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 classifiers.

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 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).

SMO can be thought of as a form of (block) coordinate descent where where the dual problem of the SVM opti-
mization 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 \( \ell_1 \)-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 \([n] := \{1, \ldots, n\}\). 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 \( 1_n \) and \( 0_n \in \mathbb{R}^n \) 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 \( 1 \) and \( 0 \).

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

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

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

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

\(^2\)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.
An candidate solution for solving the dual problem (D1) by repeatedly improving the primal problem. Hence, we focus on the dual problem. However, the empirical results of Hsieh et al. (2008) show that CD on the dual problem converges faster than SGD on the Westen-Watkins SVM subproblem. We begin by defining the function $f: \mathbb{R}^{k \times n} \rightarrow \mathbb{R}$

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

and the set

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

The dual problem of (P) is

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

The primal and dual variables $w$ and $\alpha$ are related via

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

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.

### 2.1. 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 $f: \mathbb{R}^{k \times n} \rightarrow \mathbb{R}$

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

and the set

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

The primal and dual variables $w$ and $\alpha$ are related via

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

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

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

prior to updating $\alpha \leftarrow \tilde{\alpha}$.

### 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 $\pi := [I - I] \in \mathbb{R}^{(k-1) \times k}$. For each $y \in [k]$, let $\sigma_y \in \mathbb{R}^{k \times k}$ be the permutation matrix which switches the 1st and the $y$th indices. In other words, given a vector $v \in \mathbb{R}^k$, we have

$$[\sigma_y(v)]_{ij} = \begin{cases} v_1 : j = y \\ v_y : j = 1 \\ v_j : j \notin \{1, y\}. \end{cases}$$

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

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

and the set

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

Consider the following optimization:

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

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 $\Psi: \mathcal{G} \rightarrow \mathbb{R}^{k \times n}$ as follows: Given $\beta \in \mathcal{G}$, construct an element $\Psi(\beta) := \alpha \in \mathbb{R}^{k \times n}$ whose $i$-th block is

$$\alpha_i = -\sigma y_i \pi^' \beta_i \quad (3)$$

The map $\Psi$ will serve as the change of variables map, where $\pi$ reduces the dual variable’s dimension from $k$ for $\alpha_i$ to $k - 1$ for $\beta_i$. Furthermore, $\sigma y_i$ eliminates the dependency on $y_i$ in the constraints. The following proposition shows that $\Psi$ links the two optimization problems (D1) and (D2).
3.1. Reparametrized subproblem

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

\[
f(\Psi(\beta)) = g(\beta).
\]

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

\[
\beta_i := \text{proj}_{2:k}(\sigma_y, \alpha_i) \in \mathbb{R}^{k-1}
\]

where

\[
\text{proj}_{2:k} = [0 \ 1_{k-1}] \in \mathbb{R}^{(k-1) \times k}.
\]

Then the range of \( \Xi \) is in \( \mathcal{G} \). Furthermore, \( \Xi \) and \( \Psi \) are inverses of each other. This proves that \( \Psi \) is a bijection. \( \Box \)

Consider the optimization \( \text{exact solver more apparent. To this end, we first show that} \)

\[ (S2) \]

The reason we focus on solving \( (D2) \) with BCD is because \( \Psi \) variables equivalent to solving \( (D1) \) with BCD, up to the change of

\[ \text{inverses of each other. This proves that} \]

\[ \beta \]

\[ \hat{\beta} \]

Then Algorithm 2, \text{solve subproblem} \((v, C)\) (Algorithm 2)

\[ \hat{\beta}_i \]

Multiplying a vector by the matrices \( \Theta \) and \( \pi \) both only takes \( O(k) \) time. Multiplying a vector by \( \sigma_y \) takes \( O(1) \) time since \( \sigma_i \) simply swaps two entries of the vector. Hence, the speed bottlenecks of Algorithm 1 are computing \( w^i x_i \) and \( x_i (\hat{\beta}_i - \beta_i)' \), both taking \( O(dk) \) time and running \text{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

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 \( \hat{\beta}_i \) be the \( i \)-th block of the minimizer \( \beta \) of \((S2)\). Then \( \hat{\beta}_i \) is the unique minimizer of (4) with \v

\[
\text{and} \ w \text{ as in (1).}
\]

3.2. BCD for the reparametrized dual problem

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

\[
w \leftarrow w - x_i (\tilde{\alpha}_i - \alpha_i)' = w + x_i (\tilde{\beta}_i - \beta_i)' \pi \sigma_y.
\]
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 \( \tilde{b} \in \mathbb{R}^{k-1} \). 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 \( \tilde{b} \) in and of itself. The KKT conditions for a point \( b \in \mathbb{R}^{k-1} \) to be optimal for (4) are as follows:

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

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

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

Using the KKT conditions, we check that \( \tilde{b} = \text{clip}_C(v - \bar{\gamma}1) \) for some (unknown) \( \bar{\gamma} \in \mathbb{R} \) and that \( \bar{\gamma} = 1/\tilde{b} \).

**Proof.** Let \( \bar{\gamma} \in \mathbb{R} \) be such that \( O\tilde{b} = \bar{\gamma}1 \). The stationarity condition can be rewritten as \( b_i + \lambda_i - \mu_i = v_i - \bar{\gamma} \). Thus, by complementary slackness and dual feasibility, we have

\[
\begin{align*}
\tilde{b}_i \begin{cases} 
\leq v_i - \bar{\gamma} & : b_i = C \\
v_i - \bar{\gamma} & : b_i \in (0, C) \\
\geq v_i - \bar{\gamma} & : b_i = 0
\end{cases}
\end{align*}
\]

Note that this is precisely \( \tilde{b} = \text{clip}_C(v - \bar{\gamma}1) \).

For \( \bar{\gamma} \in \mathbb{R} \), let \( b^\gamma := \text{clip}_C(v - \bar{\gamma}1) \in \mathbb{R}^{k-1} \). Thus, the \( (k-1) \)-dimensional vector \( b \) can be recovered from the scalar \( \bar{\gamma} \) via \( b^\gamma \), 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 \( \bar{\gamma} \) can be further reduced to a finite set of candidates. To this end, let us define

\[
\begin{align*}
I^\gamma_u & := \{i \in [k-1] : b^\gamma_i = C\} \\
I^\gamma_m & := \{i \in [k-1] : b^\gamma_i \in (0, C)\}
\end{align*}
\]

Note that \( I^\gamma_u \) and \( I^\gamma_m \) are determined by their cardinalities, denoted \( n^\gamma_u \) and \( n^\gamma_m \), respectively. This is because

\[
\begin{align*}
I^\gamma_u & = \{(1), \ldots, \langle n^\gamma_u \gamma \rangle\} \\
I^\gamma_m & = \{\langle n^\gamma_u + 1, \gamma \rangle, \langle n^\gamma_u + 2, \gamma \rangle, \ldots, \langle n^\gamma_m + n^\gamma_u, \gamma \rangle\}
\end{align*}
\]

Let \( \|k\| := \{0\} \cup \{k-1\} \). By definition, \( n^\gamma_u, n^\gamma_m \in \|k\| \).

For \( (n^\gamma_u, n^\gamma_m) \in \|k\|^2 \), define \( S(n^\gamma_u, n^\gamma_m) \subseteq \mathbb{R} \) by

\[
S(n^\gamma_u, n^\gamma_m) := \sum_{i=n^\gamma_u+1}^{n^\gamma_m+n^\gamma_u} v(i), \quad (6)
\]

\[
\bar{\gamma}(n^\gamma_u, n^\gamma_m) := \left( C \cdot n^\gamma_u + S(n^\gamma_u, n^\gamma_m) \right) / (n^\gamma_m + 1). \quad (7)
\]

Furthermore, define \( \hat{b}(n^\gamma_u, n^\gamma_m) \in \mathbb{R}^{k-1} \) such that, for \( i \in [k-1] \), the \( (i) \)-th entry is

\[
\hat{b}(n^\gamma_u, n^\gamma_m) := \begin{cases} 
C & : i \leq n^\gamma_u \\
\bar{\gamma}(n^\gamma_u, n^\gamma_m) & : n^\gamma_u < i \leq n^\gamma_u + n^\gamma_m \\
0 & : n^\gamma_u + n^\gamma_m < i.
\end{cases}
\]

Using the KKT conditions, we check that

\[
\tilde{b} = \hat{b}(n^\gamma_m, n^\gamma_u) = \text{clip}_C(v - \bar{\gamma}(n^\gamma_u, n^\gamma_m)1).
\]

**Proof.** It suffices to prove that \( \bar{\gamma} = \bar{\gamma}(n^\gamma_m, n^\gamma_u) \). To this end, let \( i \in [k-1] \). If \( i \in I^\gamma_u \), then \( \tilde{b}_i = v_i - \bar{\gamma} \). If \( i \in I^\gamma_m \), then \( \tilde{b}_i = C \). Otherwise, \( \tilde{b}_i = 0 \). Thus

\[
\bar{\gamma} = \bar{v}^T \tilde{b} = C \cdot n^\gamma_m + S(n^\gamma_m, n^\gamma_u) - \bar{\gamma} \cdot n^\gamma_m \quad (8)
\]

Solving for \( \bar{\gamma} \), we have

\[
\bar{\gamma} = \left( C \cdot n^\gamma_m + S(n^\gamma_m, n^\gamma_u) \right) / (n^\gamma_m + 1) = \bar{\gamma}(n^\gamma_u, n^\gamma_m),
\]

as desired.

Now, since \( (n^\gamma_m, n^\gamma_u) \in \|k\|^2 \), to find \( \tilde{b} \) we can simply check for each \( (n^\gamma_u, n^\gamma_m) \in \|k\|^2 \) if \( \hat{b}(n^\gamma_u, n^\gamma_m) \) satisfies the KKT conditions. However, this naive approach leads to an \( O(k^2) \) runtime.
To improve upon the naive approach, define
\( \mathcal{R} := \{(n^m_m, n^u_u) : \gamma \in \mathcal{R}\}. \) \hspace{1cm} (9)

Since \((n^m_m, n^u_u) \in \mathcal{R},\) to find \(\tilde{b}\) it suffices to search through \((n^m_m, n^u_u) \in \mathcal{R}\) instead of \(|k|^2\). Towards enumerating all elements of \(\mathcal{R},\) a key result is that the function \(\gamma \mapsto (I^m_m, I^u_u)\) is locally constant outside of the set of discontinuities:
\[
disc := \{v_i : i \in [k-1]\} \cup \{v_i - C : i \in [k-1]\}. \]

**Proof.** Let \(\gamma_1, \gamma_2, \gamma_3, \gamma_4 \in \mathcal{R}\) satisfy the following: 1) \(\gamma_1 < \gamma_2 \leq \gamma_3 < \gamma_4,\) 2) \(\gamma_1, \gamma_4 \in \disc,\) and 3) \(\gamma \not\in \disc\) for all \(\gamma \in (\gamma_1, \gamma_4).\) Assume for the sake of contradiction that \((I^m_{m_1}, I^u_{u_1}) \neq (I^m_{m_2}, I^u_{u_2})\). Then \(I^m_{m_2} \neq I^m_{m_1}\) or \(I^u_{u_2} \neq I^u_{u_1}\). Consider the case \(I^m_{m_2} \neq I^m_{m_1}\) Then at least one of the sets \(I^m_{m_2} \setminus I^m_{m_1}\) and \(I^m_{m_1} \setminus I^m_{m_2}\) is nonempty. Consider the case when \(I^m_{m_2} \setminus I^m_{m_1}\) is nonempty. Then there exists \(i \in [k-1]\) such that \(v_i - \gamma_2 \in (0, C)\) but \(v_i - \gamma_3 \not\in (0, C).\) 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, \(\gamma' \in \disc,\) which is a contradiction. For the other cases not considered, similar arguments lead to the same contradiction. \(\square\)

Thus, as we sweep \(\gamma\) from \(+\infty\) to \(-\infty,\) we observe finitely many distinct tuples of sets \((I^m_{m_i}, I^u_{u_i})\) and their cardinals \((n^m_{m_i}, n^u_{u_i}).\) Using the index \(t = 0, 1, 2, \ldots\) we keep track of these data in the variables \((I^m_{m_t}, I^u_{u_t})\) and \((n^m_{m_t}, n^u_{u_t}).\) For this proof sketch, we make the assumption that \(|\disc| = 2(k-1),\) i.e., no elements are repeated.

