist an integer Z and a sequence $k_i^1,\ldots,k_i^{Z-1}$ such that the map ∂f_i can be equivalently written as

$$\partial f_i(k) = \begin{cases} \hat{B}_i^z & \text{if} \quad k \in I_i^z, \\ [\hat{B}_i^z, \hat{B}_i^{z+1}] & \text{if} \quad k \in k_i^z, \end{cases} \tag{18}$$

where $\{\hat{B}_i^1,\ldots,\hat{B}_i^Z\}$ is a subset of $\{B_i^1,\ldots,B_i^Y\}$. Provided the representation (18) is known for all ∂f_i, $i = 1,\ldots,m$, the line-search $k^* = \operatorname{argmin}_{k>0} f(k)$ can be solved exactly by finding the intersection of $\partial f(k)$ and the x-axis, that is, solving the optimality condition $0 \in \partial f(k)$. To this end, we can use the same algorithm as in the binary case (see Section 3.1.1). The only difference is that the number of points k_i^z in which the subdifferential $\partial f(k)$ changes its value is higher; $m \cdot (Y-1)$ in the worst case. As the computations of the algorithm for solving $0 \in \partial f(k)$ are dominated by sorting the points k_i^z, the worst-case computational complexity is approximately $O(m \cdot Y \log(m \cdot Y))$.

Finally, we introduce Algorithm 3, which finds the required representation (18) for a given ∂f_i. In the description of Algorithm 3, we do not use the subscript $_i$ to simplify the notation. Figure 3 shows an example of input linear terms (B_i^y, C_i^y), $y \in \mathcal{Y}$ defining the function $f_i(k)$ and the sorted sequence of active terms $(\hat{B}_i^z, \hat{C}_i^z), z = 1,\ldots,Z$, and points k_i^z, $z = 1,\ldots,Z$, in which the activity of the linear terms changes. At the beginning, the algorithm finds a linear term which is active in the leftmost interval $(-\infty, k^1)$, that is, the line with the maximal slope. Then the algorithm computes intersections with the leftmost active linear term that was found and the remaining lines with lower slopes. The leftmost intersection identifies the next active term. This process is repeated until the rightmost active term is found. The worst-case computational complexity of Algorithm 3 is $O(Y^2)$. In turn, the total complexity of the line-search procedure is $O(m \cdot Y^2 + m \cdot Y \log(m \cdot Y))$, that is, $O(m \cdot Y^2)$ time is required for running Algorithm 3 m times and $O(m \cdot Y \log(m \cdot Y))$ time for solving the optimality condition $0 \in \partial f_i(k)$ as described above.

Algorithm 3 Finding explicit piece-wise linear representation (18) of ∂f_i

Require: (B^y, C^y), $y \in \mathcal{Y}$
Ensure: Z, $\{\hat{B}^1, \ldots, \hat{B}^Z\}$, and $\{k^1, \ldots, k^{Z-1}\}$
1: $\hat{y} := \text{argmax}_{y \in \hat{\mathcal{Y}}} C^y$ where $\hat{\mathcal{Y}} := \{y \mid B^y = \min_{y' \in \mathcal{Y}} B^{y'}\}$.
2: $Z := 1$, $k := -\infty$ and $\hat{B}^1 := B^{\hat{y}}$
3: **while** $k < \infty$ **do**
4: $\quad \hat{\mathcal{Y}} := \{y \mid B^y > B^{\hat{y}}\}$
5: $\quad$ **if** $\hat{\mathcal{Y}}$ is empty **then**
6: $\quad\quad k := \infty$
7: $\quad$ **else**
8: $\quad\quad \ddot{y} := \text{argmin}_{y \in \hat{\mathcal{Y}}} \frac{C^y - C^{\hat{y}}}{B^y - B^{\hat{y}}}$
9: $\quad\quad k^Z := \frac{C^{\ddot{y}} - C^{\hat{y}}}{B^{\ddot{y}} - B^{\hat{y}}}$
10: $\quad\quad Z := Z + 1$
11: $\quad\quad \hat{B}^Z := B^{\ddot{y}}$
12: $\quad\quad \hat{y} := \ddot{y}$
13: $\quad$ **end if**
14: **end while**

Note that the described algorithm is practical only if the output space $\mathcal{Y}$ is of moderate size since the complexity of the line-search grows quadratically with $Y = |\mathcal{Y}|$. For that reason, this algorithm is ineffective for structured output learning where the cardinality of $\mathcal{Y}$ grows exponentially with the number of hidden states.

4. Experiments

In this section we perform an extensive empirical evaluation of the proposed optimized cutting plane algorithm (OCA) applied to linear binary SVM classification (OCAS) and multi-class SVM classification (OCAM) .

In particular, we compare OCAS to various state-of-the-art SVM solvers. Since several of these solvers did not take part in the large-scale learning challenge, we perform an evaluation of SVM$^{\text{light}}$, Pegasos, GPDT, SGD, BMRM, SVM$^{\text{perf}}$ version 2.0 and version 2.1 on previously published medium-scale data sets (see Section 4.1.1). We show that OCAS outcompetes previous solvers gaining speedups of several orders of magnitude over some of the methods and we also analyze the speedups gained by parallelizing the various core components of OCAS.

In addition, we use the challenge data sets and follow the challenge protocol to compare OCAS with the best performing methods, which were LaRank and LibLinear (see Section 4.1.2). Finally, in section 4.2, we compare the multi-class SVM solver OCAM to the standard CPA implemented for multi-class SVM on four real-world problems using the challenge evaluation protocol.

4.1 Comparison of Linear Binary SVM

We first compare OCAS with several binary linear SVM solvers on previously published data sets followed by an analysis using the challenge criteria on the challenge data sets.

4.1.1 EVALUATION ON PREVIOUSLY USED DATA SETS

We now compare current state-of-the-art SVM solvers (SGD, Pegasos, SVM^{light} , SVM^{perf} , BMRM, GPDT[1]), on a variety of data sets with the proposed method (OCAS), using 6 experiments measuring:

1. Influence of the hyper-parameter μ on the speed of convergence
2. Training time and objective for optimal C
3. Speed of convergence (time vs. objective)
4. Time to perform a full model selection
5. Effects of parallelization
6. Scalability w.r.t. data set size

To this end, we implemented OCAS and the standard CPA[2] in C. We use the very general compressed sparse column (CSC) representation to store the data. Here, each element is represented by an index and a value (each 64bit). To solve the reduced problem (2), we use our implementation of improved SMO (Fan et al., 2005). The source code of OCAS is freely available for download as part of LIBOCAS (Franc and Sonnenburg, 2008b) and as a part of the SHOGUN toolbox (Sonnenburg and Rätsch, 2007).

All competing methods train SVM classifiers by solving the convex problem (1) either in its primal or dual formulation. Since in practice only limited precision solutions can be obtained, solvers must define an appropriate stopping condition. Based on the stopping condition, solvers can be categorized into *approximative* and *accurate*.

Approximative Solvers make use of heuristics (e.g., learning rate, number of iterations) to obtain (often crude) approximations of the optimal solution. They have a 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 and Bousquet, 2007), 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 a precise solution. Among the most efficient solvers to-date are Pegasos (Shwartz et al., 2007) and SGD (Bottou and Bousquet, 2007), both of which are based on stochastic (sub-)gradient descent.

Accurate Solvers In contrast to approximative solvers, accurate methods solve the optimization problem 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. Among the state-of-the-art accurate solvers are SVM^{light} , SVM^{perf} , BMRM and GPDT.

Because there is no widely accepted consensus on which approach is "better", we used both types of methods in our comparison.

1. Solvers include: SGD version 1.1 (svmsgd2) `http://leon.bottou.org/projects/sgd`, SVM^{light} 6.01 and SVM^{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` and GPDT `http://dm.unife.it/gpdt`.
2. To not measure implementation specific effects (solver, dot-product computation etc.).

Experimental Setup We trained all methods on the data sets summarized in Table 2. We augmented the Cov1, CCAT, Astro data sets from Joachims (2006) by the MNIST, an artificial dense data set and two larger bioinformatics splice site recognition data sets for worm and human.[3]

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

Table 2: Summary of the data sets used in the experimental evaluation. Sparsity denotes the average number of non-zero elements of a data set in percent. Split describes the size of the train/validation/test sets in percent.

