An Exact Solver for the Weston-Watkins SVM Subproblem

Yutong Wang [1] **Clayton Scott** [1 2]

Abstract

Recent empirical evidence suggests that the Weston-Watkins support vector machine is among the best performing multiclass extensions of the binary SVM. Current state-of-the-art solvers repeatedly solve a particular subproblem approximately using an iterative strategy. In this work, we propose an algorithm that solves the subproblem exactly using a novel reparametrization of the Weston-Watkins dual problem. For linear WW-SVMs, our solver shows significant speed-up over the state-of-the-art solver when the number of classes is large. Our exact subproblem solver also allows us to prove linear convergence of the overall solver.

1. Introduction

Support vector machines (SVMs) (Boser et al., 1992; Cortes & Vapnik, 1995) are a powerful class of algorithms for classification. In the large scale studies by Fernández-Delgado et al. (2014) and by Klambauer et al. (2017), SVMs are shown to be among the best performing classifers.

The original formulation of the SVM handles only binary classification. Subsequently, several variants of multiclass SVMs have been proposed (Lee et al., 2004; Crammer & Singer, 2001; Weston & Watkins, 1999). However, as pointed out by Doğan et al. (2016), no variant has been considered canonical.

The empirical study of Doğan et al. (2016) compared nine prominent variants of multiclass SVMs and demonstrated that the Weston-Watkins (WW) and Crammer-Singer (CS) SVMs performed the best with the WW-SVM holding a slight edge in terms of both efficiency and accuracy. This work focuses on the computational issues of solving the WW-SVM optimization efficiently.

SVMs are typically formulated as quadratic programs. State-of-the-art solvers such as LIBSVM (Chang & Lin, 2011) and LIBLINEAR (Fan et al., 2008) apply block coordinate descent to the associated dual problem, which entails repeatedly solving many small subproblems. For the binary case, these subproblems are easy to solve exactly.

The situation in the multiclass case is more complex, where the form of the subproblem depends on the variant of the multiclass SVM. For the CS-SVM, the subproblem can be solved exactly in $O(k \log k)$ time where k is the number of classes (Crammer & Singer, 2001; Duchi et al., 2008; Blondel et al., 2014; Condat, 2016). However, for the WW-SVM, only iterative algorithms that approximate the subproblem minimizer have been proposed, and these lack runtime guarantees (Keerthi et al., 2008; Igel et al., 2008).

In this work, we propose an algorithm called *Walrus*[1] that finds the exact solution of the Weston-Watkins subproblem in $O(k \log k)$ time. We implement Walrus in C++ inside the LIBLINEAR framework, yielding a new solver for the *linear* WW-SVM. For datasets with large number of classes, we demonstrate significant speed-up over the state-of-the-art linear solver Shark (Igel et al., 2008). We also rigorously prove the linear convergence of block coordinate descent for solving the dual problem of linear WW-SVM, confirming an assertion of Keerthi et al. (2008).

1.1. Related works

Existing literature on solving the optimization from SVMs largely fall into two categories: linear and kernel SVM solvers. The seminal work of Platt (1998) introduced the sequential minimal optimization (SMO) for solving kernel SVMs. Subsequently, many SMO-type algorithms were introduced which achieve faster convergence with theoretical guarantees (Keerthi et al., 2001; Fan et al., 2005; Steinwart et al., 2011; Torres-Barrán et al., 2021).

[1]Department of Electrical Engineering and Computer Science, University of Michigan. [2]Department of Statistics, University of Michigan. Correspondence to: Yutong Wang <yutongw@umich.edu>, Clayton Scott <clayscot@umich.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]WW-subproblem analytic log-linear runtime solver

SMO can be thought of as a form of *(block) coordinate descent* where where the dual problem of the SVM optimization is decomposed into small subproblems. As such, SMO-type algorithms are also referred to as *decomposition methods*. For binary SVMs, the smallest subproblems are 1-dimensional and thus easy to solve exactly. However, for multiclass SVMs with k classes, the smallest subproblems are k-dimensional. Obtaining exact solutions for the subproblems is nontrivial.

Many works have studied the convergence properties of decomposition focusing on asymptotics (List & Simon, 2004), rates (Chen et al., 2006; List & Simon, 2009), binary SVM without offsets (Steinwart et al., 2011), and multiclass SVMs (Hsu & Lin, 2002). Another line of research focuses on primal convergence instead of the dual (Hush et al., 2006; List & Simon, 2007; List et al., 2007; Beck et al., 2018).

Although kernel SVMs include linear SVMs as a special case, solvers specialized for linear SVMs can scale to larger data sets. Thus, linear SVM solvers are often developed separately. Hsieh et al. (2008) proposed using coordinate descent (CD) to solve the linear SVM dual problem and established linear convergence. Analogously, Keerthi et al. (2008) proposed block coordinate descent (BCD) for multiclass SVMs. Coordinate descent on the dual problem is now used by the current state-of-the-art linear SVM solvers LIBLINEAR (Fan et al., 2008), liquidSVM (Steinwart & Thomann, 2017), and Shark (Igel et al., 2008).

There are other approaches to solving linear SVMs, e.g., using the cutting plane method (Joachims, 2006), and stochastic subgradient descent on the primal optimization (Shalev-Shwartz et al., 2011). However, these approaches do not converge as fast as CD on the dual problem (Hsieh et al., 2008).

For the CS-SVM introduced by Crammer & Singer (2001), an exact solver for the subproblem is well-known and there is a line of research on improving the solver's efficiency (Crammer & Singer, 2001; Duchi et al., 2008; Blondel et al., 2014; Condat, 2016). For solving the kernel CS-SVM dual problem, convergence of an SMO-type algorithm was proven in (Lin, 2002). For solving the linear CS-SVM dual problem, linear convergence of coordinate descent was proven by Lee & Chang (2019). Linear CS-SVMs with ℓ_1-regularizer have been studied by Babichev et al. (2019)

The Weston-Watkins SVM was introduced by Bredensteiner & Bennett (1999); Weston & Watkins (1999); Vapnik (1998). Empirical results from Doğan et al. (2016) suggest that the WW-SVM is the best performing multiclass SVMs among nine prominent variants. The WW-SVM loss function has also been successfully used in natural language processing by Schick & Schütze (2020).

Hsu & Lin (2002) gave an SMO-type algorithm for solving the WW-SVM, although without convergence guarantees. Keerthi et al. (2008) proposed using coordinate descent on the linear WW-SVM dual problem with an iterative subproblem solver. Furthermore, they asserted that the algorithm converges linearly, although no proof was given. The software Shark (Igel et al., 2008) features a solver for the linear WW-SVM where the subproblem is approximately minimized by a greedy coordinate descent-type algorithm. MSVMpack (Didiot & Lauer, 2015) is a solver for multiclass SVMs which uses the Frank-Wolfe algorithm. The experiments of (van den Burg & Groenen, 2016) showed that MSVMpack did not scale to larger number of classes for the WW-SVM. To our knowledge, an exact solver for the subproblem has not previously been developed.

1.2. Notations

Let n be a positive integer. Define $[n] := \{1, \dots, n\}$. All vectors are assumed to be column vectors unless stated otherwise. If $v \in \mathbb{R}^n$ is a vector and $i \in [n]$, we use the notation $[v]_i$ to denote the i-th component of v. Let $\mathbb{1}_n$ and $\mathbb{0}_n \in \mathbb{R}^n$ denote the vectors of all ones and zeros, respectively. When the dimension n can be inferred from the context, we drop the subscript and simply write $\mathbb{1}$ and $\mathbb{0}$.