By construction, the maximal element of \(\disc\) is \(v_{\text{max}}.\) When \(\gamma > v_{\text{max}},\) we check that \(n^m_{m_t} = n^u_{u_t} = \emptyset.\) Thus, we put \(I^m_{m_0} = I^u_{u_0} = \emptyset\) and \((n^m_{m_0}, n^u_{u_0}) = (0,0).\)

Now, suppose \(\gamma\) has swept across \(t - 1\) points of discontinuity and that \((I^m_{m_{t-1}}, I^u_{u_{t-1}}), n^m_{m_{t-1}}, n^u_{u_{t-1}}\) have all been defined. Suppose that \(\gamma\) crossed a single new point of discontinuity \(\gamma' \in \disc.\) In other words, \(\gamma'' < \gamma < \gamma'\) so that \(\gamma''\) is the largest element of \(\disc\) such that \(\gamma'' < \gamma'.\)

By the assumption that no elements of \(\disc\) are repeated, exactly one of the following possibilities is true:

there exists \(i \in [k-1]\) such that \(\gamma' = v_i.\) \hspace{1cm} (Entry)

there exists \(i \in [k-1]\) such that \(\gamma' = v_i - C.\) \hspace{1cm} (Exit)

Under the (Entry) case, the index \(i\) gets added to \(I^m_{m_t}\) while \(I^u_{u_t}\) remains unchanged. Hence, we have the updates
\[
I^m_{m_t} := I^m_{m_{t-1}} \cup \{i\}, \quad I^u_{u_t} := I^u_{u_{t-1}} - \{i\} 
\]
\[
n^m_{m_t} := n^m_{m_{t-1}} + 1, \quad n^u_{u_t} := n^u_{u_{t-1}} - 1. \hspace{1cm} (10)
\]

Under the (Exit) case, the index \(i\) moves from \(I^m_{m_{t-1}}\) to \(I^u_{u_{t-1}}.\) Hence, we have the updates
\[
I^m_{m_t} := I^m_{m_{t-1}} - \{i\}, \quad I^u_{u_t} := I^u_{u_{t-1}} \cup \{i\}
\]
\[
n^m_{m_t} := n^m_{m_{t-1}} - 1, \quad n^u_{u_t} := n^u_{u_{t-1}} + 1. \hspace{1cm} (11)
\]

Thus, \(\{(n^m_{m_i}, n^u_{u_i})\}_{i=0}^{2(k-1)} = \mathcal{R}.\) 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 \(v_{\text{max}} \leq 0,\) then \(\tilde{b} = 0\) satisfies the KKT conditions. Algorithm 2-line 3 handles this exceptional case. Below, we assume \(v_{\text{max}} > 0.\)

### Algorithm 2 solve_subproblem(v, C)

1: Input: \(v \in \mathbb{R}^{k-1}\)
2: Let \((1), \ldots, (k-1)\) sort \(v,\) i.e., \(v_{(1)} \geq \cdots \geq v_{(k-1)}\).
3: if \(v_{(1)} \leq 0\) then HALT and output: \(0 \in \mathbb{R}^{k-1}\).
4: \(n^u_0 := 0, n^m_0 := 0, S^0 := 0\)
5: \((\delta_1, \ldots, \delta_t) \leftarrow \text{get_up dn seq}()\) (Subroutine 3).
6: for \(t = 1, \ldots, \ell\) do
7: \((n^m_t, n^u_t, S^t) \leftarrow \text{update vars}()\) (Subroutine 4).
8: \(\tilde{\gamma}^t := \left(C \cdot n^u_t + S^t\right) / (n^m_t - 1)\)
9: if KKTcond() (Subroutine 5) returns true then
10: \(\quad \text{HALT and output: } \tilde{b}^t \in \mathbb{R}^{k-1}\) where
11: \(\quad \tilde{b}^t_{(i)} := \begin{cases} C & : i \leq n^t_m \\ v_{(i)} - \tilde{\gamma}^t & : n^t_m < i \leq n^t_u + n^t_m \\ 0 & : n^t_u + n^t_m < i. \end{cases} \)
12: end if
13: end for

Algorithm 2-line 4 initializes the state variables \(n^m_t\) and \(n^u_t\) as discussed in the last section. The variable \(S^t\) is also initialized and will be updated to maintain \(S^t = S(n^m_t,n^u_t)\) 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) \(\text{vals}\) consists of tuples \((\gamma', \delta')\) where \(\gamma' \in \disc\) and \(\delta' \in \{up, dn\}\) is a decision variable indicating whether \(\gamma'\) satisfies the (Entry) or the (Exit) condition. 2) \(\text{vals}\) is sorted so that the \(\gamma'\)'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 \((\delta_1, \ldots, \delta_t)\) consistent with the ordering.

In the “for” loop, Algorithm 2-line 7 calls Subroutine 4 which updates the variables \(n^m_t, n^u_t\) using (11) or (13), depending on \(\delta_t.\) The variable \(S^t\) is updated accordingly so that \(S^t = S(n^m_t,n^u_t).\)

We skip to Algorithm 2-line 9 which constructs the putative
Subroutine 3 get up dn seq Note: all variables from Algorithm 2 are assumed to be visible here.

1: vals ← \{(v_i, dn) : v_i > 0, i = 1, \ldots, k - 1\} ∪ 
   \{(v_i - C, up) : v_i > C, i = 1, \ldots, k - 1\} as a multiset, 
   where elements may be repeated.
2: Order the set vals = \{(\gamma_1, \delta_1), \ldots, (\gamma_\ell, \delta_\ell)\} such that 
   \gamma_1 \geq \cdots \geq \gamma_\ell, \ell = |vals|, and for all j1, j2 \in [\ell] such 
   that j1 < j2 and \gamma_j1 = \gamma_j2, we have \delta_j1 = dn implies 
   \delta_j2 = dn.
Note that by construction, for each \ell \in [\ell], there exists 
   i \in [k - 1] such that \gamma_i = v_i or \gamma_i = v_i - C.
3: Output: sequence (\delta_1, \ldots, \delta_\ell) whose elements are 
   retrieved in order from left to right.

Subroutine 4 update vars Note: all variables from 
Algorithm 2 are assumed to be visible here.

1: if \delta_t = up then
2:   \hat{n}_t := \hat{n}_t^{t-1} + 1, \quad \hat{n}_m := \hat{n}_m^{t-1} - 1
3:   \hat{S} := \hat{S}^{t-1} - v^{(n_t-1)}
4: else
5:   \hat{n}_m := \hat{n}_m^{t-1} + 1, \quad \hat{n}_t := \hat{n}_t^{t-1}.
6:   \hat{S} := \hat{S}^{t-1} + v^{(n_t+n_m)}
7: end if
8: Output: (\hat{n}_m, \hat{n}_t, \hat{S})

solution \(\hat{y}\). Observe that \(\hat{y} = \hat{y}(n_m, n_t)\) 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 \(\hat{y}\) 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.

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) noisy” 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. 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 \(\beta\) and \(w\) have been allocated. The primal 
objective is the value of (P) at the current \(w\) and the dual 
objective is \(-1\) times the value of (D2) at the current \(\beta\). 
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.

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  |

\(^4\)See Section A.6.

\(^5\)See Section A.6.2.
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 a guarantee. Furthermore, Shark uses a doubly-nested 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, which has 6 and 105 classes, respectively. The results, shown in Figure 1, support our hypothesis: Walrus and Shark are equally fast for small number of classes, and comparable to Shark for small number of classes. We implemented Walrus in the LIBLINEAR WW-SVM solver Shark on datasets with a large number of classes. We presented an algorithm called Walrus for exactly solving the WW-SVM solver Shark on datasets with a large 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 of classes, and comparable to Shark for small number of classes.

Consider a single run of Walrus on a fixed data set with a given hyperparameter $C$. Let $D_{\text{walrus}}$ denote the duality gap achieved by Walrus at the end of the $t$-th outer iteration. Let $\delta \in (0, 1)$. Define $E_{\text{walrus}}^t$ to be the elapsed time at the end of the $t$-th iteration where $t$ is minimal such that $D_{\text{walrus}}^t \leq \delta \cdot D_{\text{walrus}}^1$. Define $E_{\text{shark}}^t$ and $D_{\text{shark}}^t$ similarly. In all experiments $D_{\text{walrus}}^1/D_{\text{shark}}^1 \in [0.99999, 1.00001]$. Thus, the ratio $E_{\text{walrus}}^t/E_{\text{shark}}^t$ 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 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 later 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 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 di-
X-coordinates jittered for better visualization.

**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 multi-class 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*, 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., 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.

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-Heisner, 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(Febr):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., Alaiz, 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, ICSCCE*, 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.
A.1. Regarding offsets

In this section, we review the literature on SVMs in particular with regard to offsets. For binary kernel SVMs, Steinwart et al. (2011) demonstrates that kernel SVMs without offset achieve comparable classification accuracy as kernel SVMs with offset. Furthermore, they propose algorithms that solve kernel SVMs without offset that are significantly faster than solvers for kernel SVMs with offset.

For binary linear SVMs, Hsieh et al. (2008) introduced coordinate descent for the dual problem associated to linear SVMs without offsets, or with the bias term included in the \( w \) term. Chiu et al. (2020) studied whether the method of Hsieh et al. (2008) can be extended to allow offsets, but found evidence that the answer is negative. For multiclass linear SVMs, Keerthi et al. (2008) studied block coordinate descent for the CS-SVM and WW-SVM, both without offsets. We are not aware of a multiclass analogue to Chiu et al. (2020) although the situation should be similar.

The previous paragraph discussed coordinate descent in relation to the offset. Including the offset presents challenges to primal methods as well. In Section 6 of Shalev-Shwartz et al. (2011), the authors argue that including an unregularized offset term in the primal objective leads to slower convergence guarantee. Furthermore, Shalev-Shwartz et al. (2011) observed that including an unregularized offset did not significantly change the classification accuracy.

The original Crammer-Singer (CS) SVM was proposed without offsets (Crammer & Singer, 2001). In Section VI of (Hsu & Lin, 2002), the authors show the CS-SVM with offsets do not perform better than CS-SVM without offsets. Furthermore, CS-SVM with offsets requires twice as many iterations to converge than without.

A.2. Proof of Proposition 3.2

Below, let \( i \in [n] \) be arbitrary. First, we note that \( -\pi' = \begin{bmatrix} -1' \\ I_{k-1} \end{bmatrix} \) and so

\[
\pi' \beta_i = \begin{bmatrix} -1' \beta_i \\ \beta_i \end{bmatrix}.
\]

(14)

Now, let \( j \in [k] \), we have by (3) that

\[
[\alpha_i]_j = [-\sigma_{y_i} \pi' \beta_i]_j = [-\pi' \beta_i]_j = -1' \beta_i = -\sum_{t \in [k-1]} [\beta_i]_t = -\sum_{t \in [k]: t \neq y_i} [\beta_i]_t = -\sum_{t \in [k]: t \neq y_i} [\alpha_i]_t.
\]

(15)

Thus, \( \alpha \in \mathcal{F} \). This proves that \( \Psi(\mathcal{G}) \subseteq \mathcal{F} \).
Next, let us define another map $\Xi : \mathcal{F} \to \mathbb{R}^{(k-1)\times n}$ as follows: For each $\alpha \in \mathcal{F}$, define $\beta := \Xi(\alpha)$ block-wise by
$$
\beta_i := \text{proj}_{2:k}(\sigma_y, \alpha_i) \in \mathbb{R}^{k-1}
$$
where
$$
\text{proj}_{2:k} = [0 \ 0 \ I_{k-1}] \in \mathbb{R}^{(k-1)\times k}.
$$
By construction, we have for each $j \in [k-1]$ that $[\beta_j] = [\sigma_y, \alpha_i]_{j+1} = [\sigma_y, \alpha_i]_{j+1} = [\alpha_i]_{j+1}$. Since $j + 1 \neq 1$ for any $j \in [k-1]$, we have that $\sigma_y, (j+1) \neq y_i$ for any $j \in [k-1]$. Thus, $[\beta_j] = [\alpha_i]_{\sigma_y, (j+1)} \in [0, C]$. This proves that $\Xi(\mathcal{F}) \subseteq \mathcal{G}$.

Next, we prove that for all $\alpha \in \mathcal{F}$ and $\beta \in \mathcal{G}$, we have $\Xi(\Psi(\beta)) = \beta$ and $\Psi(\Xi(\alpha)) = \alpha$.

By construction, the $i$-th block of $\Xi(\Psi(\beta))$ is given by
$$
\text{proj}_{2:k}(\sigma_y, (-\sigma_y, \pi' \beta_i)) = -\text{proj}_{2:k}(\sigma_y, \sigma_y, \pi' \beta_i) = -\text{proj}_{2:k}(\pi' \beta_i)
$$
for the second equality, we used the fact that $\sigma_y^2 = 1$ for all $y \in [k]$. Thus, $\Xi(\Psi(\beta)) = \beta$.

Next, note that the $i$-th block of $\Psi(\Xi(\alpha))$ is, by construction,
$$
-\sigma_y, \pi' \text{proj}_{2:k} = -\sigma_y, \pi' \begin{bmatrix} 0 & 1 \ -I_{k-1} \end{bmatrix} \sigma_y, \alpha_i = -\sigma_y, \pi' \sigma_y, \alpha_i
$$
Recall that $\pi' = \begin{bmatrix} 0 & 1 \ -I_{k-1} \end{bmatrix}$ and so $\pi' = \begin{bmatrix} 0 & 1 \ -I_{k-1} \end{bmatrix}$. Therefore,
$$
\begin{bmatrix} 0 & \pi' \end{bmatrix} \sigma_y, \alpha_i = \sum_{j=2}^{k} \sigma_y, \alpha_i = \sum_{j \in [k] \setminus y_i} [\alpha_i]_j = [\sigma_y, \alpha_i]_j
$$
and, for $j = 2, \ldots, k$,
$$
[\sigma_y, \alpha_i]_j = -[\sigma_y, \alpha_i]_j.
$$
Hence, we have just shown that $[\sigma_y, \pi'] \sigma_y, \alpha_i = -\sigma_y, \alpha_i$. Continuing from (16), we have
$$
-\sigma_y, \pi' \text{proj}_{2:k} = -\sigma_y, (-\sigma_y, \pi' \beta_i) = \sigma_y, \pi', \beta_i
$$
This proves that $\Psi(\Xi(\alpha)) = \alpha$. Thus, we have shown that $\Psi$ and $\Xi$ are inverses of one another. This proves that $\Psi$ is a bijection.

Finally, we prove that $f(\Psi(\beta)) = g(\beta)$.
Recall that
$$
f(\alpha) := \frac{1}{2} \sum_{i, s \in [n]} x' x_i \alpha_i \alpha_s - \sum_{i \in [n]} \sum_{j \neq y_i} \alpha_{ij}
$$
Thus,
$$
\alpha' \alpha_s = (-\sigma_y, \pi' \beta_i)'(-\sigma_y, \pi' \beta_s) = \beta_i^s \sigma_y, \sigma_y, \pi' \beta_s
$$
On the other hand, (3) implies that $\sigma_y, \alpha_i = -\pi' \beta_i$. Hence
$$
\sum_{j \in [k] \setminus y_i} \alpha_{ij} = \sum_{j \in [k] \setminus y_i} [\alpha_i]_{\sigma_y, (j+1)} = \sum_{j \in [k] \setminus y_i} [\sigma_y, \alpha_i]_j = \sum_{j \in [k] \setminus y_i} [\sigma_y, \alpha_i]_j = \sum_{j \in [k] \setminus y_i} [\beta_j]_j = 1' \beta_i.
$$
Thus,
$$
f(\alpha) := \frac{1}{2} \sum_{i, s \in [n]} x' x_i \alpha_i \alpha_s - \sum_{i \in [n]} \sum_{j \neq y_i} \alpha_{ij} = \frac{1}{2} \sum_{i, s \in [n]} x' x_i \beta_i^s \pi \sigma_y, \sigma_y, \pi' \beta_s - \sum_{i \in [n]} 1' \beta_i = g(\beta)
$$
as desired. Finally, we note that $\sigma_y = \sigma_y'$ for all $y \in [k]$. This concludes the proof of Proposition 3.2.
A.3. Proof of Proposition 3.5

We prove the following lemma which essentially unpacks the succinct Proposition 3.5:

Lemma A.1. Recall the situation of Corollary 3.3: Let \( \beta \in \mathcal{G} \) and \( i \in [n] \). Let \( \alpha = \Psi(\beta) \). Consider

\[
\min_{\tilde{\beta} \in \mathcal{G}} g(\tilde{\beta}) \text{ such that } \tilde{\beta}_s = \beta_s, \forall s \in [n] \setminus \{i\}.
\]

Let \( w \) be as in (1), i.e., \( w = -\sum_{i \in [n]} x_i c_i \). Then a solution to (17) is given by \( [\beta_1, \ldots, \beta_{i-1}, \tilde{\beta}_i, \beta_{i+1}, \ldots, \beta_n] \) where \( \tilde{\beta}_i \) is a minimizer of

\[
\min_{\tilde{\beta}_i \in \mathbb{R}^{k-1}} \frac{1}{2} \tilde{\beta}_i \left( \Theta \tilde{\beta}_i - \beta_i' \right) \left( (1 - \pi \sigma_y, w' x_i) / \|x_i\|^2 \Theta + \Theta \beta_i \right) \text{ such that } 0 \leq \tilde{\beta}_i \leq C.
\]

Furthermore, the above optimization has a unique minimizer which is equal to the minimizer of (4) where

\[
v := (1 - \rho_y, \pi w' x_i + \Theta \beta_i \|x_i\|^2 / \|x_i\|^2)
\]

and \( w \) is as in (1).