These data sets have been used and are described in detail in Joachims (2006), Shwartz et al. (2007) and Franc and Sonnenburg (2008a). The Covertype, Astrophysics and CCAT data sets were provided to us by Shai Shalev-Shwartz and should match the ones used in Joachims (2006). The Worm splice data set was provided by Gunnar Rätsch. We did not apply any extra preprocessing to these data sets.[4]

The artificial data set was generated from two Gaussian distributions with different diagonal covariance matrices of multiple scale. Unless 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 that is implemented in SVM$^{\text{perf}}$, that is, $\frac{F(\mathbf{w})-F_t(\mathbf{w})}{C} \leq \frac{\varepsilon_{perf}}{100} = 10^{-3}$. Note that these ε have very different meanings denoting the maximum KKT violation for SVM$^{\text{light}}$, the maximum tolerated violation of constraints for SVM$^{\text{perf}}$ and for 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 the experiments we use C as a shortcut for C_{light}.[5]

Influence of the Hyper-parameter μ on the Speed of Convergence In contrast to the standard CPA, OCAS has a single hyper-parameter μ (see Section 3). The value of μ determines the point $\mathbf{w}_t^c = \mathbf{w}_t^b(1-\mu) + \mathbf{w}_t\mu$ at which the new cutting plane is selected. The convergence proof (see Theorem 2) requires μ to be from the interval $(0,1]$, however, the theorem does not indicate which value is the optimal one. For this reason, we empirically determined the value of μ.

For varying $\mu \in \{0.01, 0.05, 0.1, \ldots, 1\}$ we measured the time required by OCAS to train the classifier on the Astro, CCAT and Cov1 data sets. The regularization constant C was set to the

3. Data sets found at: Worm and Human `http://www.fml.tuebingen.mpg.de/raetsch/projects/lsmkl`, Cov1 `http://kdd.ics.uci.edu/databases/covertype/covertype.html`, CCAT `http://www.daviddlewis.com/resources/testcollections/rcv1/`, MNIST `http://yann.lecun.com/exdb/mnist/`.
4. However, we noted that the Covertype, Astro-ph and CCAT data set already underwent preprocessing because the latter two have $\|x_i\|_2 = 1$.
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`.

optimal value for the given data set. Figure 4 shows the results. For Astro and CCAT the optimal value is $\mu = 0.1$ while for Cov1 it is $\mu = 0.01$. For all three data sets the training time does not change significantly within the interval $(0, 0.2)$. Thus we selected $\mu = 0.1$ to be the best value and we used this setting in all remaining experiments.

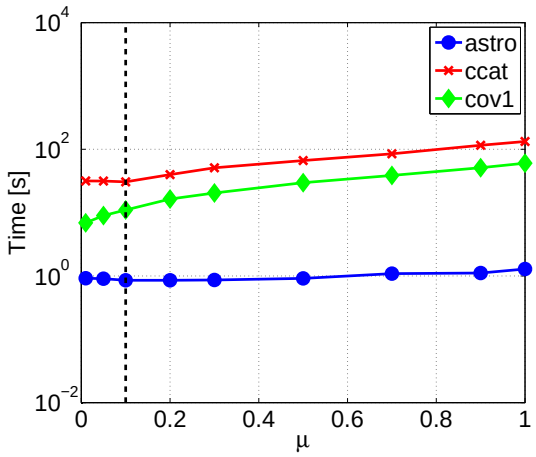

Figure 4: Training time vs. value of the hyper-parameter μ of the OCAS solver measured on the Astro, CCAT and Cov1 data sets. The value $\mu = 0.1$ (dash line) is used in all remaining experiments.

Training Time and Objective For Optimal C We trained all methods on all except the human splice data set using the training data and measured training time (in seconds) and computed the unconstrained objective value $F(\mathbf{w})$ (cf. Equation 11).

The results are displayed in Table 3. The proposed method—OCAS—consistently outperforms all its competitors in the *accurate solver* category on all benchmark data sets 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$^{perf2.0}$—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 $\mathbf{w}^*$. However, as training times for SGD are significantly below all others, it is unclear whether SGD achieves the same precision using less time with 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, that is, in the course of optimization record time and objective for several time points. The results are shown in Figure 5. OCAS was stopped when reaching the maximum time or a precision of $1 - F(\mathbf{w}^*)/F(\mathbf{w}) \leq 10^{-6}$ and in all cases achieved the minimum objective. In three of the six data sets, OCAS not only, achieves the

6. In contrast to SVM$^{\text{perf}}$, BMRM and our implementation of CPA did not converge for large C on Worm even after 5000 iterations. Most likely, the core solver of SVM$^{\text{perf}}$ is more robust.

	Astro	CCAT	Cov1	MNIST	Worm	Artificial
svmlight	**2.0939e+03**	**8.1235e+04**	**2.5044e+06**	**6.7118e+05**	**3.1881e+04**	**1.3170e+02**
	2972 22	77429 5295	1027310 41531	622391 2719	2623193 44852	231059 3060
svmperf2.1	**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
svmperf2.0	**2.1188e+03**	**8.1760e+04**	**2.5071e+06**	**6.7276e+05**	**3.2327e+04**	**1.3182e+02**
	-1 11	-1 1250	-1 345	-1 6115	-1 16515	-1 455
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
gpdt	**1.1725e+03**	**1.5418e+05**	**1.3034e+06**	**5.9796e+06**	**1.3205e+04**	**1.2642e+02**
	130 5	3570 2263	4844 1794	526 118	38092 39095	615 137

Table 3: Training time for optimal C comparing OCAS with other SVM solvers. ”-” means not converged, blank not attempted. Shown in bold is the unconstrained SVM objective value Eq. (11). The two numbers below the objective value denote the number of iterations (left) and the training time in seconds (right). Lower time and objective values are better. All methods solve the unbiased problem. As convergence criteria, the standard settings described in Section 4.1.1 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 the lowest objective value. The objective value obtained by SGD and Pegasos is often far away from the optimal solution; see text for a further discussion.

best objective as expected at a later time point, but already from the very beginning. Further analysis made clear that OCAS wins over SGD in cases where *large C*s 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 (11) but the performance on a validation set, that is, optimization is stopped when the validation error does not change. This has been discussed for leave-one-out in Franc et al. (2008) and we—to some extent—agree with this. One should, however, note that in this case one does not obtain an SVM but *some classifier* instead. A comparison should not then be limited to SVM solvers but should also be open to any other large scale approach, like online algorithms (e.g., perceptrons). We argue that to compare *SVM solvers* in a fair way one needs to compare objective values. We therefore ran Pegasos using a larger number of iterations on the Astro and splice data sets. On the Astro data set, Pegasos surpassed the SVM$^{\text{light}}$ objective after 10^8 iterations, requiring 228 seconds. SVM$^{\text{light}}$, in comparison, needed only 22 seconds. Also, on the splice data set we ran Pegasos for 10^{10} iterations, which took 13,000 seconds and achieved a similar objective as that of SVM$^{perf2.0}$, requiring only 1224 seconds. Finally, note that, although BMRM, SVM$^{\text{perf}}$ and our implementation of CPA solve the same equivalent problem using the CPA, differences in implementations lead to varying results.[7] Since it

7. Potentially due to a programming error in this pre-release of BMRM, it did not show convergence on the splice data set even after $>$ 6500 iterations.