Let m be a positive integer. Matrices $\mathbf{w} \in \mathbb{R}^{m \times n}$ are denoted by boldface font. The (j, i)-th entry of $\mathbf{w}$ is denoted by w_{ji}. The columns of $\mathbf{w}$ are denoted by the same symbol $w_1, \dots, w_n$ using regular font with a single subscript, i.e., $[w_i]_j = w_{ji}$. A column of $\mathbf{w}$ is sometimes referred to as a *block*. We will also use boldface Greek letter to denote matrices, e.g., $\boldsymbol{\alpha} \in \mathbb{R}^{m \times n}$ with columns $\alpha_1, \dots, \alpha_n$.

The 2-norm of a vector v is denoted by $\|v\|$. The Frobenius norm of a matrix $\mathbf{w}$ is denoted by $\|\mathbf{w}\|_F$. The $m \times m$ identity and all-ones matrices are denoted by $\mathbf{I}_m$ and $\mathbf{O}_m$, respectively. When m is clear from the context, we drop the subscript and simply write $\mathbf{I}$ and $\mathbf{O}$.

For referencing, section numbers from our supplementary materials will be prefixed with an "A", e.g., Section A.5.

2. Weston-Watkins linear SVM

Throughout this work, let $k \geq 2$ be an integer denoting the number of classes. Let $\{(x_i, y_i)\}_{i \in [n]}$ be a training dataset of size n where the instances $x_i \in \mathbb{R}^d$ and labels $y_i \in [k]$.

The Weston-Watkins linear SVM [2] solves the optimization

$$\min_{\mathbf{w}\in\mathbb{R}^{d\times k}} \frac{1}{2}\|\mathbf{w}\|_F^2 + C\sum_{i=1}^{n}\sum_{\substack{j\in[k]:\\ j\neq y_i}} \texttt{hinge}(w'_{y_i}x_i - w'_j x_i) \quad \text{(P)}$$

where $\texttt{hinge}(t) = \max\{0, 1-t\}$ and $C > 0$ is a hyperparameter.

Note that if an instance x_i is the zero vector, then for any $\mathbf{w} \in \mathbb{R}^{d\times k}$ we have $\texttt{hinge}(w'_{y_i}x_i - w'_j x_i) = 1$. Thus, we can simply ignore such an instance. Below, we assume that $\|x_i\| > 0$ for all $i \in [n]$.

2.1. Dual of the linear SVM

In this section, we recall the dual of (P). Derivation of all results here can be found in Hsu & Lin (2002); Keerthi et al. (2008).

We begin by defining the function $f : \mathbb{R}^{k\times n} \to \mathbb{R}$

$$f(\boldsymbol{\alpha}) := \frac{1}{2}\sum_{i,s\in[n]} x'_s x_i \alpha'_i \alpha_s - \sum_{i\in[k]}\sum_{\substack{j\in[k]:\\ j\neq y_i}} \alpha_{ij}$$

and the set

$$\mathcal{F} := \Big\{\boldsymbol{\alpha}\in\mathbb{R}^{k\times n} \mid 0 \le \alpha_{ij} \le C,\ \forall i\in[n],\ j\in[k],\ j\neq y_i,\ \alpha_{iy_i} = -\sum_{j\in[k]\setminus\{y_i\}} \alpha_{ij},\ \forall i\in[n]\Big\}.$$

The dual problem of (P) is

$$\min_{\boldsymbol{\alpha}\in\mathcal{F}} f(\boldsymbol{\alpha}). \quad \text{(D1)}$$

The primal and dual variables $\mathbf{w}$ and $\boldsymbol{\alpha}$ are related via

$$\mathbf{w} = -\sum_{i\in[n]} x_i \alpha'_i. \quad (1)$$

State-of-the-art solver Shark (Igel et al., 2008) uses coordinate descent on the dual problem (D1). It is also possible to solve the primal problem (P) using stochastic gradient descent (SGD) as in Pegasos (Shalev-Shwartz et al., 2011). However, the empirical results of Hsieh et al. (2008) show that CD on the dual problem converges faster than SGD on the primal problem. Hence, we focus on the dual problem.

2.2. Solving the dual with block coordinate descent

Block coordinate descent (BCD) is an iterative algorithm for solving the dual problem (D1) by repeatedly improving a candidate solution $\boldsymbol{\alpha} \in \mathcal{F}$. Given an $i \in [n]$, an *inner iteration* performs the update $\boldsymbol{\alpha} \leftarrow \widetilde{\boldsymbol{\alpha}}$ where $\widetilde{\boldsymbol{\alpha}}$ is a minimizer of the *i-th subproblem*:

$$\min_{\widehat{\boldsymbol{\alpha}}\in\mathcal{F}} f(\widehat{\boldsymbol{\alpha}}) \text{ such that } \widehat{\alpha}_s = \alpha_s,\ \forall s \in [n]\setminus\{i\}. \quad \text{(S1)}$$

An *outer iteration* performs the inner iteration once for each $i \in [n]$ possibly in a random order. By running several outer iterations, an (approximate) minimizer of (D1) is putatively obtained.

Later, we will see that it is useful to keep track of $\mathbf{w}$ so that (1) holds throughout the BCD algorithm. Suppose that $\boldsymbol{\alpha}$ and $\mathbf{w}$ satisfy (1). Then $\mathbf{w}$ must be updated via

$$\mathbf{w} \leftarrow \mathbf{w} - x_i(\widetilde{\alpha}_i - \alpha_i)' \quad (2)$$

prior to updating $\boldsymbol{\alpha} \leftarrow \widetilde{\boldsymbol{\alpha}}$.

3. Reparametrization of the dual problem

In this section, we introduce a new way to parametrize the dual optimization (D1) which allows us to derive an algorithm for finding the exact minimizer of (S1).

Define the matrix $\boldsymbol{\pi} := \begin{bmatrix}\mathbb{1} & -\mathbf{I}\end{bmatrix} \in \mathbb{R}^{(k-1)\times k}$. For each $y \in [k]$, let $\boldsymbol{\sigma}_y \in \mathbb{R}^{k\times k}$ be the permutation matrix which switches the 1st and the yth indices. In other words, given a vector $v \in \mathbb{R}^k$, we have

$$[\boldsymbol{\sigma}_y(v)]_j = \begin{cases} v_1 & : j = y \\ v_y & : j = 1 \\ v_j & : j \notin \{1, y\}. \end{cases}$$

Define the function $g : \mathbb{R}^{(k-1)\times n} \to \mathbb{R}$

$$g(\boldsymbol{\beta}) := \frac{1}{2}\sum_{i,s\in[n]} x'_s x_i \beta'_i \boldsymbol{\pi}\boldsymbol{\sigma}_{y_i}\boldsymbol{\sigma}_{y_s}\boldsymbol{\pi}'\beta_s - \sum_{i\in[n]} \mathbb{1}'\beta_i$$

and the set

$$\mathcal{G} := \Big\{\boldsymbol{\beta}\in\mathbb{R}^{(k-1)\times n} \mid 0 \le \beta_{ij} \le C,\ \forall i\in[n],\ j\in[k-1]\Big\}.$$

Consider the following optimization:

$$\min_{\boldsymbol{\beta}\in\mathcal{G}} g(\boldsymbol{\beta}). \quad \text{(D2)}$$

Up to a change of variables, the optimization (D2) is equivalent to the dual of the linear WW-SVM (D1). In other words, (D2) is a reparametrization of (D1). Below, we make this notion precise.

Definition 3.1. Define a map $\Psi : \mathcal{G} \to \mathbb{R}^{k\times n}$ as follows: Given $\boldsymbol{\beta} \in \mathcal{G}$, construct an element $\Psi(\boldsymbol{\beta}) := \boldsymbol{\alpha} \in \mathbb{R}^{k\times n}$ whose i-th block is

$$\alpha_i = -\boldsymbol{\sigma}_{y_i}\boldsymbol{\pi}'\beta_i. \quad (3)$$

[2] Similar to other works on multiclass linear SVMs (Hsu & Lin, 2002; Keerthi et al., 2008), the formulation (P) does not use *offsets*. For discussions, see Section A.1.