Proof. First, we prove a simple identity:

\[
\pi \pi' = \begin{bmatrix} 1 & -I_{k-1} \\
-1 & -I_{k-1} \end{bmatrix} = I + O = \Theta.
\]

Next, recall that by definition, we have

\[
g(\beta) := \left( \frac{1}{2} \sum_{s,t \in [n]} x_s x_t \beta_t' \pi \sigma_y, \beta_s \pi' \beta_s \right) - \left( \sum_{s \in [n]} \| \beta_s' \|^2 \right).
\]

Let us group the terms of \( g(\beta) \) that depends on \( \beta_i \):

\[
g(\beta) = \frac{1}{2} x_i' x_i \beta_i' \pi \sigma_y, \beta_i \pi' \beta_i + \frac{1}{2} \sum_{s \in [n]: s \neq i} x_s x_i \beta_i' \pi \sigma_y, \beta_s \pi' \beta_s + \frac{1}{2} \sum_{t \in [n]: t \neq i} x_t x_i \beta_i' \pi \sigma_y, \beta_t \pi' \beta_t + \frac{1}{2} \sum_{s,t \in [n]} x_s x_t \beta_t' \pi \sigma_y, \beta_s \pi' \beta_s - \sum_{s \in [n]} \| \beta_s' \|^2
\]

\[
= \frac{1}{2} x_i' x_i \beta_i' \Theta \beta_i \quad \text{and (18)}
\]

\[
\quad + \sum_{s \in [n]: s \neq i} x_s x_i \beta_i' \pi \sigma_y, \beta_s \pi' \beta_s - \| \beta_i' \|^2
\]

\[
\quad + \frac{1}{2} \sum_{s,t \in [n]} x_s x_t \beta_t' \pi \sigma_y, \beta_s \pi' \beta_s - \sum_{s \in [n]: s \neq i} \| \beta_s' \|^2
\]

\[=: C_i
\]

where \( C_i \) is a scalar quantity which does not depend on \( \beta_i \). Thus, plugging in \( \tilde{\beta} \), we have

\[
g(\tilde{\beta}) = \frac{1}{2} \|x_i\|^2 \beta_i' \Theta \beta_i + \sum_{s \in [n]: s \neq i} x_s x_i \beta_i' \pi \sigma_y, \beta_s \pi' \beta_s - \| \beta_i' \|^2 + C_i.
\]
Furthermore,

\[
\sum_{s \in [n]: s \neq i} x'_s x_i \hat{\beta}'_i \pi \sigma y_i \sigma_y \pi' \beta_s = \sum_{s \in [n]: s \neq i} \hat{\beta}'_i \pi \sigma y_i \sigma_y \pi' \beta_s x'_s x_i
\]

\[
= \hat{\beta}'_i \pi \sigma y_i \left( \sum_{s \in [n]: s \neq i} \sigma_y \pi' \beta_s x'_s \right) x_i
\]

\[
= \hat{\beta}'_i \pi \sigma y_i \left( -\sigma_y \pi' \beta_i x'_i + \sum_{s \in [n]} \sigma_y \pi' \beta_s x'_s \right) x_i
\]

\[
= \hat{\beta}'_i \pi \sigma y_i \left( -\sigma_y \pi' \beta_i x'_i - \sum_{s \in [n]} \alpha_s x'_s \right) x_i \quad \therefore (3)
\]

\[
= \hat{\beta}'_i \pi \sigma y_i \left( -\sigma_y \pi' \beta_i x'_i + w' x_i \right) \therefore (1)
\]

\[
= \hat{\beta}'_i \left( \pi \sigma y_i w' x_i - \pi' \beta_i \|x_i\|^2 \right) \therefore \sigma^2_{\beta_i} = 1
\]

\[
= \hat{\beta}'_i \left( \pi \sigma y_i w' x_i - \Theta \beta_i \|x_i\|^2 \right) \therefore (18)
\]

Therefore, we have

\[
g(\tilde{\beta}) = \frac{1}{2} \|x_i\|^2 \hat{\beta}'_i \Theta \hat{\beta}_i + \hat{\beta}'_i \left( \pi \sigma y_i w' x_i - \Theta \beta_i \|x_i\|^2 \right) + C_i
\]

\[
= \frac{1}{2} \|x_i\|^2 \hat{\beta}'_i \Theta \hat{\beta}_i - \hat{\beta}'_i \left( 1 - \pi \sigma y_i w' x_i + \Theta \beta_i \|x_i\|^2 \right) + C_i
\]

Thus, (17) is equivalent to

\[
\min_{\beta \in \mathcal{V}} \quad \frac{1}{2} \|x_i\|^2 \hat{\beta}'_i \Theta \hat{\beta}_i - \hat{\beta}'_i \left( 1 - \pi \sigma y_i w' x_i + \Theta \beta_i \|x_i\|^2 \right) + C_i
\]

\[
\quad \text{s.t.} \quad \beta_s = \beta_s, \forall s \in [n] \setminus \{i\}.
\]

Dropping the constant $C_i$ and dividing through by $\|x_i\|^2$ does not change the minimizers. Hence, (17) has the same set of minimizers as

\[
\min_{\beta \in \mathcal{V}} \quad \frac{1}{2} \hat{\beta}' \Theta \hat{\beta}_i - \hat{\beta}'_i \left( (1 - \pi \sigma y_i w' x_i) / \|x_i\|^2 + \Theta \beta_i \|x_i\|^2 \right) + C_i
\]

\[
\quad \text{s.t.} \quad \beta_s = \beta_s, \forall s \in [n] \setminus \{i\}.
\]

Due to the equality constraints, the only free variable is $\hat{\beta}_i$. Note that the above optimization, when restricted to $\hat{\beta}_i$, is equivalent to the optimization (4) with

\[
v := (1 - \pi \sigma y_i w' x_i) / \|x_i\|^2 + \Theta \beta_i
\]

and $w$ is as in (1). The uniqueness of the minimizer is guaranteed by Theorem 3.4.

\[
\Box
\]

### A.4. Global linear convergence

Wang & Lin (2014) established the global linear convergence of the so-called feasible descent method when applied to a certain class of problems. As an application, they prove global linear convergence for coordinate descent for solving the dual problem of the binary SVM with the hinge loss. Wang & Lin (2014) considered optimization problems of the following form:

\[
\min_{x \in \mathcal{X}} f(x) := g(Ex) + b'x
\]
where $f: \mathbb{R}^n \to \mathbb{R}$ is a function such that $\nabla f$ is Lipschitz continuous, $\mathcal{X} \subseteq \mathbb{R}^n$ is a polyhedral set, $\arg \min_{x \in \mathcal{X}} f(x)$ is nonempty, $g: \mathbb{R}^m \to \mathbb{R}$ is a strongly convex function such that $\nabla g$ is Lipschitz continuous, and $E \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$ are fixed matrix and vector, respectively.

Below, let $\mathcal{P}_\mathcal{X}: \mathbb{R}^n \to \mathcal{X}$ denote the orthogonal projection on $\mathcal{X}$.

**Definition A.2.** In the context of (20), an iterative algorithm that produces a sequence $\{x^0, x^1, x^2, \ldots\} \subseteq \mathcal{X}$ is a feasible descent method if there exists a sequence $\{\epsilon^0, \epsilon^1, \epsilon^2, \ldots\} \subseteq \mathbb{R}^n$ such that for all $t \geq 0$

$$
x^{t+1} = \mathcal{P}_\mathcal{X} \left( x^t - \nabla f(x^t) + \epsilon^t \right) \tag{21}
$$

$$
\|\epsilon^t\| \leq B \|x^t - x^{t+1}\| \tag{22}
$$

$$
f(x^t) - f(x^{t+1}) \geq \Gamma \|x^t - x^{t+1}\|^2 \tag{23}
$$

where $B, \Gamma > 0$.

One of the main result of (Wang & Lin, 2014) is

**Theorem A.3 (Theorem 8 from (Wang & Lin, 2014)).** Suppose an optimization problem $\min_{x \in \mathcal{X}} f(x)$ is of the form (20) and $\{x^0, x^1, x^2, \ldots\} \subseteq \mathcal{X}$ is a sequence generated by a feasible descent method. Let $f^* := \min_{x \in \mathcal{X}} f(x)$. Then there exists $\Delta \in (0, 1)$ such that

$$
f(x^{t+1}) - f^* \leq \Delta (f(x^t) - f^*), \quad \forall t \geq 0.
$$

Now, we begin verifying that the WW-SVM dual optimization and the BCD algorithm for WW-SVM satisfies the requirements of Theorem A.3.

Given $\beta \in \mathbb{R}^{(k-1) \times n}$, define its vectorization

$$
\text{vec}(\beta) = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} \in \mathbb{R}^{(k-1)n}.
$$

Define the matrix $P_{is} = \pi \sigma_{yi} x_i \sigma_{yj} \pi' \in \mathbb{R}^{(k-1) \times (k-1)}$, and $Q \in \mathbb{R}^{(k-1)n \times (k-1)n}$ by

$$
Q = \begin{bmatrix}
P_{11} & P_{12} & \cdots & P_{1n} \\
P_{21} & P_{22} & \cdots & P_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
P_{n1} & P_{n2} & \cdots & P_{nn}
\end{bmatrix}.
$$

Let

$$
E = \begin{bmatrix}
x_1 \sigma_{yi} \pi' \\
x_2 \sigma_{yj} \pi' \\
\vdots \\
x_n \sigma_{yn} \pi'
\end{bmatrix}.
$$

We observe that $Q = E' E$. Thus, $Q$ is symmetric and positive semi-definite. Let $\|Q\|_{op}$ be the operator norm of $Q$.

**Proposition A.4.** The optimization (D2) is of the form (20). More precisely, the optimization (D2) can be expressed as

$$
\min_{\beta \in \mathcal{V}} g(\beta) = \varphi(E \text{vec}(\beta)) - \|\text{vec}(\beta)\|
$$

where the feasible set $\mathcal{G}$ is a nonempty polyhedral set (i.e., defined by a system of linear inequalities, hence convex), $\varphi$ is strongly convex, $\nabla g$ is Lipschitz continuous with Lipschitz constant $L := \|Q\|_{op}$. Furthermore, (24) has at least one minimizer.
Proof. Observe
\[
g(\beta) = \frac{1}{2} \sum_{i,e \in [n]} x_i^x x_i^y \pi \sigma, \pi', \pi' \sigma, \pi \sigma' - \sum_{i \in [n]} \beta_i^o
\]
\[
= \frac{1}{2} \text{vec}(\beta)'Q \text{vec}(\beta) - 1' \text{vec}(\beta)
\]
\[
= \frac{1}{2} (E \text{vec}(\beta))' (E \text{vec}(\beta)) - 1' \text{vec}(\beta)
\]
\[
= \varphi(\text{vec}(\beta)) - 1' \text{vec}(\beta)
\]
where \(\varphi(\beta) = \frac{1}{2} \| \beta \|^2\). Note that \(\text{vec}(\nabla g(\beta)) = Q \text{vec}(\beta) - 1\). Hence, the Lipschitz constant of \(g\) is \(\|Q\|_{op}\). For the “Furthermore” part, note that the above calculation shows that (24) is a quadratic program where the second order term is positive semi-definite and the constraint set is convex. Hence, (24) has at least one minimizer. \(\square\)

Let \(B = [0, C]^{k-1}\). Let \(\beta^t\) be \(\beta\) at the end of the \(t\)-iteration of the outer loop of Algorithm 1. Define
\[
\beta^t := [\beta^t_1, \cdots, \beta^t_{t-1}, \beta^t_{t+1}, \cdots, \beta^t_n].
\]
By construction, we have
\[
\beta^t_1 = \arg \min_{\beta \in B} g \left( |\beta^t_1, \cdots, \beta^t_{t-1}, 1, \beta^t_{t+1}, \cdots, \beta^t_n| \right)
\]
For each \(i = 1, \cdots, n\), let
\[
\nabla_i g(\beta) = \left[ \frac{\partial g}{\partial \beta_{1i}}(\beta), \frac{\partial g}{\partial \beta_{2i}}(\beta), \cdots, \frac{\partial g}{\partial \beta_{(k-1)i}}(\beta) \right]'.
\]
By Lemma 24 (Wang & Lin, 2014), we have
\[
\beta^t_{i+1} = \mathcal{P}_B(\beta^t_i - \nabla_i g(\beta^t_i))
\]
where \(\mathcal{P}_B\) denotes orthogonal projection on to \(B\). Now, define \(\epsilon^t \in \mathbb{R}^{(k-1) \times n}\) such that
\[
\epsilon^t_i = \beta^t_{i+1} - \beta^t_i - \nabla_i g(\beta^t_i) + \nabla_i g(\beta^t_i).
\]

**Proposition A.5.** The BCD algorithm for the WW-SVM is a feasible descent method. More precisely, the sequence \(\{\beta^0, \beta^1, \ldots\}\) satisfies the following conditions:

\[
\beta^t_{i+1} = \mathcal{P}_\mathcal{G} \left( \beta^t_i - \nabla g(\beta^t_i) + \epsilon^t \right)\]
\[
\| \epsilon^t \| \leq (1 + \sqrt{\nu}L)\| \beta^t_i - \beta^t_{i+1} \|
\]
\[
g(\beta^t_i) - g(\beta^t_{i+1}) \geq \Gamma \| \beta^t_i - \beta^t_{i+1} \|^2
\]
where \(L\) is as in Proposition A.4, \(\Gamma := \min_{i \in [n]} \frac{\| x_i \|^2}{2} \), \(\mathcal{G}\) is the feasible set of (D2), and \(\mathcal{P}_\mathcal{G}\) is the orthogonal projection onto \(\mathcal{G}\).