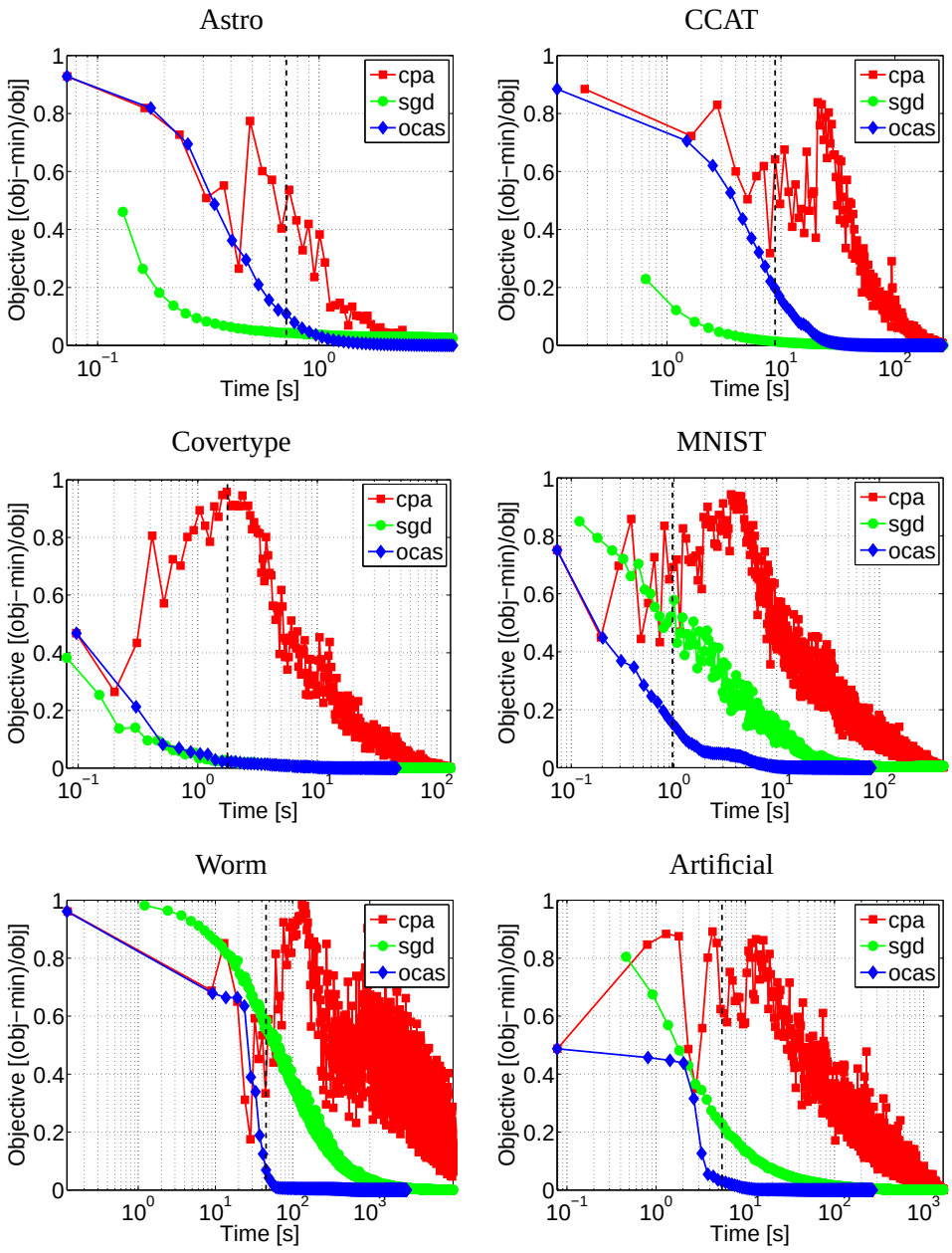

Figure 5: Objective value vs. training time of CPA (red), SGD (green) and OCAS (blue) measured for different numbers of training examples.The dashed line shows the time required to run SGD for 10 iterations. OCAS was stopped when the precision fell below 10^{-6} or the training time for CPA was achieved. In all cases, OCAS achieves the minimal objective value and, even from the beginning, outperforms all other methods, including SGD, on half of the data sets.

is still interesting to see how the methods perform w.r.t. classification performance, we describe the analysis 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 during model selection. We therefore train all methods using different settings[8] for C on the training part of all data sets, evaluate them on the validation set and choose the best model to do predictions on the test set. As the performance measure, we use the area under the receiver operator characteristic curve (auROC) (Fawcett, 2003). Results are displayed in Table 4.

	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	0.5	86694	0.005	42491
svmperf2.0	1	67	1	20827	50	1765	5	21113	5	106241	0.005	111621
svmperf2.1	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
gpdt	1	30	1	33693	5	11615	10	408	0.5	283941	0.005	90807

Table 4: Model selection experiment comparing OCAS with 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 chosen based on 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. Because 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 the artificial data set. However, often a C smaller 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 3 for training times for the optimal C). For further discussion see the text.

Again, among the *accurate methods* OCAS outperforms its competitors by a large margin, followed by SVM$^{\text{perf}}$. Note that for all *accurate methods* the performance is very similar and has little variance. Except for the artificial data set, 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 a lower C, the objective function is dominated by the regularization term $\|w\|$. A potential explanation is that SGD's update rule puts more emphasis on the regularization term, and SGD, when not run for a large number of iterations, does imply early stopping.

Our suggestion for practitioners is to use OCAS whenever a reliable and efficient large-scale solver with proven convergence guarantees is required. This is typically the case when the solver is

8. For Worm and Artificial we used $C \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5\}$, for CCAT, Astro, Cov1 $C \in \{0.1, 0.5, 1, 5, 10\}$ and for MNIST $C \in \{1, 5, 10, 50, 100\}$.

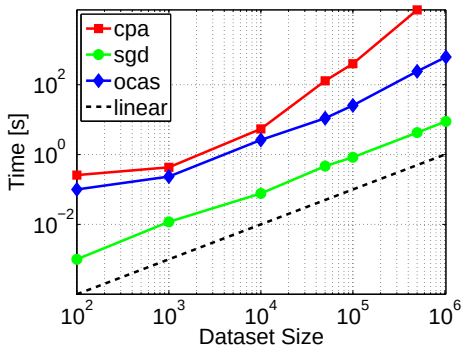

Figure 6: This figure displays how the methods scale with data set size on the Worm splice data set. The slope of the "lines" in this figure denotes the exponent e in $O(m^e)$, where the black line denotes linear effort $O(m)$.

to be operated by non-expert users who know little (or nothing) about tuning the hyper-parameters of the optimization algorithm. Therefore, as long as the full data set fits into memory, we recommend OCAS. Otherwise, if sub-sampling is not an option, online approximative solvers like SGD are the only viable way to proceed.

Effects of Parallelization As OCAS training times are very low on the above data sets, we also apply OCAS to the 15 million human splice data set. 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 speedup w.r.t. the single CPU algorithm in Table 5. Also shown is the accumulated time for each of the threads. As can be seen—except for the line-

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

Table 5: Speedups due to parallelizing OCAS on the 15 million human splice data set.

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).

Scalability w.r.t. Data Set Size In this section, we investigate how computational times of OCAS, CPA and SGD scale with the number of examples on the Worm splice data set for sizes 100 to $1,026,036$. Results are shown in Figure 6. We again use our implementation of CPA which shares essential sub-routines with OCAS. Both OCAS and SGD scale roughly linearly. Note that SGD is much faster (because it runs for a fixed number of iterations and thus stops early).

4.1.2 EVALUATION ON CHALLENGE DATA

In this section, we use the challenge data sets and follow the challenge protocol to compare OCAS to the best-performing methods, which were LaRank (Bordes et al., 2007) and LibLinear (Fan et al., 2008). To this end, we apply OCAS to the challenge data sets Alpha, Gamma and Zeta following the challenge protocol for the SVM track.

The data sets are artificially generated based on a mixture of Gaussians and have certain properties (see Table 6): The Alpha data set is separable with a large margin using quadratic features.

Data Set	Optimal Model	Number of examples training	testing	validation	Dim.	Description
Alpha	quadratic	500,000	300,000	100,000	500	well separable
Gamma	semi-quadratic	500,000	300,000	100,000	500	Multiscale low var.
Zeta	linear	500,000	300,000	100,000	2000	Intrinsic dim. 400

Table 6: Summary of the three challenge data sets used: Alpha, Gamma, Zeta.

The Gamma data set is well separable too, but contains features living on different scales. Finally, the optimal model for Zeta is a linear classifier—of its 2,000 features 1,600 are nuisance dimensions. The challenge protocol requires training on the unmodified data sets with $C = 0.01$ and precision $\varepsilon = 0.01$. To measure convergence speed, objective values are measured *while training*. The second challenge experiment simulates model selection by training for different $C \in \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

The left column in Figure 7 displays the course of convergence of the three methods. While OCAS is quite competitive on Gamma and Zeta in this experiment, it is slower on Alpha. It should also be noted that OCAS, in contrast to the online-style algorithms LaRank and LibLinear, has to do a full pass through the data in each iteration. However, it usually requires very few iterations to obtain precise solutions.

In the simulated model selection experiment (right column of Figure 7), OCAS performs well for low values of C on all data sets. However, at first glance it is competitive for large values of C only on Zeta. Investigating objective values on Gamma for LibLinear, we noticed that they significantly deviate (objective values much larger, deviation by 50% for $C = 10$) from LaRank/OCAS for $C \in \{1, 10\}$. Still, on Alpha OCAS is slower.

4.2 Comparison of Linear Multi-Class SVMs

In this section, we compare the proposed multi-class SVM solver OCAM described in Section 3.2 with multi-class CPA (CPAM). We consider the Crammer and Singer (2001) formulation of multi-class SVMs which corresponds to the minimization of the following convex objective,

$$F(\mathbf{w}) := \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m} \max_{y \in \mathcal{Y}} \left([\![y_i \neq y]\!] + \langle \mathbf{w}_y - \mathbf{w}_{y_i}, \mathbf{x}_i \rangle \right), \tag{19}$$

where $\mathbf{w} = [\mathbf{w}_1, \ldots, \mathbf{w}_Y]$ is a matrix of parameter vectors and $\{(\mathbf{x}_1, y_i), \ldots, (\mathbf{x}_m, y_m)\} \in (\Re^n \times \mathcal{Y})^m$ is a set of training examples. The multi-class classification rule then reads $h(\mathbf{x}) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle \mathbf{x}, \mathbf{w}_y \rangle$.