The map Ψ will serve as the change of variables map, where $\boldsymbol{\pi}$ reduces the dual variable's dimension from k for α_i to $k-1$ for β_i. Furthermore, $\boldsymbol{\sigma}_{y_i}$ eliminates the dependency on y_i in the constraints. The following proposition shows that Ψ links the two optimization problems (D1) and (D2).

Proposition 3.2. *The image of Ψ is $\mathcal{F}$, i.e., $\Psi(\mathcal{G}) = \mathcal{F}$. Furthermore, $\Psi : \mathcal{G} \to \mathcal{F}$ is a bijection and*

$$f(\Psi(\boldsymbol{\beta})) = g(\boldsymbol{\beta}).$$

Sketch of proof. Define another map $\Xi : \mathcal{F} \to \mathbb{R}^{(k-1)\times n}$ as follows: For each $\boldsymbol{\alpha} \in \mathcal{F}$, define $\boldsymbol{\beta} := \Xi(\boldsymbol{\alpha})$ block-wise by

$$\beta_i := \mathtt{proj}_{2:k}(\boldsymbol{\sigma}_{y_i}\alpha_i) \in \mathbb{R}^{k-1}$$

where

$$\mathtt{proj}_{2:k} = \begin{bmatrix}\mathbb{0} & \mathbf{I}_{k-1}\end{bmatrix} \in \mathbb{R}^{(k-1)\times k}.$$

Then the range of Ξ is in $\mathcal{G}$. Furthermore, Ξ and Ψ are inverses of each other. This proves that Ψ is a bijection. □

3.1. Reparametrized subproblem

Since the map Ψ respects the block-structure of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the result below follows immediately from Proposition 3.2:

Corollary 3.3. *Let $\boldsymbol{\beta} \in \mathcal{G}$ and $i \in [n]$. Let $\boldsymbol{\alpha} = \Psi(\boldsymbol{\beta})$. Consider*

$$\min_{\widehat{\boldsymbol{\beta}} \in \mathcal{G}} \; g(\widehat{\boldsymbol{\beta}}) \text{ such that } \widehat{\beta}_s = \beta_s, \forall s \in [n] \setminus \{i\}. \qquad \text{(S2)}$$

Let $\widetilde{\boldsymbol{\beta}} \in \mathcal{F}$ be arbitrary. Then $\widetilde{\boldsymbol{\beta}}$ is a minimizer of (S2) *if and only if $\widetilde{\boldsymbol{\alpha}} := \Psi(\widetilde{\boldsymbol{\beta}})$ is a minimizer of* (S1).

Below, we focus on solving (D2) with BCD, i.e., repeatedly performing the update $\boldsymbol{\beta} \leftarrow \widetilde{\boldsymbol{\beta}}$ where $\widetilde{\boldsymbol{\beta}}$ is a minimizer of (S2) over different $i \in [n]$. By Corollary 3.3, this is equivalent to solving (D1) with BCD, up to the change of variables Ψ.

The reason we focus on solving (D2) with BCD is because the subproblem can be cast in a simple form that makes an exact solver more apparent. To this end, we first show that the subproblem (S2) is a quadratic program of a particular form. Define the matrix $\boldsymbol{\Theta} := \mathbf{I}_{k-1} + \mathbf{O}_{k-1}$.

Theorem 3.4. *Let $v \in \mathbb{R}^{k-1}$ be arbitrary and $C > 0$. Consider the optimization*

$$\begin{aligned} \min_{b \in \mathbb{R}^{k-1}} \quad & \frac{1}{2} b'\boldsymbol{\Theta} b - v'b \\ s.t. \quad & 0 \le b \le C. \end{aligned} \qquad (4)$$

Then Algorithm 2, `solve_subproblem`(v, C)*, computes the unique minimizer of* (4) *in $O(k \log k)$ time.*

We defer further discussion of Theorem 3.4 and Algorithm 2 to the next section. The quadratic program (4) is the *generic* form of the subproblem (S2), as the following result shows:

Proposition 3.5. *In the situation of Corollary 3.3, let $\widetilde{\beta}_i$ be the i-th block of the minimizer $\widetilde{\boldsymbol{\beta}}$ of* (S2)*. Then $\widetilde{\beta}_i$ is the unique minimizer of* (4) *with*

$$v := (\mathbb{1} - \boldsymbol{\pi}\boldsymbol{\sigma}_{y_i}\mathbf{w}'x_i)/\|x_i\|_2^2 + \boldsymbol{\Theta}\beta_i$$

and $\mathbf{w}$ *as in* (1).

3.2. BCD for the reparametrized dual problem

As mentioned in Section 2.2, it is useful to keep track of $\mathbf{w}$ so that (1) holds throughout the BCD algorithm. In Proposition 3.5, we see that $\mathbf{w}$ is used to compute v. The update formula (2) for $\mathbf{w}$ in terms of $\widetilde{\boldsymbol{\alpha}}$ can be cast in terms of $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$ by using (3):

$$\mathbf{w} \leftarrow \mathbf{w} - x_i(\widetilde{\alpha}_i - \alpha_i)' = \mathbf{w} + x_i(\widetilde{\beta}_i - \beta_i)'\boldsymbol{\pi}\boldsymbol{\sigma}_{y_i}.$$

We now have all the ingredients to state the reparametrized block coordinate descent pseudocode in Algorithm 1.

Algorithm 1 Block coordinate descent on (D2)

$\boldsymbol{\beta} \leftarrow \mathbf{0}_{(k-1)\times n}$
$\mathbf{w} \leftarrow \mathbf{0}_{d\times k}$
while not converged **do**
  for $i \leftarrow 1$ **to** n **do**
    $v \leftarrow (\mathbb{1} - \boldsymbol{\pi}\boldsymbol{\sigma}_{y_i}\mathbf{w}'x_i)/\|x_i\|_2^2 + \boldsymbol{\Theta}\beta_i$
    $\widetilde{\beta}_i \leftarrow$ `solve_subproblem`(v, C) (Algorithm 2)
    $\mathbf{w} \leftarrow \mathbf{w} + x_i(\widetilde{\beta}_i - \beta_i)'\boldsymbol{\pi}\boldsymbol{\sigma}_{y_i}$
    $\beta_i \leftarrow \widetilde{\beta}_i$
  end for
end while

Multiplying a vector by the matrices $\boldsymbol{\Theta}$ and $\boldsymbol{\pi}$ both only takes $O(k)$ time. Multiplying a vector by $\boldsymbol{\sigma}_{y_i}$ takes $O(1)$ time since $\boldsymbol{\sigma}_{t_i}$ simply swaps two entries of the vector. Hence, the speed bottlenecks of Algorithm 1 are computing $\mathbf{w}'x_i$ and $x_i(\widetilde{\beta}_i - \beta_i)'$, both taking $O(dk)$ time and running `solve_subproblem`(v, C), which takes $O(k \log k)$ time. Overall, a single inner iteration of Algorithm 1 takes $O(dk + k \log k)$ time. If x_i is s-sparse (only s entries are nonzero), then the iteration takes $O(sk + k \log k)$ time.

3.3. Linear convergence

Similar to the binary case (Hsieh et al., 2008), BCD converges *linearly*, i.e., it produces an ϵ-accurate solution in $O(\log(1/\epsilon))$ outer iterations:

Theorem 3.6. *Algorithm 1 has global linear convergence. More precisely, let $\boldsymbol{\beta}^t$ be $\boldsymbol{\beta}$ at the end of the t-th iteration*

of the outer loop of Algorithm 1. Let $g^* = \min_{\boldsymbol{\beta}\in\mathcal{G}} g(\boldsymbol{\beta})$. *Then there exists* $\Delta \in (0,1)$ *such that*

$$g(\boldsymbol{\beta}^{t+1}) - g^* \leq \Delta(g(\boldsymbol{\beta}^t) - g^*), \qquad \forall t = 0, 1, 2 \dots \quad (5)$$

where Δ *depends on the data* $\{(x_i, y_i)\}_{i\in[n]}$, k *and* C.