The proof of Proposition A.5 essentially generalizes Proposition 3.4 of (Luo & Tseng, 1993) to the higher dimensional setting:

**Proof.** Recall that \(\mathcal{G} = B^{\times n} := B \times \cdots \times B\). Note that the \(i\)-th block of \(\beta^t_i - \nabla g(\beta^t_i) + \epsilon^t\) is
\[
\beta^t_i - \nabla_i g(\beta^t_i) + \epsilon^t_i = \beta^t_i - \nabla_i g(\beta^t_i) + (\beta^t_{i+1} - \beta^t_i - \nabla_i g(\beta^t_i) + \nabla_i g(\beta^t_i)) = \beta^t_{i+1} - \nabla_i g(\beta^t_{i+1}).
\]
Thus, the \(i\)-th block of \(\mathcal{P}_\mathcal{G}(\beta^t_i - \nabla g(\beta^t_i) + \epsilon^t)\) is
\[
\mathcal{P}_B(\beta^t_{i+1} - \nabla_i g(\beta^t_{i+1})) = \beta^t_{i+1}.
\]
This is precisely the identity (26).
Next, we have

\[ \|\epsilon_i^t\| \leq \|\beta_{i+1}^t - \beta_i^t\| + \|\nabla_i g(\beta^{t,i}) - \nabla_i g(\beta^t)\| \leq \|\beta_{i+1}^t - \beta_i^t\| + L\|\beta_i^{t,i} - \beta_i^t\| \leq \|\beta_{i+1}^t - \beta_i^t\| + L\|\beta^{t+1} - \beta^t\|. \]

From this, we get that

\[ \|\epsilon^t\| = \sqrt{\sum_{i=1}^{n} \|\epsilon_i^t\|^2} \leq \sqrt{\sum_{i=1}^{n} (\|\beta_{i+1}^t - \beta_i^t\| + L\|\beta_i^{t,i} - \beta_i^t\|)^2} \leq \sqrt{\sum_{i=1}^{n} \|\beta_{i+1}^t - \beta_i^t\|^2 + \sum_{i=1}^{n} L^2\|\beta_i^{t,i} - \beta_i^t\|^2} = \|\beta^{t+1} - \beta^t\| + \sqrt{nL}\|\beta^{t+1} - \beta^t\| = (1 + \sqrt{nL})\|\beta^{t+1} - \beta^t\|. \]

Thus, we conclude that \( \|\epsilon^t\| \leq (1 + \sqrt{nL})\|\beta^{t+1} - \beta^t\| \) which is (27).

Finally, we show that

\[ g(\beta^{t,i-1}) - g(\beta^{t,i}) + \nabla_i g(\beta^{t,i})(\beta_i^{t+1} - \beta_i^t) \geq \Gamma\|\beta_i^{t+1} - \beta_i^t\|^2 \]

where \( \Gamma := \min_{i \leq [n]} \frac{\|x_i\|^2}{2} \).

**Lemma A.6.** Let \( \beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n \in \mathbb{R}^{k-1} \) be arbitrary. Then there exist \( v \in \mathbb{R}^{k-1} \) and \( C \in \mathbb{R} \) which depend only on \( \beta_1, \cdots, \beta_{i-1}, \beta_{i+1}, \cdots, \beta_n \), but not on \( \beta \), such that

\[ g \left( (\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n) \right) = \frac{1}{2} \|x_i\|^2 \beta - v'\beta - C. \]

In particular, we have

\[ \nabla_i g \left( (\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n) \right) = \|x_i\|^2 \beta - v. \]

**Proof.** The result follows immediately from the identity (19). \( \square \)

**Lemma A.7.** Let \( \beta_1, \cdots, \beta_{i-1}, \beta, \eta, \beta_{i+1}, \cdots, \beta_n \in \mathbb{R}^{k-1} \) be arbitrary. Then we have

\[ g \left( (\beta_1, \cdots, \beta_{i-1}, \eta, \beta_{i+1}, \cdots, \beta_n) \right) - g \left( (\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n) \right) + \nabla_i g \left( (\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n) \right) \cdot (\beta - \eta) = \frac{\|x_i\|^2}{2} \|\eta - \beta\|^2 \]

**Proof.** Let \( v, C \) be as in Lemma A.6. We have

\[ g \left( (\beta_1, \cdots, \beta_{i-1}, \eta, \beta_{i+1}, \cdots, \beta_n) \right) - g \left( (\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n) \right) = \frac{\|x_i\|^2}{2} \|\eta\|^2 - \frac{\|x_i\|^2}{2} \|\beta\|^2 + v'\beta \]

and

\[ = \frac{\|x_i\|^2}{2} (\|\eta\|^2 - \|\beta\|^2) + v'(\beta - \eta) \]
\[ \nabla_i g([\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n])' (\beta - \eta) = (\|x_i\|^2 \beta - v)' (\beta - \eta) = \|x_i\|^2 (\|\beta\|^2 - \beta' \eta) - \beta' (\beta - \eta). \]

Thus,

\[
g([\beta_1, \cdots, \beta_{i-1}, \eta, \beta_{i+1}, \cdots, \beta_n]) - g([\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n]) + \nabla_i g([\beta_1, \cdots, \beta_{i-1}, \beta, \beta_{i+1}, \cdots, \beta_n])' (\beta - \eta) \\
= \frac{\|x_i\|^2}{2} (\|\eta\|^2 - \|\beta\|^2) + v' (\beta - \eta) + \|x_i\|^2 (\|\beta\|^2 - \beta' \eta) - v' (\beta - \eta) \\
= \frac{\|x_i\|^2}{2} (\|\eta\|^2 - \|\beta\|^2) + \|x_i\|^2 (\|\beta\|^2 - \beta' \eta) \\
= \|x_i\|^2 \left( \frac{1}{2} (\|\eta\|^2 - \|\beta\|^2) + (\|\beta\|^2 - \beta' \eta) \right) \\
= \|x_i\|^2 \left( \frac{1}{2} (\|\eta\|^2 + \|\beta\|^2) - \beta' \eta \right) \\
= \frac{\|x_i\|^2}{2} \|\eta - \beta\|^2
\]

as desired.

Applying Lemma A.7, we have

\[
g(\beta^{t,i-1}) - g(\beta^{t,i}) + \nabla_i g(\beta^{t,i})' (\beta^{t+1}_i - \beta^t_i) \geq \frac{\|x_i\|^2}{2} \|\beta^{t+1}_i - \beta^t_i\|^2.
\]

Since (25) is true, we have by Lemma 24 of (Wang & Lin, 2014) that

\[
\nabla_i g(\beta^{t,i})' (\beta^t_i - \beta^{t+1}_i) \geq 0
\]

Equivalently, \(\nabla_i g(\beta^{t,i})' (\beta^{t+1}_i - \beta^t_i) \leq 0\). Thus, we deduce that

\[
g(\beta^{t,i-1}) - g(\beta^{t,i}) \geq \frac{\|x_i\|^2}{2} \|\beta^{t+1}_i - \beta^t_i\|^2 \geq \Gamma \|\beta^{t+1}_i - \beta^t_i\|^2
\]

Summing the above identity over \(i \in [n]\), we have

\[
g(\beta^{t,0}) - g(\beta^{t,n}) = \sum_{i=1}^{n} g(\beta^{t,i-1}) - g(\beta^{t,i}) \geq \Gamma \sum_{i=1}^{n} \|\beta^{t+1}_i - \beta^t_i\|^2 = \Gamma \|\beta^{t+1} - \beta^t\|^2
\]

Since \(\beta^{t,0} = \beta^{t}\) and \(\beta^{t,n} = \beta^{t+1}\), we conclude that \(g(\beta^{t}) - g(\beta^{t+1}) \geq \Gamma \|\beta^{t+1} - \beta^t\|^2\).

To conclude the proof of Theorem 3.6, we note that Proposition A.5 and Proposition A.4 together imply that the requirements of Theorem 8 from (Wang & Lin, 2014) (restated as Theorem A.3 here) are satisfied for the BCD algorithm for WW-SVM. Hence, we are done.

**A.5. Proof of Theorem 3.4**

The goal of this section is to prove Theorem 3.4. The time complexity analysis has been carried out at the end of Section 4 of the main article. Below, we focus on the part of the theorem on the correctness of the output. Throughout this section, \(k \geq 2\), \(C > 0\) and \(v \in \mathbb{R}^{k-1}\) are assumed to be fixed. Additional variables used are summarized in Table 2.
and hence has a unique global minimum \( \tilde{b} \).

We first prove part 1. The optimization (4) is a minimization over a convex domain with strictly convex objective,

\[ \text{Proof.} \]

Let \( v_{\max} = \max_{i \in [k-1]} v_i \). The optimization (4) has a unique global minimum \( \tilde{b} \) satisfying the following:

1. \( \tilde{b} = \text{clip}_C(v - \tilde{\gamma}1) \) for some \( \tilde{\gamma} \in \mathbb{R} \)
2. \( \tilde{\gamma} = \sum_{i=1}^{k-1} \tilde{b}_i \). In particular, \( \tilde{\gamma} \geq 0 \).
3. If \( v_i \leq 0 \), then \( \tilde{b}_i = 0 \). In particular, if \( v_{\max} \leq 0 \), then \( \tilde{b} = 0 \).
4. If \( v_{\max} > 0 \), then \( 0 < \tilde{\gamma} < v_{\max} \).

\[ \text{Proof.} \] First we prove part 1. The optimization (4) is a minimization over a convex domain with strictly convex objective, and hence has a unique global minimum \( \tilde{b} \). For each \( i \in [k-1] \), let \( \lambda_i, \mu_i \in \mathbb{R} \) be the dual variables for the constraints \( 0 \geq b_i - C \) and \( 0 \geq -b_i \), respectively. The Lagrangian for the optimization (4) is

\[ \mathcal{L}(b, \lambda, \mu) = \frac{1}{2} b'(I + C)b - v'b + (b - C)' \lambda + (-b)' \mu. \]

Thus, the stationarity (or gradient vanishing) condition is

\[ 0 = \nabla_b \mathcal{L}(b, \lambda, \mu) = (I + O)b - v + \lambda - \mu. \]

The KKT conditions are as follows:

for all \( i \in [k-1] \), the following holds:

\[ [(I + O)b]_i + \lambda_i - \mu_i = v_i \quad \text{stationarity} \quad (29) \]

\[ C \geq b_i \geq 0 \quad \text{primal feasibility} \quad (30) \]

\[ \lambda_i \geq 0 \quad \text{dual feasibility} \quad (31) \]

\[ \mu_i \geq 0 \quad \text{complementary slackness} \quad (32) \]

\[ \lambda_i(C - b_i) = 0 \quad \text{complementary slackness} \quad (33) \]

\[ \mu_i b_i = 0 \quad \text{complementary slackness} \quad (34) \]
(29) to (34) are satisfied if and only if \( \overline{b} = \overline{b} \) is the global minimum.

Let \( \gamma \in \mathbb{R} \) be such that \( \gamma 1 = O \overline{b} \). Note that by definition, part 2 holds. Furthermore, (29) implies

\[
\overline{b} = v - \gamma 1 - \lambda + \mu. \tag{35}
\]

Below, fix some \( i \in [k - 1] \). Note that \( \lambda_i \) or \( \mu_i \) cannot both be nonzero. Otherwise, (33) and (34) would imply that \( C = \overline{b}_i = 0 \), a contradiction. We claim the following:

1. If \( v_i - \gamma \in [0, C] \), then \( \lambda_i = \mu_i = 0 \) and \( \overline{b}_i = v_i - \gamma \).
2. If \( v_i - \gamma > C \), then \( \overline{b}_i = C \).
3. \( v_i - \gamma < 0 \), then \( \overline{b}_i = 0 \).

We prove the first claim. To this end, suppose \( v_i - \gamma \in [0, C] \). We will show \( \lambda_i = \mu_i = 0 \) by contradiction. Suppose \( \lambda_i > 0 \). Then we have \( C = \overline{b}_i \) and \( \mu_i = 0 \). Now, (35) implies that \( C = \overline{b}_i = v_i - \gamma - \lambda_i \). However, we now have \( v_i - \gamma - \lambda_i \leq C - \lambda_i < C \), a contradiction. Thus, \( \lambda_i = 0 \). Similarly, assuming \( \mu_i > 0 \) implies

\[
0 = \overline{b}_i = v_i - \lambda + \mu_i \geq 0 + \mu_i > 0,
\]

a contradiction. This proves the first claim.

Next, we prove the second claim. Note that

\[
C \geq \overline{b}_i = v_i - \gamma - \lambda_i + \mu_i > C - \lambda_i + \mu_i \implies 0 > -\lambda_i + \mu_i \geq -\lambda_i.
\]

In particular, we have \( \lambda_i > 0 \) which implies \( C = \overline{b}_i \) by complementary slackness.

Finally, we prove the third claim. Note that

\[
0 \leq \overline{b}_i = v_i - \gamma - \lambda_i + \mu_i < -\lambda_i + \mu_i \leq \mu_i
\]

Thus, \( \mu_i > 0 \) and so \( 0 = \overline{b}_i \) by complementary slackness. This proves that \( \overline{b} = \text{clip}_C(v - \gamma 1) \), which concludes the proof of part 1.

For part 2, note that \( \gamma = \sum_{i=1}^{k-1} \overline{b}_i \) holds by definition. The “in particular” portion follows immediately from \( \overline{b} \geq 0 \).

We prove part 3 by contradiction. Suppose there exists \( i \in [k - 1] \) such that \( v_i \leq 0 \) and \( \overline{b}_i > 0 \). Thus, by (34), we have \( \mu_i = 0 \). By (29), we have \( b_i + \gamma \leq b_i + \gamma + \lambda_i = v_i \leq 0 \). Thus, we have \( -\gamma \geq b_i > 0 \), or equivalently, \( \gamma < 0 \).

However, this contradicts part 2. Thus, \( \overline{b}_i = 0 \) whenever \( v_i \leq 0 \). The “in particular” portion follows immediately from the observation that \( v_{\max} \leq 0 \) implies that \( v_i \leq 0 \) for all \( i \in [k - 1] \).

For part 4, we first prove that \( \gamma < v_{\max} \) by contradiction. Suppose that \( \gamma \geq v_{\max} \). Then we have \( v - \gamma 1 \leq v - v_{\max} 1 \leq 0 \).