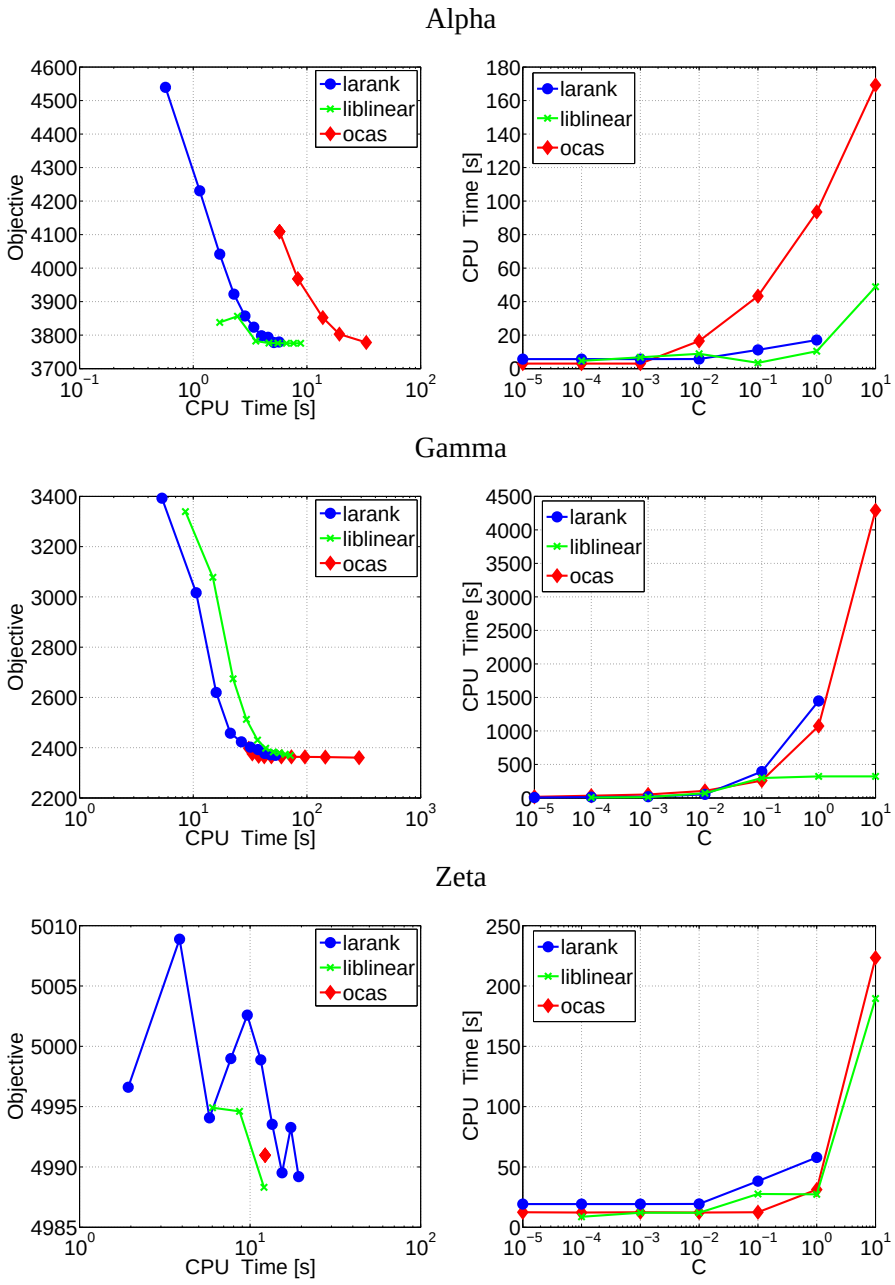

Figure 7: Results of LaRank, LibLinear and OCAS on the Alpha, Gamma and Zeta challenge data sets. The left column of the figures displays the unconstrained SVM primal objective (11) (C is not scaled with m) w.r.t. SVM training time for fixed $C = 0.01$. The right column displays the SVM training time for different C. We omitted the data set size vs. CPU time figure since all three methods show a similar curve (a line with slope ≈ 1 in log-log representation, corresponding to the expected $O(m)$ effort) . Note that for the Zeta data set OCAS converges after a single pass through the data, which results in collapsing the performance curve into a single point. For low C, OCAS achieves very competitive results. For further explanation see text.

We implemented OCAM and CPAM in C, exactly according to the description in Section 3.2. Both implementations use the Improved Mitchel-Demyanov-Malozemov algorithm (Franc, 2005) as the core QP solver and they use exactly the same functions for the computation of cutting planes and classifier outputs. The implementation of both methods is freely available for download as part of LIBOCAS (Franc and Sonnenburg, 2008b). The experiments are performed on an AMD Opteron-based 2.2GHz machine running Linux.

In the evaluation we compare OCAM with CPAM to minimize programming bias. In addition, we perform a comparison with SVM$^{\text{multi-class}}$ v2.20 later in Section 5.2.

We use four data sets with inherently different properties that are summarized in Table 7. The Malware data set is described in Section 5.2. The remaining data sets, MNIST, News20 and Sector, are downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`. We used the versions with the input features scaled to the interval $[0,1]$. Each data set is randomly split into a training and a testing part.

	features		number of	num. of examples	
	number of	type	classes	training	testing
Malware	3,413	dense	14	3,413	3,414
MNIST	780	dense	10	60,000	10,000
News20	62,060	sparse	20	15,935	3,993
Sector	55,197	sparse	105	6,412	3,207

Table 7: Multi-class data sets used in the comparison of OCAM and CPAM.

	Malware		MNIST		News20		Sector	
	error	time	error	time	error	time	error	time
Standard CPAM	10.25	12685	7.07	15898	14.45	7276	5.58	12840
Proposed OCAM	10.16	1705	7.08	5387	14.45	1499	5.61	3970
speedup		7.4		3.0		4.9		3.2

Table 8: Comparison of OCAM and CPA on a simulated model selection problem. The reported time corresponds to training over the whole range of regularization constants *C*s. The error is the minimal test classification over the classifiers trained with different *C*s.

In the first experiment, we train the multi-class classifiers on training data with a range of regularization constants $C = \{10^0, 10^1, \ldots, 10^7\}$ (for Malware $C = \{10^0, \ldots, 10^8\}$ since the optimal $C = 10^7$ is the boundary value). Both solvers use the same stopping condition (10) with $\varepsilon = 0.01F(\mathbf{w})$. We measure the total time required for training over the whole range of Cs and the best classification error measured on the testing data. Table 8 summarizes the results. While the classification accuracy of OCAM and CPAM are comparable, OCAM consistently outperforms the standard CPAM in terms of runtime, achieving speedup of factor from 3 to 7.4.

In the second experiment, we measure the three performance figures defined in the large-scale challenge: (i) the objective value as a function of the runtime, (ii) the runtime as a function of C and (iii) the runtime as a function of the data set size. For figures (i) and (ii) we use the optimal C obtained in the first experiments. Results for the first three data sets are shown in Figure 8.