Luo & Tseng (1992) proved asymptotic[3] linear convergence for cyclic coordinate descent for a certain class of minimization problems where the subproblem in each coordinate is *exactly* minimized. Furthermore, Luo & Tseng (1992) claim that the same result holds if the subproblem is *approximately* minimized, but did not give a precise statement (e.g., approximation in which sense).

Keerthi et al. (2008) asserted without proof that the results of Luo & Tseng (1992) can be applied to BCD for WW-SVM. Possibly, no proof was given since no solver, exact nor approximate with approximation guarantees, was known at the time. Theorem 3.6 settles this issue, which we prove in Section A.4 by extending the analysis of Luo & Tseng (1992); Wang & Lin (2014) to the multiclass case.

4. Sketch of proof of Theorem 3.4

Throughout this section, let $v \in \mathbb{R}^{k-1}$ and $C > 0$ be fixed. We first note that (4) is a minimization of a strictly convex function over a compact domain, and hence has unique minimizer $\widetilde{b} \in \mathbb{R}^{k-1}$. Furthermore, it is the unique point satisfying the KKT conditions, which we present below. Our goal is to sketch the argument that Algorithm 2 outputs the minimizer upon termination. The full proof can be found in Section A.5.

4.1. Intuition

We first study the structure of the minimizer $\widetilde{b}$ in and of itself. The KKT conditions for a point $b \in \mathbb{R}^{k-1}$ to be optimal for (4) are as follows:

$$\begin{aligned} \forall i \in [k-1], \exists \lambda_i, \mu_i \in \mathbb{R} &\text{ satisfying} \\ [(\mathbf{I}+\mathbf{O})b]_i + \lambda_i - \mu_i = v_i &\quad \text{stationarity} \qquad \text{(KKT)} \\ C \geq b_i \geq 0 &\quad \text{primal feasibility} \\ \lambda_i \geq 0, \text{ and } \mu_i \geq 0 &\quad \text{dual feasibility} \\ \lambda_i(C - b_i) = 0, \text{ and } \mu_i b_i = 0 &\quad \text{complementary slackness} \end{aligned}$$

Below, let $\max_{i\in[k-1]} v_i =: v_{\max}$, and $\langle 1\rangle, \dots, \langle k-1\rangle$ be an argsort of v, i.e., $v_{\langle 1\rangle} \geq \cdots \geq v_{\langle k-1\rangle}$.

Definition 4.1. The *clipping map* $\texttt{clip}_C : \mathbb{R}^{k-1} \to [0, C]^{k-1}$ is the function defined as follows: for $w \in \mathbb{R}^{k-1}$, $[\texttt{clip}_C(w)]_i := \max\{0, \min\{C, w_i\}\}$.

Using the KKT conditions, we check that $\widetilde{b} = \texttt{clip}_C(v - \widetilde{\gamma}\mathbb{1})$ for some (unknown) $\widetilde{\gamma} \in \mathbb{R}$ and that $\widetilde{\gamma} = \mathbb{1}'\widetilde{b}$.

[3] Asymptotic in the sense that (5) is only guaranteed after $t > t_0$ for some unknown t_0.

Proof. Let $\widetilde{\gamma} \in \mathbb{R}$ be such that $\mathbf{O}\widetilde{b} = \widetilde{\gamma}\mathbb{1}$. The stationarity condition can be rewritten as $\widetilde{b}_i + \lambda_i - \mu_i = v_i - \widetilde{\gamma}$. Thus, by complementary slackness and dual feasibility, we have

$$\widetilde{b}_i \begin{cases} \leq v_i - \widetilde{\gamma} & : \widetilde{b}_i = C \\ = v_i - \widetilde{\gamma} & : \widetilde{b}_i \in (0, C) \\ \geq v_i - \widetilde{\gamma} & : \widetilde{b}_i = 0 \end{cases}$$

Note that this is precisely $\widetilde{b} = \texttt{clip}_C(v - \widetilde{\gamma}\mathbb{1})$. □

For $\gamma \in \mathbb{R}$, let $b^\gamma := \texttt{clip}_C(v - \gamma\mathbb{1}) \in \mathbb{R}^{k-1}$. Thus, the $(k-1)$-dimensional vector $\widetilde{b}$ can be recovered from the scalar $\widetilde{\gamma}$ via $b^{\widetilde{\gamma}}$, reducing the search space from $\mathbb{R}^{k-1}$ to $\mathbb{R}$.

However, the search space $\mathbb{R}$ is still a continuum. We show that the search space for $\widetilde{\gamma}$ can be further reduced to a finite set of candidates. To this end, let us define

$$\begin{aligned} I_{\mathrm{u}}^\gamma &:= \{i \in [k-1] : b_i^\gamma = C\} \\ I_{\mathrm{m}}^\gamma &:= \{i \in [k-1] : b_i^\gamma \in (0, C)\}. \end{aligned}$$

Note that I_{u}^γ and I_{m}^γ are determined by their cardinalities, denoted n_{u}^γ and n_{m}^γ, respectively. This is because

$$\begin{aligned} I_{\mathrm{u}}^\gamma &= \{\langle 1\rangle, \langle 2\rangle, \dots, \langle n_{\mathrm{u}}^\gamma\rangle\} \\ I_{\mathrm{m}}^\gamma &= \{\langle n_{\mathrm{u}}^\gamma + 1\rangle, \langle n_{\mathrm{u}}^\gamma + 2\rangle, \dots, \langle n_{\mathrm{u}}^\gamma + n_{\mathrm{m}}^\gamma\rangle\}. \end{aligned}$$

Let $[\![k]\!] := \{0\} \cup [k-1]$. By definition, $n_{\mathrm{m}}^\gamma, n_{\mathrm{u}}^\gamma \in [\![k]\!]$. For $(n_{\mathrm{m}}, n_{\mathrm{u}}) \in [\![k]\!]^2$, define $S^{(n_{\mathrm{m}}, n_{\mathrm{u}})}, \widehat{\gamma}^{(n_{\mathrm{m}}, n_{\mathrm{u}})} \in \mathbb{R}$ by

$$S^{(n_{\mathrm{m}}, n_{\mathrm{u}})} := \sum_{i=n_{\mathrm{u}}+1}^{n_{\mathrm{u}}+n_{\mathrm{m}}} v_{\langle i\rangle}, \quad (6)$$

$$\widehat{\gamma}^{(n_{\mathrm{m}}, n_{\mathrm{u}})} := \left(C \cdot n_{\mathrm{u}} + S^{(n_{\mathrm{m}}, n_{\mathrm{u}})}\right)/(n_{\mathrm{m}} + 1). \quad (7)$$

Furthermore, define $\widehat{b}^{(n_{\mathrm{m}}, n_{\mathrm{u}})} \in \mathbb{R}^{k-1}$ such that, for $i \in [k-1]$, the $\langle i\rangle$-th entry is

$$\widehat{b}_{\langle i\rangle}^{(n_{\mathrm{m}}, n_{\mathrm{u}})} := \begin{cases} C & : i \leq n_{\mathrm{u}} \\ v_{\langle i\rangle} - \gamma^{(n_{\mathrm{m}}, n_{\mathrm{u}})} & : n_{\mathrm{u}} < i \leq n_{\mathrm{u}} + n_{\mathrm{m}} \\ 0 & : n_{\mathrm{u}} + n_{\mathrm{m}} < i. \end{cases}$$

Using the KKT conditions, we check that

$$\widetilde{b} = \widehat{b}^{(n_{\mathrm{m}}^{\widetilde{\gamma}}, n_{\mathrm{u}}^{\widetilde{\gamma}})} = \texttt{clip}_C(v - \widehat{\gamma}^{(n_{\mathrm{m}}^{\widetilde{\gamma}}, n_{\mathrm{u}}^{\widetilde{\gamma}})}\mathbb{1}).$$