Thus, by part 1, we have \( \overline{b} = \text{clip}_C(v - \gamma 1) = 0 \). By part 2, we must have that \( \gamma = \sum_{i=1}^{k-1} \overline{b}_i = 0 \). However, \( \gamma \geq v_{\max} > 0 \), which is a contradiction.

Finally, we prove that \( \gamma > 0 \) again by contradiction. Suppose that \( \gamma = 0 \). Then part 2 and the fact that \( \overline{b} \geq 0 \) implies that \( \overline{b} = 0 \). However, by part 1, we have \( \overline{b} = \text{clip}_C(v) \). Now, let \( i^* \) be such that \( v_{i^*} = v_{\max} \). This implies that \( \overline{b}_{i^*} = \text{clip}_C(v_{\max}) > 0 \), a contradiction.

\[\square\]

A.5.2. Recovering \( \gamma \) from discrete data

**Definition A.10.** For \( \gamma \in \mathbb{R} \), let \( b^\gamma := \text{clip}_C(v - \gamma 1) \in \mathbb{R}^{k-1} \). Define

\[
I_u^\gamma := \{ i \in [k - 1] : b^\gamma_i = C \}
\]

\[
I_m^\gamma := \{ i \in [k - 1] : b^\gamma_i \in (0, C) \}
\]

\[
n_u^\gamma := |I_u^\gamma|, \quad \text{and} \quad n_m^\gamma := |I_m^\gamma|.
\]

Let \( \|k\| := \{0\} \cup [k - 1] \). Note that by definition, \( n_m^\gamma, n_u^\gamma \in \|k\| \).
Note that \( I_u^* \) and \( I_m^* \) are determined by their cardinalities. This is because
\[
I_u^* = \{(1), (2), \ldots, (n_u^*)\}, \\
I_m^* = \{(n_u^* + 1), (n_u^* + 2), \ldots, (n_u^* + n_m^*)\}.
\]

**Definition A.11.** Define
\[
disc^+ := \{v_i : i \in [k - 1], v_i > 0\} \cup \{v_i - C : i \in [k - 1], v_i - C > 0\} \cup \{0\}.
\]

Note that \( disc^+ \) is slightly different from \( disc \) as defined in the main text.

**Lemma A.12.** Let \( \gamma', \gamma'' \in disc^+ \) be such that \( \gamma \notin disc^+ \) for all \( \gamma \in (\gamma', \gamma'') \). The functions
\[
(\gamma', \gamma'') \ni \gamma \mapsto I_m^\gamma \\
(\gamma', \gamma'') \ni \gamma \mapsto I_u^\gamma
\]
are constant.

**Proof.** We first prove \( I_m^\lambda = I_u^\rho \). Let \( \lambda, \rho \in (\gamma', \gamma'') \) be such that \( \lambda < \rho \). Assume for the sake of contradiction that \( I_m^\lambda \neq I_u^\rho \). Then either 1) \( i \in [k - 1] \) such that \( v_i - \lambda \in (0, C) \) but \( v_i - \rho \notin (0, C) \) or 2) \( i \in [k - 1] \) such that \( v_i - \lambda \notin (0, C) \) but \( v_i - \rho \in (0, C) \). This implies that there exists some \( \gamma \in (\lambda, \rho) \) such that \( v_i - \gamma \notin \{0, C\} \), or equivalently, \( \gamma \in \{v_i, v_i - C\} \). Hence, \( \gamma \in disc^+ \), which is a contradiction. Thus, for all \( \lambda, \rho \in (\gamma', \gamma'') \), we have \( I_m^\lambda = I_u^\rho \).

Next, we prove \( I_m^\lambda = I_u^\rho \). Let \( \lambda, \rho \in (\gamma', \gamma'') \) be such that \( \lambda < \rho \). Assume for the sake of contradiction that \( I_m^\lambda \neq I_u^\rho \). Then either 1) \( i \in [k - 1] \) such that \( v_i - \lambda \geq C \) but \( v_i - \rho < C \) or 2) \( i \in [k - 1] \) such that \( v_i - \lambda < C \) but \( v_i - \rho \geq C \). This implies that there exists some \( \gamma \in (\lambda, \rho) \) such that \( v_i - \gamma = C \), or equivalently, \( \gamma = v_i \). Hence, \( \gamma \in disc^+ \), which is a contradiction. Thus, for all \( \lambda, \rho \in (\gamma', \gamma'') \), we have \( I_m^\lambda = I_u^\rho \). \( \qed \)

**Definition A.13.** For \((n_m, n_u) \in \|k\|^2\), define \( S^{(n_m, n_u)}, \tilde{\gamma}^{(n_m, n_u)} \in \mathbb{R} \) by
\[
S^{(n_m, n_u)} := \sum_{i=n_u+1}^{n_u+n_m} v_{(i)}, \\
\tilde{\gamma}^{(n_m, n_u)} := (C \cdot n_u + S^{(n_m, n_u)}) / (n_m + 1).
\]

Furthermore, define \( \tilde{b}^{(n_m, n_u)} \in \mathbb{R}^{k - 1} \) such that, for \( i \in [k - 1] \), the \( \langle i \rangle \)-th entry is
\[
\tilde{b}^{(n_m, n_u)}_{(i)} := \begin{cases} 
C & : i \leq n_u \\
v_{(i)} - \gamma^{(n_m, n_u)} & : n_u < i \leq n_u + n_m \\
0 & : n_u + n_m < i.
\end{cases}
\]

Below, recall \( \ell \) as defined on Subroutine 3-line 2.

**Lemma A.14.** Let \( t \in [\ell] \). Let \( n_u^t, n_u^{t-1} \) and \( \tilde{b} \) be as in the for loop of Algorithm 2. Then \( \tilde{\gamma}^{(n_u^t, n_u^{t-1})} = \tilde{\gamma}^t \) and \( \tilde{b}^{(n_u^t, n_u^{t-1})} = \tilde{b}^t \).

**Proof.** It suffices to show that \( S^t = S^{(n_u^t, n_u^{t-1})} \) where the former is defined as in Algorithm 2 and the latter is defined as in Definition A.13. In other words, it suffices to show that
\[
S^t = \sum_{j \in [k - 1] : n_u^t < j \leq n_u^{t-1} + n_u^{t-1}} v_{(j)}. \quad (36)
\]

We prove (36) by induction. The base case \( t = 0 \) follows immediately due to the initialization in Algorithm 2-line 4.

Now, suppose that (36) holds for \( S^{t-1} \):
\[
S^{t-1} = \sum_{j \in [k - 1] : n_u^{t-1} < j \leq n_u^{t-1-1} + n_u^{t-1}} v_{(j)}. \quad (37)
\]
Consider the first case that $\delta_i = \text{up}$. Then we have $n_u^t + n_m^t = n_u^{t-1} + n_m^{t-1}$ and $n_u^t = n_u^{t-1} + 1$. Thus, we have

$$S^t = S^{t-1} - v_{(n_u^{t-1})} \quad \therefore \text{Subroutine 4-line 3,}$$

$$= \sum_{j \in [k-1]: n_u^{t-1} + 1 < j \leq n_u^{t-1} + n_m^{t-1}} v_{(j)} \quad \therefore (37)$$

$$= \sum_{j \in [k-1]: n_u^t < j \leq n_u^t + n_m^t} v_{(j)}$$

which is exactly the desired identity in (36).

Consider the second case that $\delta_i = \text{dn}$. Then we have $n_u^t + n_m^t = n_u^{t-1} + n_m^{t-1} + 1$ and $n_u^t = n_u^{t-1}$. Thus, we have

$$S^t = S^{t-1} + v_{(n_u^t + n_m^t)} \quad \therefore \text{Subroutine 4-line 6,}$$

$$= \sum_{j \in [k-1]: n_u^{t-1} + 1 < j \leq n_u^{t-1} + n_m^{t-1} + 1} v_{(j)} \quad \therefore (37)$$

$$= \sum_{j \in [k-1]: n_u^t < j \leq n_u^t + n_m^t} v_{(j)}$$

which, again, is exactly the desired identity in (36).

$\square$

**Lemma A.15.** Let $\gamma$ be as in Lemma A.9. Then we have

$$\delta = \check{b}^{(n_u^\gamma, n_m^\gamma)} = \text{clip}_C(v - \gamma^{(n_u^\gamma, n_m^\gamma)}) \cdot 1.$$

**Proof.** It suffices to prove that $\gamma = \check{\gamma}^{(n_u^\gamma, n_m^\gamma)}$. To this end, let $\gamma \in [k-1]$. If $i \in I_m^\gamma$, then $\delta_i = v_i - \gamma_i$. If $i \in I_u^\gamma$, then $\delta_i = C$. Otherwise, $\delta_i = 0$. Thus

$$\check{\gamma} = \langle \delta \rangle = C \cdot \check{n_u^\gamma} + S^{(n_u^\gamma, n_m^\gamma)} - \gamma \cdot n_m^\gamma$$

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

$$\check{\gamma} = (C \cdot \check{n_u^\gamma} + S^{(n_u^\gamma, n_m^\gamma)}) / (n_m^\gamma + 1) = \gamma^{(n_u^\gamma, n_m^\gamma)},$$

as desired.

$\square$

**A.5.3. Checking the KKT conditions**

**Lemma A.16.** Let $(n_m, n_u) \in \|k\|^2$. To simplify notation, let $\delta := \check{b}^{(n_u, n_m)}$, $\gamma := \check{\gamma}^{(n_u, n_m)}$. We have $\Omega \delta = \gamma \mathbb{1}$ and for all $i \in [k-1]$ that

$$[(I + \Omega) \delta]_{(i)} = \begin{cases} C + \gamma : i \leq n_u \\ v_{(i)} : n_u < i \leq n_u + n_m \\ \gamma : n_u + n_m < i. \end{cases}$$

(38)

Furthermore, $\delta$ satisfies the KKT conditions (29) to (34) if and only if, for all $i \in [k-1]$,

$$v_{(i)} \begin{cases} \geq C + \gamma : i \leq n_u \\ \in [\gamma, C + \gamma] : n_u < i \leq n_u + n_m \\ \leq \gamma : n_u + n_m < i. \end{cases}$$

(39)
Proof. First, we prove $Ob = \gamma 1$ which is equivalent to $[Ob]_j = \gamma$ for all $j \in [k - 1]$. This is a straightforward calculation:

$$[Ob]_j = 1'b = \sum_{i \in [k-1]} b_{(i)}$$

$$= \sum_{i \in [k-1]: i \leq n_u} b_{(i)} + \sum_{i \in [k-1]: n_u < i \leq n_u + n_m} b_{(i)} + \sum_{i \in [k-1]: n_u + n_m < i} b_{(i)}$$

$$= \sum_{i \in [k-1]: i \leq n_u} C + \sum_{i \in [k-1]: n_u < i \leq n_u + n_m} v_{(i)} - \gamma$$

$$= C \cdot n_u + S(n'_m n'_u) - n_m \gamma$$

$$= \gamma.$$ 

Since $[(I + O)b]_i = [O]_i + [Ob]_i$, the identity (38) now follows immediately.

Next, we prove the “Furthermore” part. First, we prove the “only if” direction. By assumption, we have $b = \tilde{b}$ and so $\gamma = \tilde{\gamma}$. Furthermore, from Lemma A.9 we have $b = \text{clip}_C(v - \tilde{\gamma}1)$ and so $b = \text{clip}_C(v - \gamma1)$. To proceed, recall that by construction, we have

$$b_{(i)} = \begin{cases} C & : i \leq n_u \\ v - \gamma & : n_u < i \leq n_u + n_m \\ 0 & : n_u + n_m < i \end{cases}$$

Thus, if $i \leq n_u$, then $C = b_{(i)} = [\text{clip}_C(v - \tilde{\gamma}1)]_{(i)}$ implies that $v_{(i)} - \gamma \geq C$. If $n_u < i \leq n_u + n_m$, then $b_{(i)} = v_{(i)} - \gamma$. Since $b_{(j)} \in [0, C]$ for all $j \in [k - 1]$, we have in particular that $v_{(i)} - \gamma \in [0, C]$. Finally, if $n_u + n_m < i$, then $0 = b_{(i)} = [\text{clip}_C(v - \tilde{\gamma}1)]_{(i)}$ implies that $v - \gamma \leq 0$. In summary,

$$v_{(i)} - \gamma \begin{cases} \geq C & : i \leq n_u \\ \in [0, C] & : n_u < i \leq n_u + n_m \\ \leq 0 & : n_u + n_m < i. \end{cases}$$

Note that the above identity immediately implies (39).

Next, we prove the “if” direction. Using (38) and (39), we have

$$[(I + O)b]_{(i)} - v_{(i)} \begin{cases} \leq 0 & : i \leq n_u \\ = 0 & : n_u < i \leq n_u + n_m \\ \geq 0 & : n_u + n_m < i. \end{cases}$$

For each $i \in [k - 1]$, define $\lambda_i, \mu_i \in \mathbb{R}$ where

$$\lambda_{(i)} = \begin{cases} -[(I + O)b]_{(i)} - v_{(i)} & : i \leq n_u \\ 0 & : n_u < i \leq n_u + n_m \\ 0 & : n_u + n_m < i \end{cases}$$

and

$$\mu_{(i)} = \begin{cases} 0 & : i \leq n_u \\ 0 & : n_u < i \leq n_u + n_m \\ [(I + O)b]_{(i)} - v_{(i)} & : n_u + n_m < i. \end{cases}$$

It is straightforward to verify that all of (29) to (34) are satisfied for all $i \in [k - 1]$, i.e., the KKT conditions hold at $b$. 

Recall that we use indices with angle brackets $\langle 1 \rangle, \langle 2 \rangle, \ldots, \langle k - 1 \rangle$ to denote a fixed permutation of $[k - 1]$ such that

$$v_{(1)} \geq v_{(2)} \geq \cdots \geq v_{(k - 1)}.$$
Corollary A.17. Let \( t \in [\ell] \) and \( \hat{b} \) be the unique global minimum of the optimization (4). Then \( \hat{b}^t = \hat{b} \) if and only if KKT\_{\text{cond}}() returns true during the \( t \)-th iteration of Algorithm 2.