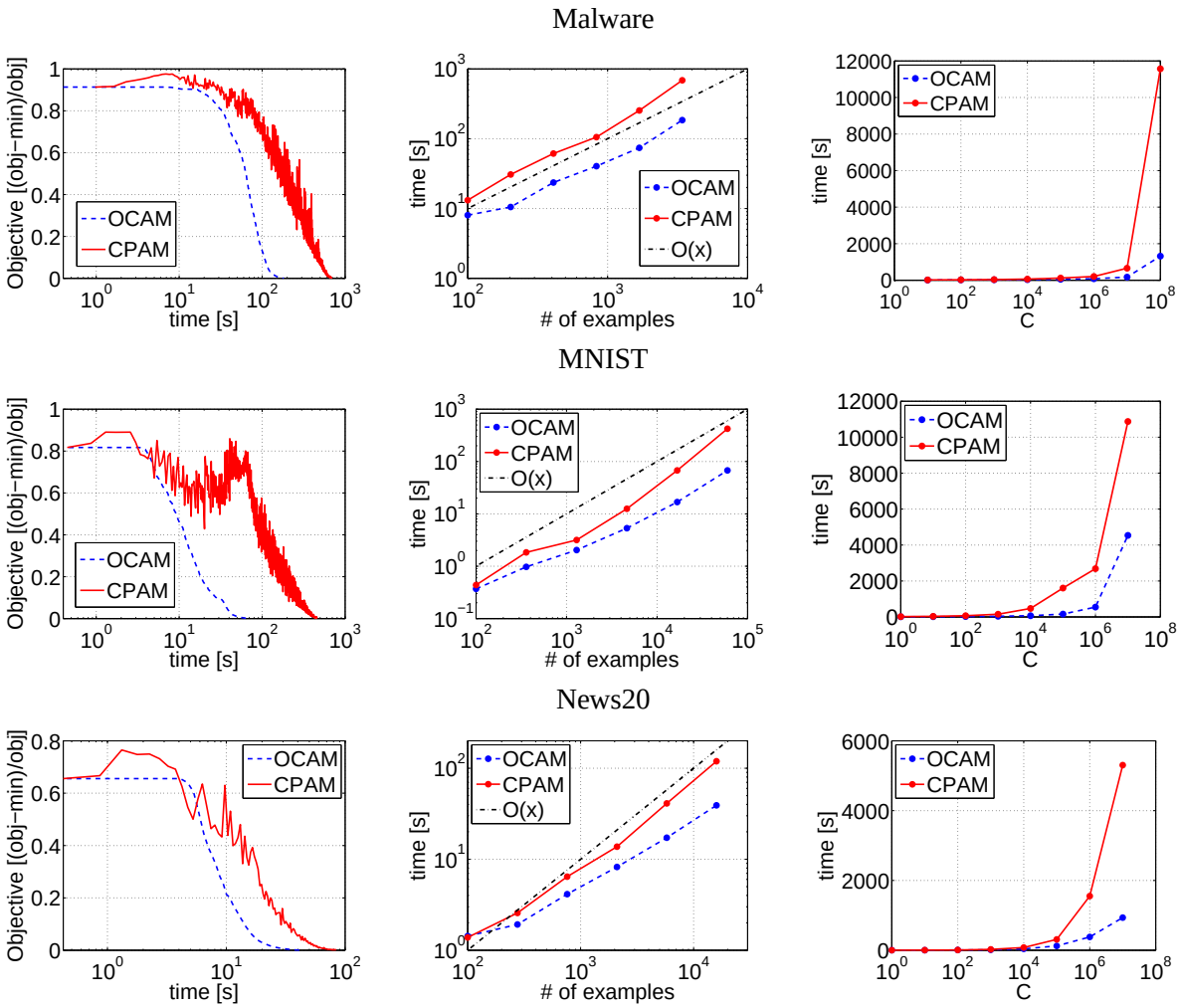

Figure 8: Results of the standard CPAM and the proposed OCAM on the Malware, MNIST and News20 data sets. The left column of figures displays the unconstrained SVM objective (19) w.r.t. SVM training time. The middle column displays the training time as a function of the number of examples. In both experiments C was fixed to its optimal value as determined in model selection. The right column shows the training time for different Cs. See text for a discussion of the results.

The objective vs. time figure is consistent with the results obtained in Section 4.1 for the two-class variant, that is, the objective value of the standard CPAM significantly fluctuates while OCAM decreases the objective monotonically and converges faster in all cases. The data size vs. time figure shows that in both cases the runtime is approximately linear w.r.t. the number of examples, though the curve of CPAM grows slightly faster compared to OCAM. The main difference shows the figure depicting the runtime as a function of C. It is seen that OCAM is considerably faster for large values of C, which is crucial for efficient model selection (see the experiment in Section 5.2) .

5. Applications

In this section we attack two real-world applications. First, in Section 5.1, we apply OCAS to a human acceptor splice site recognition problem. Second, in Section 5.2, we use OCAM for learning a behavioral malware classifier.

5.1 Human Acceptor Splice Site Recognition

To demonstrate the effectiveness of our proposed method, OCAS, we apply it to the problem of human acceptor splice site detection. Splice sites mark the boundaries between potentially protein-coding exons and (non-coding) introns. In the process of translating DNA to protein, introns are excised from pre-mRNA after transcription (Figure 9). Most splice sites are so-called *canonical splice sites* that are characterized by the presence of the dimers `GT` and `AG` at the donor and acceptor sites, respectively.

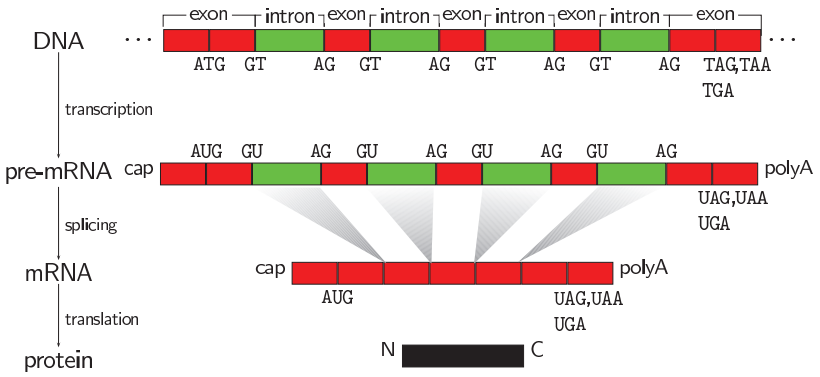

Figure 9: The major steps in protein synthesis. In the process of converting DNA to messenger RNA, the introns (green) are spliced out. Here we focus on detecting the so-called acceptor splice sites that employ the `AG` consensus and are found at the "left-hand side" boundary of exons. Figure taken from (Sonnenburg, 2002).

However, the occurrence of the dimer alone is not sufficient to detect a splice site. The classification task for splice site sensors, therefore, consists in discriminating true splice sites from decoy sites that also exhibit the consensus dimers. Assuming a uniform distribution of the four bases, adenine (`A`), cytosine (`C`), guanine (`G`) and thymine (`T`), one would expect 1/16th of the dimers to contain the `AG` acceptor splice site consensus. Considering the size of the human genome, which consists of about 3 billion base pairs, this constitutes a large-scale learning problem (the expected number of `AG`s is 180 million).

Many different methods to detect splice sites have been proposed. They all predict splice sites based on the local context, that is, a short window around the `AG` dimer. Currently, support vector machines are the most accurate splice site detectors (Degroeve et al., 2005; Sonnenburg et al., 2007b). Sonnenburg et al. (2007b) showed that prediction accuracy steadily increases with training sample size. However, even though they already used the `linadd` algorithm (Sonnenburg et al., 2007a) to speed up string kernel-based SVMs on a quad-core system, they could not use all available 50 million training points (but "only" 8 million). The string kernel that performed best was the

weighted degree (WD) string kernel *with shifts* (Rätsch et al., 2005). It basically counts matching k-mers for various k in a position-dependent way. Employing a giant string kernel feature space, Sonnenburg et al. (2007b) achieved $45.58\% \pm 0.38$ aoPRC in a genome-wide study on human acceptor splice sites—also available as the DNA data set used in the large-scale learning challenge.

On the other hand, Degroeve et al. (2005) trained a linear SVM based on a number of preselected and explicitly computed string kernel feature spaces that are subsets from the spectrum (Leslie et al., 2002) and WD kernel (Rätsch et al., 2005) feature spaces: Left and right of the splice site spectrum kernels of order 3 up to order 6 were used (Leslie et al., 2002). Over the whole window, a WD kernel of order 3 with weights equal to 1 was used (Rätsch and Sonnenburg, 2004). Even though this approach scales well, they used $< 100,000$ data points (potentially, since they relied on the unmodified SVM$^{\text{light}}$ binary).

Here, we propose to train OCAS on all available 50 million strings of length 141 from Sonnenburg et al. (2007b) using the features corresponding to two weighted spectrum kernels (one left and one right of the splice site, that is, positions 1-59 and 62-141) and a WD kernel (applied to the whole string). For the spectrum kernels of order 1 up to 8 and for the WD kernel of order 8 is used. Thus, the spanned string kernel feature space has 12,495,340 dimensions.

As the raw string-based data set already has a size of $7.1 \cdot 10^9$ bytes and even a sparse representation of each string would increase the data set by a factor of more than 3,000 ($(141+59+80) \cdot 12$ bytes per feature vector, assuming a 4 byte integer and an 8 byte float), we will *implicitly* compute features from the raw input strings on demand. The only required operations in OCAS for which we will have to expand the features are the addition to a dense vector $\mathbf{w} \leftarrow \mathbf{w} + \alpha\Phi(x)$ and the output computation $\mathbf{w} \cdot \Phi(x)$.

We implemented a rather general framework that allows stacking of arbitrary features that support such operations (dense and sparse real-valued, weighted spectrum and WD kernel features for specified k-mer length). As we know from Section 4.1.1, most time is spent in computing outputs, hence we parallelized this part of the code (based on shared memory parallelization, that is, posix threads).

Before training on the 50 million examples, we perform model selection on only 1 million examples to determine the optimal k-mer length for the two spectrum kernels, the WD kernel and its weighting and the SVM regularization constant C. The optimal parameter setting was found to be $C = 1$, $k_{wspec} = 8$, $k_{wd} = 8$, where the WD kernel weights are taken from the first 8 weights of the weights of a wd kernel of order 40.[9] Parameters were selected from $C \in \{0.5, 1, 3, 5, 10\}$, $k_{wspec} \in \{3, 6, 8\}$, $k_{wd} = \{3, 6, 8\}$ with the WD kernel-weighting from order 8, 25 or 40.

We then trained on 50 million examples on an 8-core AMD Opteron Linux-based machine, obtaining a record area over the precision recall curve (aoPRC) of 42.23%. For comparison, the previous best method achieved aoPRC of 45.58% (variance 0.38%). Note that this is the DNA data set used in the large-scale learning challenge, for which the best participant obtained an aoPRC of 80.89% (lower is better). OCAS converged in just 138 iterations, however, the total training time was about 40 hours, of which almost 34 hours were spent on computing outputs (already in a parallelized way; see Table 9 for the detailed timing statistics). Even though we observed that this parallelization was quite effective, it suggests that we are measuring random memory access speed. Due to the size of the normal vector (about 100MB since we are using double precision floats) we see only cache misses. This suggests that even using just single precision floats would reduce the

9. In Sonnenburg (2008) it was suggested that the WD kernel-weighting influences the effect of mismatches of the WD kernel score.

training time by 17 hours. Even though modern DDR-SDRAM is capable of speeds of up to 8 GB/s (Wikipedia, 2009) when being accessed in a linear way, we observed a memory speed of only 1.4GB/s on this system. It turns out that only DDR-333 memory is installed with a peak transfer rate of 2.7GB/s. Thus, additional speedups can be achieved by distributed memory parallelization and by grouping the access of features in $\mathbf{w}$ to minimize cache misses. Alternatively, switching to a many-core architecture like the NVIDIA Tesla s1070 computing system[10] that employs 960 CPU cores and a peak memory rate of 400GB/s could drastically reduce training times, potentially to even under 1 minute. Finally, it should be noted that storing the 138 cutting planes required almost 13 GB of memory.