Proof. It suffices to prove that $\widetilde{\gamma} = \widehat{\gamma}^{(n_{\mathrm{m}}^{\widetilde{\gamma}}, n_{\mathrm{u}}^{\widetilde{\gamma}})}$. To this end, let $i \in [k-1]$. If $i \in I_{\mathrm{m}}^{\widetilde{\gamma}}$, then $\widetilde{b}_i = v_i - \widetilde{\gamma}$. If $i \in I_{\mathrm{u}}^{\widetilde{\gamma}}$, then $\widetilde{b}_i = C$. Otherwise, $\widetilde{b}_i = 0$. Thus

$$\widetilde{\gamma} = \mathbb{1}'\widetilde{b} = C \cdot n_{\mathrm{u}}^{\widetilde{\gamma}} + S^{(n_{\mathrm{m}}^{\widetilde{\gamma}}, n_{\mathrm{u}}^{\widetilde{\gamma}})} - \widetilde{\gamma} \cdot n_{\mathrm{m}}^{\widetilde{\gamma}} \quad (8)$$

Solving for $\widetilde{\gamma}$, we have

$$\widetilde{\gamma} = \left(C \cdot n_{\mathrm{u}}^{\widetilde{\gamma}} + S^{(n_{\mathrm{m}}^{\widetilde{\gamma}}, n_{\mathrm{u}}^{\widetilde{\gamma}})}\right)/(n_{\mathrm{m}}^{\widetilde{\gamma}} + 1) = \widehat{\gamma}^{(n_{\mathrm{m}}^{\widetilde{\gamma}}, n_{\mathrm{u}}^{\widetilde{\gamma}})},$$

as desired. □

Now, since $(n_{\mathtt{m}}^{\widetilde{\gamma}}, n_{\mathtt{u}}^{\widetilde{\gamma}}) \in \llbracket k \rrbracket^2$, to find $\widetilde{b}$ we can simply check for each $(n_{\mathtt{m}}, n_{\mathtt{u}}) \in \llbracket k \rrbracket^2$ if $\widehat{b}^{(n_{\mathtt{m}}, n_{\mathtt{u}})}$ satisfies the KKT conditions. However, this naive approach leads to an $O(k^2)$ runtime.

To improve upon the naive approach, define

$$\Re := \{(n_{\mathtt{m}}^{\gamma}, n_{\mathtt{u}}^{\gamma}) : \gamma \in \mathbb{R}\}. \tag{9}$$

Since $(n_{\mathtt{m}}^{\widetilde{\gamma}}, n_{\mathtt{u}}^{\widetilde{\gamma}}) \in \Re$, to find $\widetilde{b}$ it suffices to search through $(n_{\mathtt{m}}, n_{\mathtt{u}}) \in \Re$ instead of $\llbracket k \rrbracket^2$. Towards enumerating all elements of $\Re$, a key result is that the function $\gamma \mapsto (I_{\mathtt{m}}^{\gamma}, I_{\mathtt{u}}^{\gamma})$ is locally constant outside of the set of discontinuities:

$$\texttt{disc} := \{v_i : i \in [k-1]\} \cup \{v_i - C : i \in [k-1]\}.$$

Proof. Let $\gamma_1, \gamma_2, \gamma_3, \gamma_4 \in \mathbb{R}$ satisfy the following: 1) $\gamma_1 < \gamma_2 < \gamma_3 < \gamma_4$, 2) $\gamma_1, \gamma_4 \in \texttt{disc}$, and 3) $\gamma \notin \texttt{disc}$ for all $\gamma \in (\gamma_1, \gamma_4)$. Assume for the sake of contradiction that $(I_{\mathtt{m}}^{\gamma_2}, I_{\mathtt{u}}^{\gamma_2}) \neq (I_{\mathtt{m}}^{\gamma_3}, I_{\mathtt{u}}^{\gamma_3})$. Then $I_{\mathtt{m}}^{\gamma_2} \neq I_{\mathtt{m}}^{\gamma_3}$ or $I_{\mathtt{u}}^{\gamma_2} \neq I_{\mathtt{u}}^{\gamma_3}$. Consider the case $I_{\mathtt{m}}^{\gamma_2} \neq I_{\mathtt{m}}^{\gamma_3}$. Then at least one of the sets $I_{\mathtt{m}}^{\gamma_2} \setminus I_{\mathtt{m}}^{\gamma_3}$ and $I_{\mathtt{m}}^{\gamma_3} \setminus I_{\mathtt{m}}^{\gamma_2}$ is nonempty. Consider the case when $I_{\mathtt{m}}^{\gamma_2} \setminus I_{\mathtt{m}}^{\gamma_3}$ is nonempty. Then there exists $i \in [k-1]$ such that $v_i - \gamma_2 \in (0, C)$ but $v_i - \gamma_3 \notin (0, C)$. This implies that there exists some $\gamma' \in (\gamma_2, \gamma_3)$ such that $v_i - \gamma' \in \{0, C\}$, or equivalently, $\gamma' \in \{v_i, v_i - C\}$. Hence, $\gamma' \in \texttt{disc}$, which is a contradiction. For the other cases not considered, similar arguments lead to the same contradiction. □

Thus, as we sweep γ from $+\infty$ to $-\infty$, we observe finitely many distinct tuples of sets $(I_{\mathtt{m}}^{\gamma}, I_{\mathtt{u}}^{\gamma})$ and their cardinalities $(n_{\mathtt{m}}^{\gamma}, n_{\mathtt{u}}^{\gamma})$. Using the index $t = 0, 1, 2 \ldots$, we keep track of these data in the variables $(I_{\mathtt{m}}^{t}, I_{\mathtt{u}}^{t})$ and $(n_{\mathtt{m}}^{t}, n_{\mathtt{u}}^{t})$. For this proof sketch, we make the assumption that $|\texttt{disc}| = 2(k-1)$, i.e., no elements are repeated.

By construction, the maximal element of `disc` is $v_{\max}$. When $\gamma > v_{\max}$, we check that $n_{\mathtt{m}}^{\gamma} = n_{\mathtt{u}}^{\gamma} = \emptyset$. Thus, we put $I_{\mathtt{m}}^{0} = I_{\mathtt{u}}^{0} = \emptyset$ and $(n_{\mathtt{m}}^{0}, n_{\mathtt{u}}^{0}) = (0, 0)$.

Now, suppose γ has swept across $t-1$ points of discontinuity and that $I_{\mathtt{m}}^{t-1}, I_{\mathtt{u}}^{t-1}, n_{\mathtt{m}}^{t-1}, n_{\mathtt{u}}^{t-1}$ have all been defined. Suppose that γ crossed a single new point of discontinuity $\gamma' \in \texttt{disc}$. In other words, $\gamma'' < \gamma < \gamma'$ where γ'' is the largest element of `disc` such that $\gamma'' < \gamma'$.