Proof. First, by Lemma A.9 we have \( \hat{b}^t = \hat{b} \) if and only if \( \hat{b}^t \) satisfies the KKT conditions (29) to (34). From Lemma A.14, we have \( \hat{b}(n^u_m,n^t) = \hat{b}^t \) and \( \hat{\gamma}(n^u_m,n^t) = \hat{\gamma}^t \). To simplify notation, let \( \gamma = \hat{\gamma}(n^u_m,n^t) \). By Lemma A.16, \( \hat{b}(n^u_m,n^t) \) satisfies the KKT conditions (29) to (34) if and only if the following are true:

\[
\begin{align*}
    v_{(i)} & \geq C + \gamma : i \leq n^t_u \\
    \in [\gamma, C + \gamma] : n^t_u < i \leq n^t_u + n^t_m \\
    \leq \gamma & : n^t_u + n^t_m < i.
\end{align*}
\]

Since \( v_{(1)} \geq v_{(2)} \geq \cdots \), the above system of inequalities holds for all \( i \in [k-1] \) if and only if

\[
\begin{align*}
    C + \gamma & \leq v_{(n^t)} \quad : \text{if } n^t_u > 0, \\
    \gamma & \leq v_{(n^t_u + n^t_u)} \quad \text{and } v_{(n^t_u + 1)} = C + \gamma \quad : \text{if } n^t_m > 0, \\
    v_{(n^t_u + n^t_m + 1)} & \leq \gamma \quad : \text{if } n^t_u + n^t_m < k - 1.
\end{align*}
\]

Note that the above system holds if and only if KKT\_{\text{cond}}() returns true. \( \square \)

A.5.4. The variables \( n^t_u \) and \( n^t_u \)

Definition A.18. Define the set \( \text{vals}^+ = \{(v_j, d_j, n, j) : v_j > 0, j = 1, \ldots, k - 1\} \cup \{(v_j - C, \text{up}, j) : v_j > C, j = 1, \ldots, k - 1\} \). Sort the set \( \text{vals}^+ = \{\gamma_1, \delta_1, j_1, \ldots, \gamma_\ell, \delta_\ell, j_\ell\} \) so that the ordering of \( \{\gamma_1, \delta_1, \ldots, \gamma_\ell, \delta_\ell\} \) is identical to \( \text{vals} \) from Subroutine 3-line 2.

To illustrate the definitions, we consider the following running example

\[
\begin{align*}
    \langle j \rangle & = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\} \\
    \langle v(j) \rangle & = \{1.8, 1.4, 1.4, 1.4, 1.2, 0.7, 0.4, 0.4, 0.1, -0.2\} \\
    t & = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 \\
    \gamma_t & = 1.8, 1.4, 1.4, 1.4, 1.2, 0.8, 0.7, 0.4, 0.4, 0.4, 0.4, 0.2, 0.1 \\
    \delta_t & = \text{dn, dn, dn, dn, dn, up, \text{up, up, up, up, up, up, up, up}} \\
\end{align*}
\]

Definition A.19. Define

\[
(u(j)) := \max\{\tau \in [\ell] : v(j) - C = \gamma_\tau\}, \quad \text{and} \quad (d(j)) := \max\{\tau \in [\ell] : v(j) = \gamma_\tau\},
\]

where \( \max\emptyset = \ell + 1 \).

Below, we compute \( d(3) \), \( d(6) \) and \( u(3) \) for our running example.

\[
\begin{align*}
    t & = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 \\
    (d(3)) & = \text{dn, dn, dn, dn, dn, up, \text{up, up, up, up, up, up}} \\
    (d(6)) & = \text{dn, dn, dn, dn, dn, up, \text{up, up, up, up, up, up}} \\
    (u(3)) & = \text{dn, dn, dn, dn, dn, up, \text{up, up, up, up, up, up}} \\
\end{align*}
\]

Definition A.20. Define the following sets

\[
\begin{align*}
    \text{crit}_1(v) & = \{\tau \in [\ell] : \gamma_\tau > \gamma_{\tau+1}\} \\
    \text{crit}_2(v) & = \{\tau \in [\ell] : \gamma_\tau = \gamma_{\tau+1}, \delta_\tau = \text{up, } \delta_{\tau+1} = \text{dn}\}
\end{align*}
\]

where \( \gamma_{\ell+1} = 0 \).
Below, we illustrate the definition in our running example. The arrows \( \downarrow \) and \( \downarrow \) point to elements of \( \text{crit}_1(v) \) and \( \text{crit}_2(v) \), respectively.

\[
\begin{array}{ccccccccccccccc}
 t & = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
 \gamma_t & = & 1.8 & 1.4 & 1.4 & 1.4 & 1.2 & 0.8 & 0.7 & 0.4 & 0.4 & 0.4 & 0.4 & 0.2 & 0.1 \\
 \delta_t & = & \text{dn} & \text{dn} & \text{dn} & \text{dn} & \text{dn} & \text{up} & \text{dn} & \text{up} & \text{up} & \text{up} & \text{dn} & \text{dn} & \text{dn} & \text{dn} \\
\end{array}
\]

Later, we will show that Algorithm 2 will halt and output the global optimizer \( \hat{b} \) on or before the \( t \)-th iteration where \( t \in \text{crit}_1(v) \cup \text{crit}_2(v) \).

**Lemma A.21.** Suppose that \( t \in \text{crit}_1(v) \). Then

\[
\#\{j \in [k-1] : d(j) \leq t\} = \#\{\tau \in [t] : \delta_\tau = \text{dn}\}, \quad \text{and} \quad \#\{j \in [k-1] : u(j) \leq t\} = \#\{\tau \in [t] : \delta_\tau = \text{up}\}.
\]

**Proof.** First, we observe that

\[
\#\{\tau \in [t] : \delta_\tau = \text{up}\} = \#\{(\gamma, \delta, j') \in \text{vals}^+ : \delta = \text{up}, \gamma \geq \gamma_t\}
\]

Next, note that \( j \mapsto (\gamma_{d(j)}, \text{up}, (j)) \) is a bijection from \( \{j \in [k-1] : d(j) \leq t\} \) to \( \{(\gamma, \delta, j') \in \text{vals}^+ : \delta = \text{up}, \gamma \geq \gamma_t\} \).

To see this, we view the permutation \((1), (2), \ldots \) as a bijection mapping \((\cdot) : [k-1] \to [k-1] \) given by \( j \mapsto (j) \). Denote by \( \cdot \) the inverse of \((\cdot) \). Then the (two-sided) inverse to \( j \mapsto (\gamma_{d(j)}, \text{up}, (j)) \) is clearly given by \( (\gamma, \text{up}, j') \mapsto j' \).

This proves the first identity of the lemma. The proof of the second identity is completely analogous. \(\square\)

**Lemma A.22.** The functions \( u \) and \( d : [k-1] \to [\ell+1] \) are non-decreasing. Furthermore, for all \( j \in [k-1] \), we have \( u(j) < d(j) \).

**Proof.** Let \( j' < j'' \) be such that \( j' < j'' \). By the sorting, we have \( v_{j'}(j) \geq v_{j''}(j) \). Now, suppose that \( d(j') > d(j'') \), then by construction we have \( \gamma_{d(j')} < \gamma_{d(j'')} \). On the other hand, we have

\[
\gamma_{d(j')} = v_{j'}(j) \geq v_{j''}(j) = \gamma_{d(j'')}
\]

which is a contradiction.

For the “Furthermore” part, suppose the contrary that \( u(j) \geq d(j) \). Then we have \( \gamma_{u(j)} \leq \gamma_{d(j)} \). However, by definition, we have \( \gamma_{d(j)} = v_{j}(j) > v_{j}(j) - C = \gamma_{d(j)} \). This is a contradiction. \(\square\)

**Lemma A.23.** Let \( t \in \text{crit}_1(v) \). Then \( n^t_u = \#\{j \in [k-1] : u(j) \leq t\} \). Furthermore, \( [n^t_u] = \{j \in [k-1] : u(j) \leq t\} \).

Equivalently, for each \( j \in [k-1] \), we have \( j \leq n^t_u \) if and only if \( u(j) \leq t \).

**Proof.** First, we note that

\[
n^t_u = \#\{\tau \in [t] : \delta_\tau = \text{up}\} \quad \text{Subroutine 4-line 2} \\
= \#\{j \in [k-1] : u(j) \leq t\} \quad \text{Lemma A.21}
\]

This proves the first part. For the “Furthermore” part, let \( N := \#\{j \in [k-1] : u(j) \leq t\} \). Since \( u \) is monotonic non-decreasing (Lemma A.22), we have \( \{j \in [k-1] : u(j) \leq t\} = [N] \). Since \( N = n^t_u \) by the first part, we are done. \(\square\)

**Lemma A.24.** Let \( \hat{t}, \hat{i} \in \text{crit}_1(v) \) be such that there exists \( t \in [\ell] \) where

\[
n^\hat{t}_m = \#\{j \in [k-1] : d(j) \leq \hat{t}\} - \#\{j \in [k-1] : u(j) \leq \hat{i}\}.
\]

Then \( d(j) \leq \hat{t} \) and \( \hat{t} < u(j) \) if and only if \( n^\hat{t}_u < j \leq n^\hat{t}_u + n^\hat{t}_m \).

\[
(41)
\]
Proof. By Lemma A.23 and (41), we have \( \# \{ j \in [k - 1] : d(j) \leq \ell \} = n_u^t + n_m^t \). By Lemma A.22, \( d \) is monotonic non-decreasing and so \( [n_u^t + n_m^t] = \{ j \in [k - 1] : d(j) \leq \ell \} \). Now,
\[
\{ j \in [k - 1] : d(j) \leq \ell, \ell < u(j) \} \\
= \{ j \in [k - 1] : d(j) \leq \ell \} \cap \{ j \in [k - 1] : \ell < u(j) \} \\
= \{ j \in [k - 1] : d(j) \leq \ell \} \setminus \{ j \in [k - 1] : u(j) \leq \ell \} \\
= [n_u^t + n_m^t] \setminus [n_u^t],
\]
where in the last equality, we used Lemma A.23.

\( \square \)

**Corollary A.25.** Let \( t \in \text{crit}_1(v) \). Then \( d(j) \leq t \) and \( t < u(j) \) if and only if \( n_u^t < j \leq n_u^t + n_m^t \).

**Proof.** We apply Lemma A.24 with \( t = \ell = \ell \), which requires checking that
\[
n_m^t = \# \{ j \in [k - 1] : d(j) \leq t \} - \# \{ j \in [k - 1] : u(j) \leq t \}.
\]
This is true because from Subroutine 4-line 2 and 5, we have
\[
n_m^t = \# \{ \tau \in [t] : \delta_\tau = \text{dn} \} - \# \{ \tau \in [t] : \delta_\tau = \text{up} \}.
\]
Applying Lemma A.21, we are done.

\( \square \)

**Lemma A.26.** Let \( t \in \text{crit}_1(v) \). Let \( \varepsilon > 0 \) be such that for all \( \tau, \tau' \in \text{crit}_1(v) \) where \( \tau' < \tau \), we have \( \gamma_{\tau'} - \varepsilon > \gamma_{\tau} \). Then \( (n_m^t, n_u^t) = (n_m^{\gamma_{\tau} - \varepsilon}, n_u^{\gamma_{\tau} - \varepsilon}) \).

**Proof.** We claim that
\[
v(j) - \gamma_t + \varepsilon \begin{cases} 
< 0 & : t < d(j) \\
\in (0, C) & : \ell \leq t < u(j) \\
> C & : u(j) \leq t.
\end{cases} (42)
\]
To prove the \( t < d(j) \) case of (42), we have
\[
v(j) - \gamma_t + \varepsilon = \gamma_{d(j)} - \gamma_t + \varepsilon \quad : (40)
\]
\[
< -\varepsilon + \varepsilon = 0 \quad : t < d(j) \text{ implies that } \gamma_t - \varepsilon > \gamma_{d(j)}.
\]
To prove the \( d(j) \leq t < u(j) \) case of (42), we note that
\[
v(j) - \gamma_t + \varepsilon = \gamma_{d(j)} - \gamma_t + \varepsilon \geq \varepsilon > 0 \quad : (40)
\]
\[
\text{ implies } \gamma_{d(j)} \geq \gamma_t.
\]
For the other inequality,
\[
v(j) - \gamma_t + \varepsilon = \gamma_{u(j)} + C - \gamma_t + \varepsilon \quad : (40)
\]
\[
< -\varepsilon + C + \varepsilon = C \quad : t < u(j) \text{ implies } \gamma_t - \varepsilon > \gamma_{u(j)}.
\]
Finally, we prove the \( u(j) \leq t \) case of (42). Note that
\[
v(j) - \gamma_t + \varepsilon = \gamma_{u(j)} + C - \gamma_t + \varepsilon \geq C + \varepsilon > C \quad : (40)
\]
\[
\text{ implies } \gamma_{u(j)} \geq \gamma_t.
\]
Thus, we have proven (42). By Lemma A.23 and Corollary A.25, (42) can be rewritten as
\[
v(j) - \gamma_t + \varepsilon \begin{cases} 
< 0 & : n_u^t + n_m^t < j, \\
\in (0, C) & : n_u^t < j \leq n_u^t + n_m^t, \\
> C & : j \leq n_u^t.
\end{cases} (43)
\]
Thus, we have \( I_{n_u^t - \varepsilon} = \{ 1, \ldots, n_u^t \} \) and \( I_{n_m^t - \varepsilon} = \{ n_u^t + 1, \ldots, n_u^t + n_m^t \} \). By the definitions of \( n_u^t - \varepsilon \) and \( n_m^t - \varepsilon \), we are done.

\( \square \)
Lemma A.27. Let \( t \in \text{crit}_2(v) \). Then \((n^t_m, n^t_u) = (n^t_m, n^t_u^{\gamma_t})\).

**Proof.** Let \( \hat{t} \in \text{crit}_1(v) \) be such that \( \gamma_{\hat{t}} = \gamma_t \), and \( \hat{t} = \max\{\tau \in \text{crit}_1(v) : \gamma_{\tau} > \gamma_t\} \). We claim that

\[
\begin{align*}
v(j) - \gamma_t & \begin{cases} 
\leq 0 & : \hat{t} < d(j), \\
\in (0, C) & : d(j) \leq \hat{t}, \hat{t} < u(j), \\
\geq C & : u(j) \leq \hat{t}.
\end{cases}
\end{align*}
\]

(44)

Note that by definition, we have \( \gamma_{\hat{t}} > \gamma_t \), which implies that \( \hat{t} < \hat{t} \).

Consider the first case of (44) that \( \hat{t} < d(j) \). See the running example Figure 3. We have by construction that \( v(j) = \gamma_{d(j)} \) and so \( v(j) - \gamma_t = \gamma_{d(j)} - \gamma_t \leq 0 \).