Iterations	Output	Line Search	Add $\mathbf{a}_t$	Solver	Total
138	34 hours	222s	7 hours	5min	41 hours

Table 9: Timing statistics for the human acceptor splice site experiment.

5.2 Malware Classification

Malware is malicious software that occurs in the form of Internet worms, computer viruses and Trojan horses. Due to an enormous increase of new variants of malware, methods for its automatic detection and categorization are becoming crucial in modern anti-malware products. Rieck et al. (2008) propose a malware behavioral classifier trained from labeled examples. Malware binaries are collected via honeypots and spam-traps, and malware family labels are generated by running an anti-virus tool. This results in a corpus of more than 10,000 unique malware instances. The behavior of each binary is monitored in a sand box environment and behavior-based analysis reports summarizing operations, such as opening an outgoing IRC connection or stopping a network service, are generated. The reports have a form of text files which are then embedded into a high-dimensional vector space using the bag-of-words model. Finally, a discriminative multi-class SVM classifier is trained.

Rieck et al. (2008) use the multi-class classifier based on one-against-all decomposition, where each binary classifier is trained by a kernel SVM. To increase classification performance, the scale of the independently trained binary discriminant functions, forming the multi-class classifier, is normalized by fitting a logistic function. Rieck et al. (2008) report promising results achieving 88% classification accuracy, which is competitive with commercial anti-virus software tools handcrafted manually by computer security experts. Apart from the classification accuracy, the ability to retrain swiftly on new examples is a crucial feature for practical application of the system. While the classification accuracy was the main focus in Rieck et al. (2008), the issue of fast training was not addressed. The SVM$^{\text{light}}$ that they used required approximately 13-14 hours to perform the whole model selection on a high-end single CPU computer.[11]

To resolve the problem of fast training, we apply our proposed OCAM solver and compare its performance with SVM$^{\text{multi-class}}$ (Joachims et al., 2009). SVM$^{\text{multi-class}}$ version 2.20[12] is a highly optimized implementation of CPAM which uses numerous heuristic speedups like adaptive accu-

10. Found at `http://www.nvidia.com/object/product_tesla_s1070_us.html`.
11. Personal communication with authors of Rieck et al. (2008).
12. Found at `http://www.cs.cornell.edu/People/tj/svm_light/svm_multiclass.html`.

racy management, caching or 1-slack reformulation (for more details see Joachims et al. 2009). Note that OCAM is the plain implementation of the proposed Algorithm 2, hence there is still the possibility to improve its performance by implementing the same heuristics. SVM$^{\text{multi-class}}$ optimizes a slightly modified risk $R'(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \max_{y\in\mathcal{Y}} \left(100[\![y_i \neq y]\!] + \langle \mathbf{w}_y - \mathbf{w}_{y_i}, \mathbf{x}_i\rangle\right)$. To make objectives of SVM$^{\text{multi-class}}$ and OCAM equivalent we use the transform: $\mathbf{x}_i = \mathbf{x}'_i/100$ and $C = 100C'$ where $\mathbf{x}'_i$ and C' denote inputs and the regularization constant used by SVM$^{\text{multi-class}}$. SVM$^{\text{multi-class}}$ stops optimization when $F(\mathbf{w}_t) - F_t(\mathbf{w}_t) \leq C\varepsilon'$, hence we apply $\varepsilon = C\varepsilon'$ in OCAM to use equivalent stopping conditions. In addition, we use $\varepsilon' = 0.1$, which is the default setting in SVM$^{\text{multi-class}}$.

solver	error [%]	training time	
SVM$^{\text{multi-class}}$ v2.20	11.45 ± 0.72	25,330 sec	$\approx$ 7 hours
OCAM	11.49 ± 0.91	2,451 sec	$\approx$ 40 minutes
Rieck et al. (2008)	12		$\approx$ 13-14 hours

Table 10: Comparison of SVM$^{\text{multi-class}}$ v2.20 with the proposed OCAM on the malware classification problem. The reported error is a 5-fold cross-validation estimate of the per-class average classification error. The training time refers to the total time required by model selection. We also compare with the previous results reported in Rieck et al. (2008). OCAS achieves a speedup of factor 10 over SVM$^{\text{multi-class}}$ and of factor 20 compared to the one-against-all based classifier trained by kernel SVM$^{\text{light}}$ used in Rieck et al. (2008).

We adopted the evaluation protocol from Rieck et al. (2008). The classification accuracy is measured in terms of average per-class classification error, that is, $E = \frac{1}{Y}\sum_{y\in\mathcal{Y}} \frac{1}{m_i}\sum_{i|y_i=y}[\![y_i \neq h(\mathbf{x}_i)]\!]$, where m_i is the number of examples in the i-th class. The malware corpus of 10,072 examples is randomly split 5 times into training, validation and testing partitions of approximately equal size. The training partition is used to train the multi-class SVM using a range of different regularization constants $C \in \{10^0, 10^1, \ldots, 10^{10}\}$. The best C is selected based on classification accuracy measured on the validation part. Finally, we report average and standard deviations of the per-class average classification error computed on testing data over the 5 random splits.

Due to the very high-dimensional feature space, the explicit representation of the input vectors is inefficient. To apply the linear SVM solvers, we use the standard trick of representing the kernel matrix by the whitened empirical kernel map (Schölkopf and Smola, 2002). This representation reduces the number of features to the number of training examples. The runtime required by the singular value decomposition (SVD) to compute the whitened matrix is approximately 5 minutes, which is negligible w.r.t. the runtime of the SVM solvers. Note that training linear SVMs on the whitened kernel map is equivalent to training the kernel SVM classifier.

The experiments are performed on a laptop computer with an Intel CPU @ 1.8 GHz. Table 10 summarizes the results. The classification performance of SVM$^{\text{multi-class}}$ and OCAM is almost identical. The performance of both methods is slightly better than the results reported by Rieck et al. (2008), who used a heuristic one-against-all decomposition combined with the logistic regression. Comparison of the runtimes shows that the proposed OCAM is more than 10 times faster than the competing SVM$^{\text{multi-class}}$ and more than 20 times faster than the SVM$^{\text{light}}$ solver used in Rieck et al. (2008).

6. Conclusions

We have developed a novel method for solving large-scale risk minimization problems. Our proposed optimized cutting plane algorithm (OCA) extends the standard CPA algorithm of Teo et al. (2007) by, first, optimizing directly the primal problem via a line-search and, second, developing a new cutting plane selection strategy which significantly reduces the number of cutting planes needed to achieve an accurate solution. We have shown that the number of iterations OCA requires to converge to a ε-precise solution is approximately linear in the sample size. Applying OCA to two important learning problems, we obtained very fast specialized solvers for linear binary SVM classification (OCAS), and linear multi-class SVM classification (OCAM). In an extensive empirical evaluation on a large variety of problems comparing the proposed OCA with previous state-of-the-art SVM solvers, we achieved (depending on the task) speedups of up to three orders of magnitude obtaining the same precise SV solution. By parallelizing the subtasks of the algorithm, OCAS gained additional speedups of factors of up to 4.6 on a multi-core multiprocessor machine. Applying OCAS to a real-world splice site detection problem, we were able to train on a 12-million dimensional data set containing 50 million examples, achieving a new record performance for that task. Finally, we could reduce the training time on a malware classification problem by a factor of 20 over the previous approach. It remains as future work to derive OCAS for general structured output learning problems. Furthermore, we plan to extend OCAS to incorporate a bias term. Finally, it will be future work to investigate how the kernel framework can be incorporated into OCAS.

Acknowledgments

This work was supported in part by the FP7-ICT Programme of the European Union under the PASCAL2 Network of Excellence (ICT-216886) and by the Learning and Interference Platform of the Max Planck and Fraunhofer Societies. The main part of this work was done while VF was with the Fraunhofer IDA.FIRST when he was supported by a Marie Curie Intra-European Fellowship Grant SCOLES (MEIF-CT-2006-042107). VF was also supported by Czech Ministry of Education project 1M0567 during his current fellowship in the Center for Machine Perception. We thank Alexander Zien, Gunnar Rätsch, Konrad Rieck and Gilles Blanchard for great discussions. We also thank Konrad Rieck for providing us with the malware corpus.

Appendix A. Convergence Analysis

In this section we prove the convergence of OCA (Theorem 2). The core of the proof is adopted from Teo et al. (2007) who proved the Convergence Theorem 1 of the standard CPA. The main idea of the convergence theorem is based on deriving a lower bound on the improvement of the duality gap $F(\mathbf{w}_t^b) - F_t(\mathbf{w}_t) = \varepsilon_t$ and expressing this lower bound as a difference inequality (20), defined in Theorem 3. Having the difference inequality, (20) the proof of the convergence Theorem 2 follows easily.