By the assumption that no elements of `disc` are repeated, exactly one of the two following possibilities is true:

$$\text{there exists } i \in [k-1] \text{ such that } \gamma' = v_i, \tag{Entry}$$
$$\text{there exists } i \in [k-1] \text{ such that } \gamma' = v_i - C. \tag{Exit}$$

Under the (Entry) case, the index i gets added to $I_{\mathtt{m}}^{t-1}$ while $I_{\mathtt{u}}^{t-1}$ remains unchanged. Hence, we have the updates

$$I_{\mathtt{m}}^{t} := I_{\mathtt{m}}^{\gamma} = I_{\mathtt{m}}^{t-1} \cup \{i\}, \quad I_{\mathtt{u}}^{t} := I_{\mathtt{u}}^{\gamma} = I_{\mathtt{u}}^{t-1} \tag{10}$$
$$n_{\mathtt{m}}^{t} := n_{\mathtt{m}}^{\gamma} = n_{\mathtt{m}}^{t-1} + 1, \quad n_{\mathtt{u}}^{t} := n_{\mathtt{u}}^{\gamma} = n_{\mathtt{u}}^{t-1}. \tag{11}$$

Under the (Exit) case, the index i moves from $I_{\mathtt{m}}^{t-1}$ to $I_{\mathtt{u}}^{t-1}$. Hence, we have the updates

$$I_{\mathtt{m}}^{t} := I_{\mathtt{m}}^{\gamma} = I_{\mathtt{m}}^{t-1} \setminus \{i\}, \quad I_{\mathtt{u}}^{t} := I_{\mathtt{u}}^{\gamma} = I_{\mathtt{u}}^{t-1} \cup \{i\} \tag{12}$$
$$n_{\mathtt{m}}^{t} := n_{\mathtt{m}}^{\gamma} = n_{\mathtt{m}}^{t-1} - 1, \quad n_{\mathtt{u}}^{t} := n_{\mathtt{u}}^{\gamma} = n_{\mathtt{u}}^{t-1} + 1. \tag{13}$$

Thus, $\{(n_{\mathtt{m}}^{t}, n_{\mathtt{u}}^{t})\}_{t=0}^{2(k-1)} = \Re$. The case when `disc` has repeated elements requires more careful analysis which is done in the full proof. Now, we have all the ingredients for understanding Algorithm 2 and its subroutines.

4.2. A walk through of the solver

If $v_{\max} \leq 0$, then $\widetilde{b} = \mathbb{0}$ satisfies the KKT conditions. Algorithm 2-line 3 handles this exceptional case. Below, we assume $v_{\max} > 0$.

Algorithm 2 `solve_subproblem`(v, C)

1: **Input:** $v \in \mathbb{R}^{k-1}$
2: Let $\langle 1 \rangle, \ldots, \langle k-1 \rangle$ sort v, i.e., $v_{\langle 1 \rangle} \geq \cdots \geq v_{\langle k-1 \rangle}$.
3: **if** $v_{\langle 1 \rangle} \leq 0$ **then HALT and output:** $\mathbb{0} \in \mathbb{R}^{k-1}$.
4: $n_{\mathtt{u}}^{0} := 0$, $n_{\mathtt{m}}^{0} := 0$, $S^0 := 0$
5: $(\delta_1, \ldots, \delta_\ell) \leftarrow$ `get_up_dn_seq()` (Subroutine 3)
6: **for** $t = 1, \ldots, \ell$ **do**
7: $\quad (n_{\mathtt{m}}^{t}, n_{\mathtt{u}}^{t}, S^t) \leftarrow$ `update_vars()` (Subroutine 4).
8: $\quad \widehat{\gamma}^t := (C \cdot n_{\mathtt{u}}^{t} + S^t)/(n_{\mathtt{m}}^{t} + 1)$
9: $\quad$ **if** `KKT_cond()` (Subroutine 5) returns true **then**
10: $\quad\quad$ **HALT and output**: $\widehat{b}^t \in \mathbb{R}^{k-1}$ where

$$\widehat{b}^t_{\langle i \rangle} := \begin{cases} C & : i \leq n_{\mathtt{u}}^{t} \\ v_{\langle i \rangle} - \gamma^t & : n_{\mathtt{u}}^{t} < i \leq n_{\mathtt{u}}^{t} + n_{\mathtt{m}}^{t} \\ 0 & : n_{\mathtt{u}}^{t} + n_{\mathtt{m}}^{t} < i. \end{cases}$$

11: $\quad$ **end if**
12: **end for**

Algorithm 2-line 4 initializes the state variables $n_{\mathtt{m}}^{t}$ and $n_{\mathtt{u}}^{t}$ as discussed in the last section. The variable S^t is also initialized and will be updated to maintain $S^t = S^{(n_{\mathtt{m}}^{t}, n_{\mathtt{u}}^{t})}$ where the latter is defined at (6).

Algorithm 2-line 5 calls Subroutine 3 to construct the `vals` ordered set, which is similar to the set of discontinuities `disc`, but different in three ways: 1) `vals` consists of tuples (γ', δ') where $\gamma' \in \texttt{disc}$ and $\delta' \in \{\texttt{up}, \texttt{dn}\}$ is a decision variable indicating whether γ' satisfies the (Entry) or the (Exit) condition, 2) `vals` is sorted so that the γ's are in descending order, and 3) only positive values of `disc` are needed. The justification for the third difference is because we prove that Algorithm 2 always halts before reaching the negative values of `disc`. Subroutine 3 returns the list of symbols $(\delta_1, \ldots, \delta_\ell)$ consistent with the ordering.

Subroutine 3 `get_up_dn_seq` *Note: all variables from Algorithm 2 are assumed to be visible here.*

1: `vals` $\leftarrow \{(v_i, \texttt{dn}) : v_i > 0, i = 1, \ldots, k-1\} \cup \{(v_i - C, \texttt{up}) : v_i > C, i = 1, \ldots, k-1\}$ as a multiset, where elements may be repeated.
2: Order the set $\texttt{vals} = \{(\gamma_1, \delta_1), \ldots, (\gamma_\ell, \delta_\ell)\}$ such that $\gamma_1 \geq \cdots \geq \gamma_\ell$, $\ell = |\texttt{vals}|$, and for all $j_1, j_2 \in [\ell]$ such that $j_1 < j_2$ and $\gamma_{j_1} = \gamma_{j_2}$, we have $\delta_{j_1} = \texttt{dn}$ implies $\delta_{j_2} = \texttt{dn}$.
Note that by construction, for each $t \in [\ell]$, there exists $i \in [k-1]$ such that $\gamma_t = v_i$ or $\gamma_t = v_i - C$.
3: **Output:** sequence $(\delta_1, \ldots, \delta_\ell)$ whose elements are retrieved in order from left to right.

In the "for" loop, Algorithm 2-line 7 calls Subroutine 4 which updates the variables $n_{\texttt{m}}^t, n_{\texttt{u}}^t$ using (11) or (13), depending on δ_t. The variable S^t is updated accordingly so that $S^t = S^{(n_{\texttt{m}}^t, n_{\texttt{u}}^t)}$.

Subroutine 4 `update_vars` *Note: all variables from Algorithm 2 are assumed to be visible here.*

1: **if** $\delta_t = \texttt{up}$ **then**
2: $\quad n_{\texttt{u}}^t := n_{\texttt{u}}^{t-1} + 1, \quad n_{\texttt{m}}^t := n_{\texttt{m}}^{t-1} - 1$
3: $\quad S^t := S^{t-1} - v_{\langle n_{\texttt{u}}^{t-1} \rangle}$
4: **else**
5: $\quad n_{\texttt{m}}^t := n_{\texttt{m}}^{t-1} + 1, \quad n_{\texttt{u}}^t := n_{\texttt{u}}^{t-1}.$
6: $\quad S^t := S^{t-1} + v_{\langle n_{\texttt{u}}^t + n_{\texttt{m}}^t \rangle}$
7: **end if**
8: **Output:** $(n_{\texttt{m}}^t,\ n_{\texttt{u}}^t,\ S^t)$

We skip to Algorithm 2-line 9 which constructs the putative solution $\widehat{b}^t$. Observe that $\widehat{b}^t = \widehat{b}^{(n_{\texttt{m}}^t, n_{\texttt{u}}^t)}$ where the latter is defined in the previous section.

Going back one line, Algorithm 2-line 8 calls Subroutine 5 which checks if the putative solution $\widehat{b}^t$ satisfies the KKT conditions. We note that this can be done *before* the putative solution is constructed.

For the runtime analysis, we note that Subroutines 5 and 4 both use $O(1)$ FLOPs without dependency on k. The main "for" loop of Algorithm 2 (line 6 through 11) has $O(\ell)$ runtime where $\ell \leq 2(k-1)$. Thus, the bottlenecks are Algorithm 2-line 2 and 5 which sort lists of length at most $k-1$ and $2(k-1)$, respectively. Thus, both lines run in $O(k \log k)$ time.