Next, consider the case when \( d(j) \leq \hat{t} \) and \( \hat{t} < u(j) \). Thus,

\[
v(j) - \gamma_t > v(j) - \gamma_t \quad \because \gamma_t > \gamma_{\hat{t}}
\]

\[
= \gamma_{d(j)} - \gamma_{\hat{t}} \quad \because \text{definition of } d(j)
\]

\[
\geq 0 \quad \because d(j) \leq \hat{t} \implies \gamma_{d(j)} \geq \gamma_{\hat{t}}.
\]

On the other hand

\[
v(j) - \gamma_t = \gamma_{u(j)} + C - \gamma_t \quad \because \text{definition of } u(j)
\]

\[
< C \quad \because \hat{t} < u(j) \implies \gamma_{\hat{t}} > \gamma_{u(j)}
\]

Thus, we’ve shown that in the second case, we have \( v(j) - \gamma_t \in (0, C) \).

We consider the final case that \( u(j) \leq \hat{t} \). We have

\[
v(j) - \gamma_t = \gamma_{u(j)} + C - \gamma_t \quad \because \text{definition of } t
\]

\[
\geq C \quad \because u(j) \leq \hat{t} \implies \gamma_{u(j)} \geq \gamma_{\hat{t}}.
\]

Thus, we have proven (44).

Next, we claim that \( t, \hat{t}, \hat{t} \) satisfy the condition (41) of Lemma A.24, i.e.,

\[
n^t_m = \#\{j \in [k-1] : d(j) \leq \hat{t}\} - \#\{j \in [k-1] : u(j) \leq \hat{t}\}.
\]

To this end, we first recall that

\[
n^t_m = \#\{\tau \in [t] : \delta_{\tau} = \text{dn}\} - \#\{\tau \in [t] : \delta_{\tau} = \text{up}\}.
\]

By assumption on \( t \), for all \( \tau \) such that \( \hat{t} < \tau \leq t \), we have \( \delta_{\tau} = \text{up} \). Thus,

\[
\#\{\tau \in [t] : \delta_{\tau} = \text{dn}\} = \#\{\tau \in [\hat{t}] : \delta_{\tau} = \text{dn}\} = \#\{j \in [k-1] : d(j) \leq \hat{t}\}
\]

where for the last equality, we used Lemma A.21. Similarly, for all \( \tau \) such that \( t < \tau \leq \hat{t} \), we have \( \delta_{\tau} = \text{dn} \). Thus, we get that analogous result

\[
n^t_u = \#\{\tau \in [t] : \delta_{\tau} = \text{up}\} = \#\{\tau \in [\hat{t}] : \delta_{\tau} = \text{up}\} = \#\{j \in [k-1] : u(j) \leq \hat{t}\} = n^\hat{t}_u.
\]

(45)
Thus, we have verified the condition (41) of Lemma A.24. Now, applying Lemma A.23 and Lemma A.24, we get

\[
v_{(j)} - \gamma_t \begin{cases} 
\leq 0 & : n^i_t + n^m_m < j, \\
\in (0, C) & : n^i_t < j \leq n^i_t + n^m_m \\
\geq C & : j \leq n^i_t.
\end{cases}
\] (46)

By (45) and that \(\gamma_t = \gamma_t\), the above reduces to

\[
v_{(j)} - \gamma_t \begin{cases} 
\leq 0 & : n^i_t + n^m_m < j, \\
\in (0, C) & : n^i_t < j \leq n^i_t + n^m_m \\
\geq C & : j \leq n^i_t.
\end{cases}
\] (47)

Thus, \(I^u_{n^i_t} = \{1, \ldots, (n^i_t)\}\) and \(I^m_{n^m_m} = \{(n^i_t + 1, \ldots, (n^i_t + n^m_m)\}\). By the definitions of \(n^u_{n^i_t}\) and \(n^m_{n^m_m}\), we are done.

\[\Box\]

A.5.5. Putting it all together

If \(v_{\text{max}} \leq 0\), then Algorithm 2 returns 0.

Otherwise, by Lemma A.9, we have \(\gamma \in (0, v_{\text{max}})\).

**Lemma A.28.** Let \(t \in [\ell]\) be such that \((n^i_m, n^i_u) = (n^s_{m}, n^s_{u})\). Then during the \(t\)-th loop of Algorithm 2 we have \(b = b^t\) and \(\text{KKT-cond()}\) returns true. Consequently, Algorithm 2 returns the optimizer \(b^t\) on or before the \(t\)-th iteration.

**Proof.** We have

\[
\left.\begin{array}{l}
\bar{b} = \bar{b}(n^s_{m}, n^s_{u}) \quad \because \text{Lemma A.15} \\
\bar{b} = \bar{b}(n^i_{m}, n^i_{u}) \quad \because \text{Assumption} \\
\bar{b} = \bar{b}^t \quad \because \text{Lemma A.14}.
\end{array}\right\}
\]

Thus, by Corollary A.17 \(\text{KKT-cond()}\) returns true on the \(t\)-th iteration. This means that Algorithm 2 halts on or before iteration \(t\). Let \(\tau \in [\ell]\) be the iteration where Algorithm 2 halts and outputs \(\bar{b}^\tau\). Then \(\tau \leq t\). Furthermore, by Corollary A.17, \(b = \bar{b}^\tau\), which proves the “Consequently” part of the lemma.

\[\Box\]

By Lemma A.28, it suffices to show that \((n^i_m, n^i_u) = (n^s_{m}, n^s_{u})\) for some \(t \in [\ell]\).

We first consider the case when \(\gamma \neq \gamma_t\) for any \(t \in \text{crit}_1(v)\). Thus, there exists \(t \in \text{crit}_1(v)\) such that \(\gamma_{t+1} < \gamma < \gamma_t\), where we recall that \(\gamma_{t+1} := 0\).

Now, we return to the proof of Theorem 3.4.

\[
(n^i_m, n^i_u) = (n^s_{m} - \varepsilon, n^s_{u} - \varepsilon) \quad \because \text{Lemma A.26} \\
= (n^s_{m}, n^s_{u}) \quad \because \text{Lemma A.12}, \text{and that both } \gamma \text{ and } \gamma_t - \varepsilon \in (\gamma_{t+1}, \gamma_t)\.
\]

Thus, Lemma A.28 implies the result of Theorem 3.4 under the assumption that \(\gamma \neq \gamma_t\) for any \(t \in \text{crit}_1(v)\).

Next, we consider when \(\gamma = \gamma_t\) for some \(t \in \text{crit}_1(v)\). There are three possibilities:

1. There does not exist \(j \in [k - 1]\) such that \(v_{(j)} = \gamma_t\).
2. There does not exist \(j \in [k - 1]\) such that \(v_{(j)} - C = \gamma_t\).
3. There exist \(j_1, j_2 \in [k - 1]\) such that \(v_{(j_1)} = \gamma_t\) and \(v_{(j_2)} - C = \gamma_t\).

First, we consider case 1. We claim that

\[
(n^s_{m}, n^s_{u}) = (n^s_{m} - \varepsilon', n^s_{u} - \varepsilon') \quad \text{for all } \varepsilon' > 0 \text{ sufficiently small}. \] (48)
Weston-Watkins SVM subproblem

We first note that \( n_m^u = n_m^u - \varepsilon' \) for all \( \varepsilon' > 0 \) sufficiently small. To see this, let \( i \in [k - 1] \) be arbitrary. Note that

\[
i \in I_m^u \iff v_i - \gamma_t \geq C \iff v_i - \gamma_t + \varepsilon' \geq C, \forall \varepsilon' > 0, \text{ sufficiently small} \\
\iff i \in I_m^{u-\varepsilon'}, \forall \varepsilon' > 0, \text{ sufficiently small.}
\]

Next, we show that \( n_m^u = n_m^u - \varepsilon' \) for all \( \varepsilon' > 0 \) sufficiently small. To see this, let \( i \in [k - 1] \) be arbitrary. Note that

\[
i \in I_m^u \iff v_i - \gamma_t \in (0, C) \iff v_i - \gamma_t + \varepsilon' \in (0, C), \forall \varepsilon' > 0, \text{ sufficiently small} \\
\iff i \in I_m^{u-\varepsilon'}, \forall \varepsilon' > 0, \text{ sufficiently small}.
\]

where at “ \( \iff \) ”, we used the fact that \( v_i - \gamma_t \neq 0 \) for any \( i \in [k - 1] \). Thus, we have proven (48). Taking \( \varepsilon' > 0 \) so small so that both (48) and the condition in Lemma A.26 hold, we have

\[
(n_m^t, n_m^u) = (n_m^u - \varepsilon', n_m^u - \varepsilon') = (n_m^u, n_m^u) = (n_m^u, n_m^u).
\]

This proves Theorem 3.4 under case 1.

Next, we consider case 2. We claim that

\[
(n_m^u, n_m^u) = (n_m^u + \varepsilon', n_u^u - \varepsilon') \quad \text{for all } \varepsilon'' > 0 \text{ sufficiently small.} \tag{49}
\]

We first note that \( n_m^u = n_m^u - \varepsilon'' \) for all \( \varepsilon'' > 0 \) sufficiently small. To see this, let \( i \in [k - 1] \) be arbitrary. Note that

\[
i \in I_m^u \iff v_i - \gamma_t \geq C \iff v_i - \gamma_t - \varepsilon'' \geq C, \forall \varepsilon'' > 0, \text{ sufficiently small} \\
\iff i \in I_m^{u-\varepsilon''}, \forall \varepsilon'' > 0, \text{ sufficiently small.}
\]

where at “ \( \iff \) ”, we used the fact that \( v_i - \gamma_t \neq C \) for any \( i \in [k - 1] \). Next, we show that \( n_m^u = n_m^u - \varepsilon'' \) for all \( \varepsilon'' > 0 \) sufficiently small. To see this, let \( i \in [k - 1] \) be arbitrary. Note that

\[
i \in I_m^u \iff v_i - \gamma_t \in (0, C) \iff v_i - \gamma_t - \varepsilon'' \in (0, C), \forall \varepsilon'' > 0, \text{ sufficiently small} \\
\iff i \in I_m^{u+\varepsilon''}, \forall \varepsilon'' > 0, \text{ sufficiently small}.
\]

where again at “ \( \iff \) ”, we used the fact that \( v_i - \gamma_t \neq C \) for any \( i \in [k - 1] \). Thus, we have proven (49). Since \( \gamma_t = \gamma_t \in (0, v_{\text{max}}) \) and \( \gamma_t = \gamma_{\text{max}} \), we have in particular that \( \gamma_t < \gamma_{\text{max}} \). Thus, there exists \( \tau \in \text{crit}_1(v) \) such that \( \tau < t \) and \( \gamma_t < \gamma_{\tau} \). Furthermore, we can choose \( \tau \) such that for all \( \gamma \in (\gamma_t, \gamma_{\tau}) \), \( \gamma \notin \text{crit}_1(v) \). Let \( \varepsilon'' > 0 \) be so small that \( \gamma_t + \varepsilon'', \gamma_t - \varepsilon'' \in (\gamma_t, \gamma_{\tau}) \), and furthermore both (49) and the condition in Lemma A.26 hold. We have

\[
(n_m^t, n_m^u) = (n_m^u - \varepsilon'', n_u^u - \varepsilon'') \quad \text{Lemma A.26} \\
= (n_m^u - \varepsilon'', n_u^u + \varepsilon'') \quad \text{Lemma A.12 and } \gamma_t + \varepsilon'', \gamma_t - \varepsilon'' \in (\gamma_t, \gamma_{\tau}) \\
= (n_m^u, n_u^u) \quad \text{Lemma A.27} \tag{49} \\
= (n_m^u, n_u^u) \quad \text{Assumption.}
\]

This proves Theorem 3.4 under case 2.

Finally, we consider the last case. Under the assumptions, we have \( t \in \text{crit}_2(v) \). Then Lemma A.27 \( (n_m^t, n_m^u) = (n_m^u, n_u^u) = (n_m^u, n_u^u) \). Thus, we have proven Theorem 3.4 under case 3.

\[\square\]

A.6. Experiments

The Walrus solver is available at:

https://github.com/YutongWangUMich/liblinear

The actual implementation is in the file linear.cpp in the class SolverMCSVM_WW.
All code for downloading the datasets used, generating the train/test split, running the experiments and generating the figures are included. See the README.md file for more information.

All experiments are run on a single machine with the following specifications:

Operating system and kernel:
4.15.0-122-generic #124-Ubuntu SMP Thu Oct 15 13:03:05 UTC 2020 x86_64 GNU/Linux

Processor:
Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz

Memory:
31GiB System memory

A.6.1. On Shark’s linear WW-SVM solver

Shark’s linear WW-SVM solver is publicly available in the GitHub repository https://github.com/Shark-ML. Specifically, the C++ code is in Algorithms/QP/QpMcLinear.h in the class QpMcLinearWW. Our reimplementation follows their implementation with two major differences. In our implementations, neither Shark nor Walrus use the shrinking heuristic. Furthermore, we use a stopping criterion based on duality gap, following (Steinwart et al., 2011).

We also remark that Shark solves the following variant of the WW-SVM which is equivalent to ours after a change of variables. Let \( 0 < A \in \mathbb{R} \) be a hyperparameter.

\[
\min_{\mathbf{u} \in \mathbb{R}^{d \times k}} F_A(\mathbf{u}) := \frac{1}{2} \|\mathbf{u}\|_F^2 + A \sum_{i=1}^n \sum_{j \in [k]: j \neq y_i} \text{hinge} \left( (u_{yi}^\prime x_i - u_j^\prime x_i)/2 \right). \tag{50}
\]

Recall the formulation (P) that we consider in this work, which we repeat here:

\[
\min_{\mathbf{w} \in \mathbb{R}^{d \times k}} G_C(\mathbf{w}) := \frac{1}{2} \|\mathbf{w}\|_F^2 + C \sum_{i=1}^n \sum_{j \in [k]: j \neq y_i} \text{hinge}(w_{yi}^\prime x_i - w_j^\prime x_i). \tag{51}
\]

The formulation (50) is used by Weston & Watkins (1999), while the formulation (51) is used by Vapnik (1998). These two formulations are equivalent under the change of variables \( w = \mathbf{u}/2 \) and \( A = 4C \). To see this, note that

\[
G_C(\mathbf{w}) = G_C(\mathbf{u}/2)
\]

\[
= \frac{1}{2} \|\mathbf{u}/2\|_F^2 + C \sum_{i=1}^n \sum_{j \in [k]: j \neq y_i} \text{hinge}((u_{yi}^\prime x_i - u_j^\prime x_i)/2)
\]

\[
= \frac{1}{8} \|\mathbf{u}\|_F^2 + C \sum_{i=1}^n \sum_{j \in [k]: j \neq y_i} \text{hinge}((u_{yi}^\prime x_i - u_j^\prime x_i)/2)
\]

\[
= \frac{1}{4} \left( \frac{1}{2} \|\mathbf{u}\|_F^2 + 4C \sum_{i=1}^n \sum_{j \in [k]: j \neq y_i} \text{hinge}((u_{yi}^\prime x_i - u_j^\prime x_i)/2) \right)
\]

\[
= \frac{1}{4} F_{4C}(\mathbf{u}) = \frac{1}{4} F_A(\mathbf{u}).
\]

Thus, we have proven

**Proposition A.29.** Let \( C > 0 \) and \( \mathbf{u} \in \mathbb{R}^{d \times k} \). Then \( \mathbf{u} \) is a minimizer of \( F_{4C} \) if and only if \( \mathbf{u}/2 \) is a minimizer of \( G_C \).