The most laborious part is thus proving the auxiliary Theorem 3. The lower bound on the improvement $\varepsilon_t - \varepsilon_{t+1}$, which is the core inequality (20) in Theorem 3, is proven by showing that the objective value of the reduced problem solved at iteration $t+1$ must increase, provided the new added cutting plane violates the constraints of the reduced problem at iteration t. In the standard CPA, it is trivial to show that the new added cutting plane violates these constraints by at least ε_t.

Due to the sophisticated cutting plane selection strategy used in OCA, the violation of constraints is not obvious. Nevertheless, it can be proven, as we show in Lemma 1. Lemma 1 constitutes the main difference in the proofs of the standard CPA and the proposed OCA. The same lemma also explains why OCA converges faster than CPA. It will be shown that the minimal improvement of the reduced problem objective is a function of the constraint violation. While in the standard CPA the violation is guaranteed to be ε_t, the inequality (21) shows that the new cutting plane added in OCA violates the constraints by $\varepsilon_t + \frac{C}{2}\|\mathbf{w}_t^c - \mathbf{w}_t\|^2$. Unfortunately, we do not know how to bound the second term and thus the resulting lower bounds are the same for both algorithms.

The rest of this section is organized as follows. We first derive Lemmas 1, 2 and then we prove the auxiliary Theorem 3. Finally, we give the proof of the Convergence Theorem 2, which uses all previous results.

Theorem 3 *Assume that $\|\partial R(\mathbf{w})\| \leq G$ for all $\mathbf{w} \in \mathcal{W}$, where $\mathcal{W}$ is some domain of interest containing all $\mathbf{w}_{t'}$ for $t' \leq t$, and that $F(\mathbf{w}_t^b) - F_t(\mathbf{w}_t) = \varepsilon_t > 0$. In this case*

$$\varepsilon_t - \varepsilon_{t+1} \geq \frac{\varepsilon_t}{2} \min\left\{1, \frac{\varepsilon_t}{4C^2G^2}\right\}. \tag{20}$$

Lemma 1 *Let $F(\mathbf{w}_t^b) - F_t(\mathbf{w}_t) = \varepsilon_t > 0$. Then Algorithm 2 computes a new cutting plane $\langle \mathbf{w}, \mathbf{a}_{t+1}\rangle + b_{t+1} = 0$ which violates the constraints of the reduced problem (5) solved in the t-th iteration by at least $\frac{\varepsilon_t}{C}$, that is, it holds that*

$$C\big(b_{t+1} + \langle \mathbf{w}_t, \mathbf{a}_{t+1}\rangle - \xi_t\big) \geq \varepsilon_t + \frac{C}{2}\|\mathbf{w}_t^c - \mathbf{w}_t\|^2 \geq \varepsilon_t\,. \tag{21}$$

PROOF : We use the subgradient $\mathbf{w}_t^c + C\mathbf{a}_{t+1} \in \partial F(\mathbf{w}_t^c)$ to put a lower bound on the master objective F by means of a linear function at the point $\mathbf{w}_t^c$, that is,

$$f(\mathbf{w}) = F(\mathbf{w}_t^c) + \langle \mathbf{w}_t^c + C\mathbf{a}_{t+1}, \mathbf{w} - \mathbf{w}_t^c\rangle \leq F(\mathbf{w})\,, \qquad \forall \mathbf{w} \in \Re^n\,. \tag{22}$$

In Step 4 of Algorithm 2, the new best-so-far solution, $\mathbf{w}_t^b$, is computed as the minimizer of F over the line connecting the old best-so-far solution, $\mathbf{w}_{t-1}^b$, and the solution of reduced problem $\mathbf{w}_t$. In step 5, the new cutting plane $\langle \mathbf{w}, \mathbf{a}_{t+1}\rangle + b_{t+1} = 0$ is taken at the point $\mathbf{w}_t^c = \mathbf{w}_t^b(1-\mu) + \mathbf{w}_t\mu$, $\mu \in (0,1]$. Hence we conclude that $F(\mathbf{w}_t^b) \leq F(\mathbf{w}_t^c)$. Using the latter inequality and the lower bound (22), we obtain

$$f(\mathbf{w}_t^b) \leq F(\mathbf{w}_t^b) \leq F(\mathbf{w}_t^c) = f(\mathbf{w}_t^c)\,.$$

Since $\mathbf{w}_t^c$ lies on the line segment connecting $\mathbf{w}_t^b$ with $\mathbf{w}_t$ and because $f(\mathbf{w}_t^b) \leq f(\mathbf{w}_t^c)$ we conclude that

$$f(\mathbf{w}_t^c) \leq f(\mathbf{w}_t)\,. \tag{23}$$

Note that the inequality (23) holds only for $\mu \in (0,1]$. Using (22) we can rewrite (23) as

$$F(\mathbf{w}_t^c) \leq F(\mathbf{w}_t^c) + \langle \mathbf{w}_t^c + C\mathbf{a}_{t+1}, \mathbf{w}_t - \mathbf{w}_t^c\rangle\,. \tag{24}$$

Combining $F(\mathbf{w}_t^b) \leq F(\mathbf{w}_t^c)$ and $F(\mathbf{w}_t^b) - F_t(\mathbf{w}_t) = \varepsilon_t$ we get $F(\mathbf{w}_t^c) - F_t(\mathbf{w}_t) \geq \varepsilon_t$. Finally, substituting (24) to the latter inequality yields

$$F(\mathbf{w}_t^c) + \langle \mathbf{w}_t^c + C\mathbf{a}_{t+1}, \mathbf{w}_t - \mathbf{w}_t^c\rangle - F_t(\mathbf{w}_t) \geq \varepsilon_t\,,$$

which can be further rearranged into (21). ■

Lemma 2 *(Teo et al., 2007) Let $\Delta(\tau) = l\tau - \frac{h}{2}\tau^2$ be a concave quadratic function such that $\Delta'(0) \geq L > 0$ and $|\Delta''(\tau)| = h \leq H$, $\forall \tau \in [0,1]$. Then the maximal value of Δ attained for the interval $[0,1]$ has a lower bound defined by*

$$\max_{\tau \in [0,1]} \Delta(\tau) \geq \frac{L}{2} \min\left(1, \frac{L}{H}\right).$$

PROOF : Using $l \geq L$ and $h \leq H$ we obtain a lower bound $\Delta(\tau)$ by $L\tau - \frac{H}{2}\tau^2$. The unconstrained maximum of the lower bound is attained at point $\frac{L}{H}$, which leads to the value of $\frac{L^2}{2H}$. If $\frac{L}{H} > 1$ then the constrained maximum of the lower bound is attained at 1, which yields the maximal value of $L - \frac{H}{2}$. Using $\frac{L}{H} > 1$, the value of $L - \frac{H}{2}$ has a lower bound of $\frac{L}{2}$. Taking the minimum over both maxima proves the claim. ■

PROOF OF THEOREM 3: We can put a lower bound on the difference $\varepsilon_t - \varepsilon_{t+1}$ using the improvement of the dual objective function $D_{t+1}(\alpha_{t+1}) - D_t(\alpha_t)$ because

$$\begin{aligned} \varepsilon_t - \varepsilon_{t+1} &= F(\mathbf{w}_t^b) - F_t(\mathbf{w}_t) - F(\mathbf{w}_{t+1}^b) + F_{t+1}(\mathbf{w}_{t+1}) \\ &\geq F_{t+1}(\mathbf{w}_{t+1}) - F_t(\mathbf{w}_t) \\ &= D_{t+1}(\alpha_{t+1}) - D_t(\alpha_t). \end{aligned}$$

The inequality follows after excluding the term $F(\mathbf{w}_t^b) - F(\mathbf{w}_{t+1}^b) \geq 0$ and the last equality is the result of the fact that the primal and dual optimal values are equal.