5. Experiments

LIBLINEAR is one of the state-of-the-art solver for linear SVMs (Fan et al., 2008). However, as of the latest version 2.42, the linear Weston-Watkins SVM is not supported. We implemented our linear WW-SVM subproblem solver, *Walrus* (Algorithm 2), along with the BCD Algorithm 1 as an extension to LIBLINEAR. The solver and code for generating the figures are available[4].

Subroutine 5 `KKT_cond` *Note: all variables from Algorithm 2 are assumed to be visible here.*

1: `kkt_cond` $\leftarrow true$
2: **if** $n_{\texttt{u}}^t > 0$ **then**
3: $\quad$ `kkt_cond` $\leftarrow$ `kkt_cond` $\wedge \left(C + \widehat{\gamma}^t \leq v_{\langle n_{\texttt{u}}^t \rangle}\right)$
$\quad$ *Note: $\wedge$ denotes the logical "and".*
4: **end if**
5: **if** $n_{\texttt{m}}^t > 0$ **then**
6: $\quad$ `kkt_cond` $\leftarrow$ `kkt_cond` $\wedge \left(v_{\langle n_{\texttt{u}}^t + 1 \rangle} \leq C + \widehat{\gamma}^t\right)$
7: $\quad$ `kkt_cond` $\leftarrow$ `kkt_cond` $\wedge \left(\widehat{\gamma}^t \leq v_{\langle n_{\texttt{u}}^t + n_{\texttt{m}}^t \rangle}\right)$
8: **end if**
9: **if** $n_{\texttt{o}}^t := k - 1 - n_{\texttt{u}}^t - n_{\texttt{m}}^t > 0$ **then**
10: $\quad$ `kkt_cond` $\leftarrow$ `kkt_cond` $\wedge \left(v_{\langle n_{\texttt{u}}^t + n_{\texttt{m}}^t + 1 \rangle} \leq \widehat{\gamma}^t\right)$
11: **end if**
12: **Output:** `kkt_cond`

Table 1. Data sets used. Variables k, n and d are, respectively, the number of classes, training samples, and features.

DATA SET	k	n	d
DNA	3	2,000	180
SATIMAGE	6	4,435	36
MNIST	10	60,000	780
NEWS20	20	15,935	62,061
LETTER	26	15,000	16
RCV1	53	15,564	47,236
SECTOR	105	6,412	55,197
ALOI	1,000	81,000	128

We compare our implementation to *Shark* (Igel et al., 2008), which solves the dual subproblem (S1) using a form of greedy coordinate descent. For comparisons, we reimplemented Shark's solver also as a LIBLINEAR extension. When clear from the context, we use the terms "Walrus" and "Shark" when referring to either the subproblem solver or the overall BCD algorithm.

We perform benchmark experiments on 8 datasets from "LIBSVM Data: Classification (Multi-class)[5]" spanning a range of k from 3 to 1000. See Table 1.

In all of our experiments, Walrus and Shark perform identically in terms of testing accuracy. We report the accuracies in Section A.6.3. Below, we will only discuss runtime.

For measuring the runtime, we start the timer after the data sets have been loaded into memory and before the state variables $\boldsymbol{\beta}$ and $\mathbf{w}$ have been allocated. The primal objective is the value of (P) at the current $\mathbf{w}$ and the dual

[4] See Section A.6.
[5] See Section A.6.2.

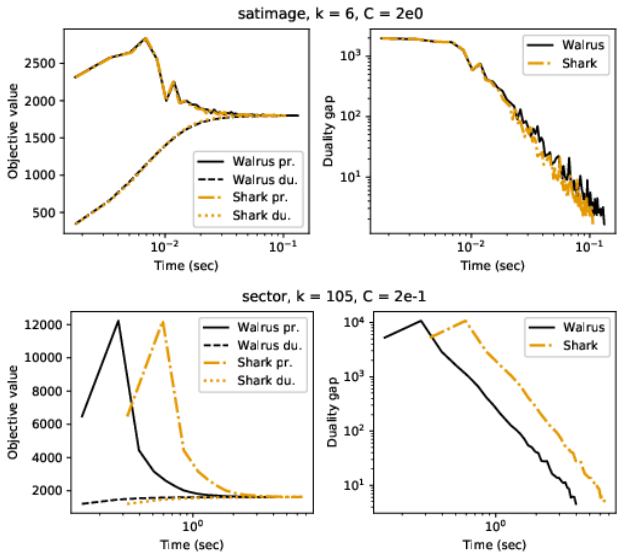

Figure 1. Runtime comparison of Walrus and Shark. Abbreviations: pr. = primal and du. = dual. The X-axes show time elapsed.

objective is -1 times the value of (D2) at the current $\boldsymbol{\beta}$. The duality gap is the primal minus the dual objective. The objective values and duality gaps are measured after each *outer* iteration, during which the timer is paused.

For solving the subproblem, Walrus is guaranteed to return the minimizer in $O(k \log k)$ time. On the other hand, to the best of our knowledge, Shark does not have such guarantee. Furthermore, Shark uses a doubly-nested for loop, each of which has length $O(k)$, yielding a worst-case runtime of $O(k^2)$. For these reasons, we hypothesize that Walrus scales better with larger k.

As exploratory analysis, we ran Walrus and Shark on the SATIMAGE and SECTOR data sets[6], which has 6 and 105 classes, respectively. The results, shown in Figure 1, support our hypothesis: Walrus and Shark are equally fast for SATIMAGE while Walrus is faster for SECTOR.

We test our hypothesis on a larger scale by running Walrus and Shark on the datasets in Table 1 over the grid of hyperparameters $C \in \{2^{-6}, 2^{-5}, \ldots, 2^2, 2^3\}$. The results are shown in Figure 2 where each dot represents a triplet $(\text{DATA SET}, C, \delta)$ where δ is a quantity we refer to as the *duality gap decay*. The Y-axis shows the comparative metric of runtime $\texttt{ET}^{\delta}_{\texttt{Walrus}}/\texttt{ET}^{\delta}_{\texttt{Shark}}$ to be defined next.

Consider a single run of Walrus on a fixed data set with a given hyperparameter C. Let $\texttt{DG}^{t}_{\texttt{Walrus}}$ denote the duality gap achieved by Walrus at the end of the t-th outer iteration. Let $\delta \in (0, 1)$. Define $\texttt{ET}^{\delta}_{\texttt{Walrus}}$ to be the elapsed time at the end of the t-th iteration where t is minimal such that $\texttt{DG}^{t}_{\texttt{Walrus}} \leq \delta \cdot \texttt{DG}^{1}_{\texttt{Walrus}}$. Define $\texttt{DG}^{t}_{\texttt{Shark}}$ and $\texttt{ET}^{\delta}_{\texttt{Shark}}$ similarly. In all experiments $\texttt{DG}^{1}_{\texttt{Walrus}}/\texttt{DG}^{1}_{\texttt{Shark}} \in [0.99999, 1.00001]$. Thus, the ratio $\texttt{ET}^{\delta}_{\texttt{Walrus}}/\texttt{ET}^{\delta}_{\texttt{Shark}}$ measures how much faster Shark is relative to Walrus.

From Figure 2, it is evident that in general Walrus converges faster on data sets with larger number of classes. Not only does Walrus beat Shark for large k, but it also seems to not do much worse for small k. In fact Walrus seems to be at least as fast as Shark for all datasets except SATIMAGE.

The absolute amount of time saved by Walrus is often more significant on datasets with larger number of classes. To illustrate this, we let $C = 1$ and compare the times for the duality gap to decay by a factor of 0.01. On the data set SATIMAGE with $k = 6$, Walrus and Shark take 0.0476 and 0.0408 seconds, respectively. On the data set ALOI with $k = 1000$, Walrus and Shark take 188 and 393 seconds, respectively.