In our experiments, we use the above proposition to rescale the variant formulation to the standard formulation.
Table 3. Data sets used from the “LIBSVM Data: Classification (Multi-class)” repository. Variables $k$, $n$ and $d$ are, respectively, the number of classes, training samples, and features. The SCALED column indicates whether a scaled version of the dataset is available on the repository. The TEST SET PROVIDED column indicates whether a test set of the dataset is provided on the repository.

| DATA SET | $k$ | $n$   | $d$ | SCALED | TEST SET AVAILABLE |
|----------|-----|-------|-----|--------|-------------------|
| DNA      | 3   | 2,000 | 180 | YES    | YES               |
| SATIMAGE | 6   | 4,435 | 36  | YES    | YES               |
| MNIST    | 10  | 60,000| 780 | YES    | YES               |
| NEWS20   | 20  | 15,935| 62,061| YES   | YES               |
| LETTER   | 26  | 15,000| 16  | YES    | YES               |
| RCV1     | 53  | 15,564| 47,236| NO    | YES               |
| SECTOR   | 105 | 6,412 | 55,197| YES   | YES               |
| ALOI     | 1,000 | 81,000| 128 | YES    | NO                |

A.6.2. DATA SETS

The data sets used are downloaded from the “LIBSVM Data: Classification (Multi-class)” repository:

https://csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html

We use the scaled version of a data set whenever available. For testing accuracy, we use the testing set provided whenever available. The data set ALOI did not have an accompanying test set. Thus, we manually created a test set using methods described in the next paragraph. See Table 3 for a summary.

The original, unsplit ALOI dataset has $k = 1000$ classes, where each class has 108 instances. For creating the test set, we split instances from each class such that first 81 elements are training instances while the last 27 elements are testing instances. This results in a “75% train /25% test” split with training and testing set consisting of 81,000 and 27,000 samples, respectively.

A.6.3. CLASSIFICATION ACCURACY RESULTS

For both algorithms, we use the same stopping criterion: after the first iteration $t$ such that $D_t^i < \delta \cdot D_1^i$. The results are reported in Table 5 and Table 6 where $\delta = 0.009$ and $\delta = 0.0009$, respectively. The highest testing accuracies are in bold.

Note that going from Table 5 to Table 6, the stopping criterion becomes more stringent. The choice of hyperparameters achieving the highest testing accuracy are essentially unchanged. Thus, for hyperparameter tuning, it suffices to use the more lenient stopping criterion with the larger $\delta$. 
Table 4. Accuracies under the stopping criterion $DG_t^* < \delta \cdot DG_1^*$ with $\delta = 0.09$

| log$_2$($C$) | -6 | -5 | -4 | -3 | -2 | -1 | 0  | 1  | 2  | 3  |
|--------------|----|----|----|----|----|----|----|----|----|----|
| DNA          | 94.60 | 94.69 | 94.52 | 94.44 | 93.59 | 92.92 | 93.09 | 92.83 | 92.50 | 92.83 |
| SATIMAGE     | 81.95 | 82.45 | 82.85 | 83.75 | 83.55 | **84.10** | 83.95 | 83.30 | 83.95 | 84.00 |
| MNIST        | 92.01 | **92.16** | 91.97 | 92.15 | 91.92 | 91.76 | 91.62 | 91.66 | 91.70 | 91.58 |
| NEWS20       | 82.24 | 83.17 | 84.20 | 84.85 | **85.45** | 85.15 | 85.07 | 84.30 | 84.40 | 83.90 |
| LETTER       | 69.62 | **71.46** | 70.92 | 69.82 | 69.72 | 70.74 | 70.50 | 70.74 | 71.00 | 69.22 |
| RCV1         | 87.23 | 87.93 | 88.46 | **88.79** | 88.78 | 88.68 | 88.51 | 88.29 | 88.19 | 88.09 |
| SECTOR       | 93.08 | 93.33 | 93.64 | 93.92 | **94.20** | 94.17 | **94.20** | 94.08 | 94.14 | 94.14 |
| ALOI         | 86.81 | 87.49 | 88.22 | 88.99 | 89.53 | 89.71 | **89.84** | 89.53 | 89.06 | 88.21 |

Table 5. Accuracies under the stopping criterion $DG_t^* < \delta \cdot DG_1^*$ with $\delta = 0.009$

| log$_2$($C$) | -6 | -5 | -4 | -3 | -2 | -1 | 0  | 1  | 2  | 3  |
|--------------|----|----|----|----|----|----|----|----|----|----|
| DNA          | 94.77 | 94.77 | **94.94** | 94.69 | 93.59 | 93.09 | 92.24 | 92.24 | 92.16 | 92.16 |
| SATIMAGE     | 82.35 | 82.50 | 82.95 | 83.55 | 83.55 | 84.10 | **84.35** | 84.20 | 84.05 | 84.25 |
| MNIST        | 92.34 | 92.28 | **92.41** | 92.37 | 92.26 | 92.13 | 92.12 | 91.98 | 91.94 | 91.70 |
| NEWS20       | 82.29 | 83.35 | 84.15 | 85.02 | **85.45** | 85.30 | 84.97 | 84.40 | 84.12 | 84.07 |
| LETTER       | 69.98 | 71.02 | **71.74** | 71.52 | 71.36 | 71.46 | 71.20 | 71.56 | 71.44 | 70.74 |
| RCV1         | 87.24 | 87.96 | 88.46 | 88.76 | **88.80** | 88.70 | 88.48 | 88.25 | 88.15 | 88.03 |
| SECTOR       | 93.14 | 93.36 | 93.64 | 93.95 | 94.04 | **94.08** | 94.04 | 93.98 | 93.92 | 93.92 |
| ALOI         | 86.30 | 87.21 | 88.20 | 89.00 | 89.34 | 89.63 | 89.99 | **90.18** | 89.78 | 89.80 |

Table 6. Accuracies under the stopping criterion $DG_t^* < \delta \cdot DG_1^*$ with $\delta = 0.0009$

| log$_2$($C$) | -6 | -5 | -4 | -3 | -2 | -1 | 0  | 1  | 2  | 3  |
|--------------|----|----|----|----|----|----|----|----|----|----|
| DNA          | 94.77 | 94.69 | **95.11** | 94.77 | 93.76 | 93.34 | 92.41 | 92.24 | 92.24 | 92.24 |
| SATIMAGE     | 82.35 | 82.65 | 83.20 | 83.65 | 83.80 | 84.10 | **84.20** | 84.10 | 84.15 | 84.10 |
| MNIST        | 92.28 | 92.38 | **92.43** | 92.24 | 92.21 | 92.13 | 92.16 | 91.92 | 91.79 | 91.65 |
| NEWS20       | 82.27 | 83.45 | 84.00 | 85.00 | **85.40** | 85.22 | 85.02 | 84.52 | 84.10 | 83.97 |
| LETTER       | 70.04 | 71.28 | **71.70** | 71.66 | 71.48 | 71.30 | 71.30 | 71.02 | 71.22 | 71.22 |
| RCV1         | 87.23 | 87.98 | 88.46 | 88.76 | **88.79** | 88.69 | 88.48 | 88.25 | 88.12 | 88.02 |
| SECTOR       | 93.20 | 93.39 | 93.64 | 93.92 | 94.01 | **94.04** | **94.04** | 94.01 | 93.95 | 93.83 |
| ALOI         | 86.17 | 87.01 | 87.99 | 88.66 | 89.04 | 89.46 | 89.64 | **89.70** | 89.69 | 89.51 |
\textbf{Weston-Watkins SVM subproblem}

Table 7. Accuracies under the stopping criterion $DG_t^\ast < \delta \cdot DG_0$ with $\delta = 0.09$ (first row in each cell), $= 0.009$ (second row) and $= 0.0009$ (third row).

| $\log_2(C)$ | -6 | -5 | -4 | -3 | -2 | -1 | 0  | 1  | 2  | 3  |
|------------|----|----|----|----|----|----|----|----|----|----|
| DATA SET   |    |    |    |    |    |    |    |    |    |    |
| DNA ( $\delta = 0.09$) | 94.60 | 94.69 | 94.52 | 94.44 | 93.59 | 92.92 | 93.09 | 92.83 | 92.50 | 92.83 |
| $\delta = 0.009$ | 94.77 | 94.77 | 94.94 | 94.69 | 93.59 | 93.09 | 92.24 | 92.24 | 92.16 | 92.16 |
| $\delta = 0.0009$ | 94.77 | 94.69 | 95.11 | 94.77 | 93.76 | 93.34 | 92.41 | 92.24 | 92.24 | 92.24 |
| SATIMAGE   | 81.95 | 82.45 | 82.85 | 83.75 | 83.55 | 84.10 | 83.95 | 83.30 | 83.95 | 84.00 |
|            | 82.35 | 82.50 | 82.95 | 83.55 | 83.55 | 84.10 | 84.35 | 84.20 | 84.05 | 84.25 |
|            | 82.35 | 82.65 | 83.20 | 83.65 | 83.80 | 84.10 | 84.20 | 84.10 | 84.15 | 84.10 |
| MNIST      | 92.01 | 92.16 | 91.97 | 92.15 | 91.76 | 91.62 | 91.70 | 91.58 | 91.66 | 91.70 |
|            | 92.34 | 92.28 | 92.41 | 92.37 | 92.24 | 92.24 | 91.94 | 91.70 | 91.94 | 91.70 |
|            | 92.28 | 92.38 | 92.43 | 92.24 | 92.13 | 92.13 | 91.92 | 91.79 | 91.79 | 91.65 |
| NEWS20     | 82.24 | 83.17 | 84.20 | 84.85 | 85.45 | 85.15 | 85.07 | 84.30 | 84.40 | 83.90 |
|            | 82.29 | 83.35 | 84.15 | 85.02 | 85.45 | 85.30 | 84.97 | 84.40 | 84.12 | 84.07 |
|            | 82.27 | 83.45 | 84.00 | 85.00 | 85.40 | 85.22 | 85.02 | 84.52 | 84.10 | 83.97 |
| LETTER     | 69.62 | 71.46 | 70.92 | 69.82 | 69.72 | 70.74 | 70.50 | 70.74 | 71.00 | 69.22 |
|            | 69.98 | 71.02 | 71.74 | 71.52 | 71.36 | 71.46 | 71.20 | 71.56 | 71.44 | 70.74 |
|            | 70.04 | 71.28 | 71.70 | 71.66 | 71.48 | 71.30 | 71.26 | 71.30 | 71.02 | 71.22 |
| RCV1       | 87.23 | 87.93 | 88.46 | 88.79 | 88.78 | 88.68 | 88.51 | 88.29 | 88.19 | 88.09 |
|            | 87.24 | 87.96 | 88.46 | 88.76 | 88.80 | 88.70 | 88.48 | 88.25 | 88.15 | 88.03 |
|            | 87.23 | 87.98 | 88.46 | 88.76 | 88.79 | 88.69 | 88.48 | 88.25 | 88.12 | 88.02 |
| SECTOR     | 93.08 | 93.33 | 93.64 | 93.92 | 94.20 | 94.17 | 94.20 | 94.08 | 94.14 | 94.14 |
|            | 93.14 | 93.36 | 93.64 | 93.95 | 94.04 | 94.08 | 94.08 | 94.08 | 93.98 | 93.92 |
|            | 93.20 | 93.39 | 93.64 | 93.92 | 94.01 | 94.04 | 94.04 | 94.01 | 93.95 | 93.83 |
| ALOI       | 86.81 | 87.49 | 88.22 | 88.99 | 89.53 | 89.71 | 89.84 | 89.53 | 89.06 | 88.21 |
|            | 86.30 | 87.21 | 88.20 | 89.00 | 89.34 | 89.63 | 89.99 | 90.18 | 89.78 | 89.80 |
|            | 86.17 | 87.01 | 87.99 | 88.66 | 89.04 | 89.46 | 89.64 | 89.70 | 89.69 | 89.51 |

A.6.4. Comparison with Convex Program Solvers

For solving (4), we compare the speed of Walrus (Algorithm 2) versus the general-purpose, commercial convex program (CP) solver MOSEK. We generate random instances of the subproblem (4) by randomly sampling $v$. The runtime results of Walrus and the CP solver are shown in Table 8 and Table 9, where each entry is the average over 10 random instances.

Table 8. Runtime in seconds for solving random instances of the problem (4). The parameter $C = 1$ is fixed while $k$ varies.

| $\log_2(k - 1)$ | 2     | 4     | 6     | 8     | 10    | 12    |
|----------------|-------|-------|-------|-------|-------|-------|
| WALRUS         | 0.0009| 0.0001| 0.0001| 0.0001| 0.0002| 0.0005|
| CP SOLVER      | 0.1052| 0.0708| 0.0705| 0.1082| 0.5721| 12.6057|

Table 9. Runtime in seconds for solving random instances of the problem (4). The parameter $k = 2^8 + 1$ is fixed while $C$ varies.

| $\log_{10}(C)$ | -3   | -2   | -1   | 0    | 1    | 2    | 3    |
|----------------|------|------|------|------|------|------|------|
| WALRUS         | 0.0004| 0.0001| 0.0001| 0.0001| 0.0001| 0.0001| 0.0001|
| CP SOLVER      | 0.1177| 0.1044| 0.1046| 0.1005| 0.1050| 0.1127| 0.1206|

As shown here, the analytic solver Walrus is faster than the general-purpose commercial solver by orders of magnitude.