The value of $D_{t+1}(\alpha_{t+1})$ is defined as the maximum of D_{t+1} w.r.t. the convex feasible set $\mathcal{A}_{t+1}$. Hence, by maximizing D_{t+1} w.r.t. a line segment lying entirely inside $\mathcal{A}_{t+1}$ we get a lower bound on $D_{t+1}(\alpha_{t+1})$, that is,

$$D_{t+1}(\alpha_{t+1}) - D_t(\alpha_t) \geq \max_{\tau \in [0,1]} \Delta(\tau) := D_{t+1}\big(\beta(1-\tau) + \gamma\tau\big) - D_t(\alpha_t),$$

where β and γ are arbitrary vectors from $\mathcal{A}_{t+1}$. Specifically, we define the vectors as

$$\beta = (\alpha_t; 0) \in \Re^{t+1} \quad \text{and} \quad \gamma = (\mathbf{0}; C) \in \Re^{t+1}. \tag{25}$$

Now we show that $\Delta(\tau)$ is a function compliant with the assumptions of Lemma 2, which will allow us to lower bound its value for the interval $[0,1]$. To this end, we need to derive the explicit form of $\Delta(\tau)$ and then compute $\Delta'(0)$ and an upper bound on $\Delta''(\tau)$, $\forall \tau \in [0,1]$. Defining a vector $\mathbf{b} = (b_1; \ldots; b_{t+1}) \in \Re^{t+1}$ and a matrix $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_{t+1}) \in \Re^{n \times (t+1)}$ we can write the objective of the dual of the reduced problem as $D_{t+1}(\alpha) = \langle \alpha, \mathbf{b} \rangle - \frac{1}{2}\|\mathbf{A}\alpha\|^2$. Using the latter definition of D_{t+1} and (25), we can rewrite $\Delta(\tau)$ as

$$\Delta(\tau) = \tau\big\langle \gamma - \beta, \mathbf{b} - \mathbf{A}^T\mathbf{A}\beta \big\rangle - \frac{1}{2}\tau^2 \big\|\mathbf{A}\beta - \mathbf{A}\gamma\big\|^2.$$

The value of the derivative $\Delta'(0)$ can be written as

$$\Delta'(0) = \big\langle \gamma - \beta, \mathbf{b} - \mathbf{A}^T\mathbf{A}\beta \big\rangle = C\big(b_{t+1} + \langle \mathbf{w}_t, \mathbf{a}_{t+1} \rangle - \xi_t\big). \tag{26}$$

The second equality of (26) was derived by using (25), $\mathbf{w}_t = -\sum_{i=1}^t \mathbf{a}_i [\alpha_t]_i$ and $F_t(\mathbf{w}_t) = D_t(\alpha_t)$. Using Lemma 1, we get a lower bound of the right-hand side of (26), that is

$$\Delta'(0) \geq \varepsilon_t. \tag{27}$$

The absolute value of the second derivative $|\Delta''(\tau)|$ can be upper bound by

$$|\Delta''(\tau)| = \|\mathbf{A}\beta - \mathbf{A}\gamma\|^2 \leq \|\mathbf{A}\beta\|^2 + \|\mathbf{A}\gamma\|^2 \leq 4C^2 \max_{i=1,\ldots,t+1} \|\mathbf{a}_i\|^2 \leq 4C^2G^2\,, \tag{28}$$

where we use the assumption $\mathbf{a}_i = \|\partial F(\mathbf{w})\| \leq G$ and the fact that the vector $\frac{1}{C}\mathbf{A}\alpha$ equals the convex combination of the columns of $\mathbf{A}$ for any $\alpha \in \mathcal{A}_{t+1}$, hence, its norm cannot be greater than $\max_i \|\mathbf{a}_i\|$. Finally, using (27) and (28) in Lemma 2 yields the claim of Theorem 3. ■

PROOF OF THEOREM 2: The proof is adopted from Teo et al. (2007). For any $\varepsilon_t > 4C^2G^2$ it follows from (20) that $\varepsilon_{t+1} \leq \frac{\varepsilon_t}{2}$. Moreover, $\varepsilon_0 \leq F(\mathbf{0})$, since F is nonnegative. Hence, we need at most $\log_2 \frac{F(\mathbf{0})}{4C^2G^2}$ iterations to achieve a level of precision better than $4C^2G^2$. Subsequently, we need to solve the following difference equation:

$$\varepsilon_{t+1} - \varepsilon_t = -\frac{\varepsilon_t^2}{8C^2G^2}\,.$$

Since this is monotonically decreasing, we can upper bound this by solving the differential equation $\varepsilon'(t) = -\frac{\varepsilon^2(t)}{8C^2G^2}$, with the boundary condition $\varepsilon(0) = 4C^2G^2$. This in turn yields $\varepsilon(t) = \frac{8C^2G^2}{t+2}$, and hence $t \leq \frac{8C^2G^2}{\varepsilon} - 2$ to achieve $\varepsilon(t) \leq \varepsilon$. For a given ε we will need $\frac{8C^2G^2}{\varepsilon} - 2$ more iterations to converge. This proves the claim.

■

References

A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In *Proceedings of International Machine Learning Conference (ICML)*, pages 89 – 96. OmniPress, 2007.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 20, pages 161 – 168. MIT Press, 2007.

C.C. Chang and C.J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

O. Chapelle. Training a Support Vector Machine in the Primal. *Neural Computation*, 19(5):1155–1178, 2007.

M. Collins, R.E. Schapire, and Y. Singer. Logistic regression AdaBoost and Bregman distance. In *Proceedings of Annual Conference on Computational Learning Theory (COLT)*, pages 158–169. Morgan Kaufman, San Francisco, 2000.

C. Cortes and V.N. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge UP, Cambridge, UK, 2000.

S. Degroeve, Y. Saeys, P. De Baets, B. Rouzé, and Y. Van de Peer. SpliceMachine: predicting splice sites from high-dimensional local context representations. *Bioinformatics*, 21(8):1332–8, 2005.

R. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

R.E. Fan, P.H. Chen, and C.J. Lin. Working set selection using second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

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

V. Franc. *Optimization Algorithms for Kernel Methods*. PhD thesis, Czech Technical University in Prague, July 2005. Supervised by V. Hlaváč.

V. Franc and S. Sonnenburg. OCAS optimized cutting plane algorithm for support vector machines. In *Proceedings of International Machine Learning Conference (ICML)*, pages 320–327. ACM Press, 2008a.

V. Franc and S. Sonnenburg. LIBOCAS, 2008b. Software available at `http://mloss.org/software/view/85/`.

V. Franc, P. Laskov, and K.-R. Müller. Stopping conditions for exact computation of leave-one-out error in support vector machines. In *Proceedings of International Machine Learning Conference (ICML)*, pages 328–335. ACM Press, 2008.

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

T. Joachims. A support vector method for multivariate performance measures. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 377 – 384. ACM New York, NY, USA, 2005.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 217 – 226. ACM New York, NY, USA, 2006.

T. Joachims, T. Finley, and C.N. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 76(1), May 2009.

C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of Pacific Biocomputing Symposium (PBS)*, pages 564–575. River Edge, NJ, World Scientific, 2002.

C.J. Lin, R.C. Weng, and S.S. Keerthi. Trust region Newton methods for large-scale logistic regression. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 561 – 568. ACM Press New York, 2007.

G. Rätsch and S. Sonnenburg. Accurate splice site detection for *Caenorhabditis elegans*. In K. Tsuda B. Schölkopf and J.-P. Vert, editors, *Kernel Methods in Computational Biology*. MIT Press, 2004.

G. Rätsch, S. S. Sonnenburg, and B. Schölkopf. RASE: recognition of alternatively spliced exons in *C. elegans*. *Bioinformatics*, 21(Suppl. 1):i369–i377, June 2005.

K. Rieck, T. Holtz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behaviour. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Fifth International Conference*, pages 108–125, July 2008.

B. Schölkopf and A. Smola. *Learning with Kernels*. The MIT Press, MA, 2002.

B. Schölkopf, J. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report TR 87, Microsoft Research, Redmond, WA, 1999.

S.S. Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 807 – 814. ACM Press, 2007.

V. Sindhwani and S.S. Keerthi. Newton methods for fast solution of semi-supervised linear svms. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007.

S. Sonnenburg. New methods for splice site recognition. Master's thesis, Humboldt University, 2002. supervised by K.-R. Müller H.-D. Burkhard and G.Rätsch.

S. Sonnenburg. *Machine Learning for Genomic Sequence Analysis*. PhD thesis, Fraunhofer Institute FIRST, 2008. supervised by K.-R. Müller and G.Rätsch.

S. Sonnenburg and G. Rätsch. Shogun, 2007. Software available at `http://mloss.org/software/view/2/`.

S. Sonnenburg, G. Rätsch, and K. Rieck. Large scale learning with string kernels. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007a.

S. Sonnenburg, G. Schweikert, P. Philips, J. Behr, and G. Rätsch. Accurate Splice Site Prediction. *BMC Bioinformatics, Special Issue from NIPS workshop on New Problems and Methods in Computational Biology Whistler, Canada, 18 December 2006*, 8:(Suppl. 10):S7, December 2007b.

S. Sonnenburg, V. Franc, E. Yomtov, and M. Sebag. The pascal large scale learning challenge. *Journal of Machine Learning Research*, 2009. (manuscript in preparation).

C.H. Teo, Q. Le, A. Smola, and S.V.N. Vishwanathan. A scalable modular convex solver for regularized risk minimization. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, August 2007.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, Sep. 2005.

Wikipedia. DDR2 SDRAM — Wikipedia, the free encyclopedia, 2009. URL `{http://en.wikipedia.org/wiki/DDR2_SDRAM}`. [Online; accessed 5-February-2009].

C.K.I Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In *Learning and Inference in Graphical Models*, pages 599–621. Kluwer Academic, 1998.

L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training. *Journal of Machine Learning Research*, 7:1467–1492, July 2006.