We remark that Figure 2 also suggests that Walrus tends to be faster during early iterations but can be slower at late stages of the optimization. To explain this phenomenon, we note that Shark solves the subproblem using an iterative descent algorithm and is set to stop when the KKT violations fall below a hard-coded threshold. When close to optimality, Shark takes fewer descent steps, and hence less time, to reach the stopping condition on the subproblems. On the other hand, Walrus takes the same amount of time regardless of proximity to optimality.

For the purpose of grid search, a high degree of optimality is not needed. In Section A.6.3, we provide empirical evidence that stopping early versus late does not change the result of grid search-based hyperparameter tuning. Specifically, Table 7 shows that running the solvers until $\delta \approx 0.01$ or until $\delta \approx 0.001$ does not change the cross-validation outcomes.

Finally, the optimization (4) is a convex quadratic program and hence can be solved using general-purpose solvers (Voglis & Lagaris, 2004). However, we find that Walrus, being specifically tailored to the optimization (4), is orders of magnitude faster. See Tables 8 and 9 in the Appendix.

6. Discussions and future works

We presented an algorithm called Walrus for exactly solving the WW-subproblem which scales with the number of classes. We implemented Walrus in the LIBLINEAR framework and demonstrated empirically that BCD using Walrus is significantly faster than state-of-the-art linear WW-SVM solver Shark on datasets with a large number

[6]The regularizers are set to the corresponding values from Table 5 of the supplementary material of Doğan et al. (2016) chosen by cross-validation.

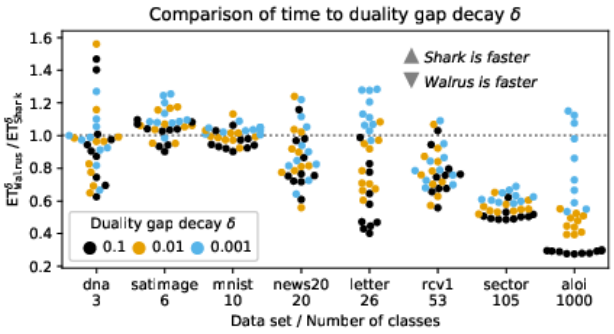

Figure 2. X-coordinates jittered for better visualization.

of classes, and comparable to Shark for small number of classes.

One possible direction for future research is whether Walrus can improve *kernel* WW-SVM solver. Another direction is lower-bounding time complexity of solving the WW-subproblem (4).

Acknowledgements

The authors were supported in part by the National Science Foundation under awards 1838179 and 2008074, by the Department of Defense, Defense Threat Reduction Agency under award HDTRA1-20-2-0002, and by the Michigan Institute for Data Science.

References

Babichev, D., Ostrovskii, D., and Bach, F. Efficient primal-dual algorithms for large-scale multiclass classification. *arXiv preprint arXiv:1902.03755*, 2019.

Beck, A., Pauwels, E., and Sabach, S. Primal and dual predicted decrease approximation methods. *Mathematical Programming*, 167(1):37–73, 2018.

Blondel, M., Fujino, A., and Ueda, N. Large-scale multiclass support vector machine training via Euclidean projection onto the simplex. In *2014 22nd International Conference on Pattern Recognition*, pp. 1289–1294. IEEE, 2014.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152, 1992.

Bredensteiner, E. J. and Bennett, K. P. Multicategory classification by support vector machines. In *Computational Optimization*, pp. 53–79. Springer, 1999.

Chang, C.-C. and Lin, C.-J. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

Chen, P.-H., Fan, R.-E., and Lin, C.-J. A study on SMO-type decomposition methods for support vector machines. *IEEE Trans. Neural Networks*, 17(4):893–908, 2006.

Chiu, C.-C., Lin, P.-Y., and Lin, C.-J. Two-variable dual coordinate descent methods for linear SVM with/without the bias term. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pp. 163–171. SIAM, 2020.

Condat, L. Fast projection onto the simplex and the l_1 ball. *Mathematical Programming*, 158(1-2):575–585, 2016.

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

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

Didiot, E. and Lauer, F. Efficient optimization of multiclass support vector machines with msvmpack. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pp. 23–34. Springer, 2015.

Doğan, U., Glasmachers, T., and Igel, C. A unified view on multi-class support vector classification. *The Journal of Machine Learning Research*, 17(1):1550–1831, 2016.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. Efficient projections onto the l_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 272–279, 2008.

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

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

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 408–415, 2008.

Hsu, C.-W. and Lin, C.-J. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

Hush, D., Kelly, P., Scovel, C., and Steinwart, I. QP algorithms with guaranteed accuracy and run time for support vector machines. *Journal of Machine Learning Research*, 7(May):733–769, 2006.

Igel, C., Heidrich-Meisner, V., and Glasmachers, T. Shark. *Journal of Machine Learning Research*, 9(Jun):993–996, 2008.

Joachims, T. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 217–226, 2006.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., and Murthy, K. R. K. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

Keerthi, S. S., Sundararajan, S., Chang, K.-W., Hsieh, C.-J., and Lin, C.-J. A sequential dual method for large scale multi-class linear SVMs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 408–416, 2008.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 971–980, 2017.

Lee, C.-P. and Chang, K.-W. Distributed block-diagonal approximation methods for regularized empirical risk minimization. *Machine Learning*, pp. 1–40, 2019.

Lee, Y., Lin, Y., and Wahba, G. Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99(465):67–81, 2004.

Lin, C.-J. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 13(5):1045–1052, 2002.

List, N. and Simon, H. U. A general convergence theorem for the decomposition method. In *International Conference on Computational Learning Theory*, pp. 363–377. Springer, 2004.

List, N. and Simon, H. U. General polynomial time decomposition algorithms. *Journal of Machine Learning Research*, 8(Feb):303–321, 2007.

List, N. and Simon, H. U. SVM-optimization and steepest-descent line search. In *Proceedings of the 22nd Annual Conference on Computational Learning Theory*, 2009.

List, N., Hush, D., Scovel, C., and Steinwart, I. Gaps in support vector optimization. In *International Conference on Computational Learning Theory*, pp. 336–348. Springer, 2007.

Luo, Z.-Q. and Tseng, P. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

Luo, Z.-Q. and Tseng, P. Error bounds and convergence analysis of feasible descent methods: a general approach. *Annals of Operations Research*, 46(1):157–178, 1993.

Platt, J. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, 1998.

Schick, T. and Schütze, H. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*, 2020.

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

Steinwart, I. and Thomann, P. liquidSVM: A fast and versatile SVM package. *arXiv preprint arXiv:1702.06899*, 2017.

Steinwart, I., Hush, D., and Scovel, C. Training SVMs without offset. *Journal of Machine Learning Research*, 12(1), 2011.

Torres-Barrán, A., Alaíz, C. M., and Dorronsoro, J. R. Faster SVM training via conjugate SMO. *Pattern Recognition*, 111:107644, 2021. ISSN 0031-3203.

van den Burg, G. and Groenen, P. Gensvm: A generalized multiclass support vector machine. *The Journal of Machine Learning Research*, 17(1):7964–8005, 2016.

Vapnik, V. Statistical learning theory, 1998.

Voglis, C. and Lagaris, I. E. Boxcqp: An algorithm for bound constrained convex quadratic problems. In *Proceedings of the 1st International Conference: From Scientific Computing to Computational Engineering, IC-SCCE, Athens, Greece*, 2004.

Wang, P.-W. and Lin, C.-J. Iteration complexity of feasible descent methods for convex optimization. *The Journal of Machine Learning Research*, 15(1):1523–1548, 2014.

Weston, J. and Watkins, C. Support vector machines for multi-class pattern recognition. In *Proc. 7th European Symposium on Artificial Neural Networks, 1999*, 1999.