SVM via Saddle Point Optimization: New Bounds and Distributed Algorithms

Lingxiao Huang
École polytechnique fédérale de Lausanne
CH-1015, Lausanne, Switzerland
lingxiao.huang@epfl.ch

Yifei Jin
Tsinghua University
Beijing 100084, China
jin-yf13@mails.tsinghua.edu.cn

Jian Li
Tsinghua University
Beijing 100084, China
Corresponding author: lijian83@mail.tsinghua.edu.cn

Abstract

We study two important SVM variants: hard-margin SVM (for linearly separable cases) and ν-SVM (for linearly non-separable cases). We propose new algorithms from the perspective of saddle point optimization. Our algorithms achieve $(1-\epsilon)$-approximations with running time $\tilde{O}(nd + n\sqrt{d/\epsilon})$ for both variants, where n is the number of points and d is the dimensionality. To the best of our knowledge, the current best algorithm for ν-SVM is based on quadratic programming approach which requires $\Omega(n^2 d)$ time in worst case [Joachims, 1998; Platt, 1999]. In the paper, we provide the first nearly linear time algorithm for ν-SVM. The current best algorithm for hard margin SVM achieved by Gilbert algorithm [Gärtner and Jaggi, 2009] requires $O(nd/\epsilon)$ time. Our algorithm improves the running time by a factor of $\sqrt{d}/\sqrt{\epsilon}$. Moreover, our algorithms can be implemented in the distributed settings naturally. We prove that our algorithms require $\tilde{O}(k(d+\sqrt{d/\epsilon}))$ communication cost, where k is the number of clients, which almost matches the theoretical lower bound. Numerical experiments support our theory and show that our algorithms converge faster on high dimensional, large and dense data sets, as compared to previous methods.

2012 ACM Subject Classification Mathematics of computing → Continuous optimization, Computing methodologies → Support vector machines

Keywords and phrases ν-SVM, hard-margin SVM, saddle point optimization, distributed algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.25

Related Version A full version of the paper is available at `https://arxiv.org/abs/1705.07252`.

Acknowledgements This research is supported in part by the National Basic Research Program of China Grant 2015CB358700, the National Natural Science Foundation of China Grant 61772297, 61632016, 61761146003, and a grant from Microsoft Research Asia.

1 Introduction

Support Vector Machine (SVM) is widely used for classification in numerous applications such as text categorization, image classification, and hand-written characters recognition.

In this paper, we focus on binary classification. If two classes of points which are linearly separable, one can use the hard-margin SVM [7, 9], which is to find a hyperplane that separate two classes of points and the margin is maximized. If the data is not linearly separable, several popular SVM variants have been proposed, such as l_2-SVM, C-SVM and ν-SVM (see e.g., the summary in [15]). The main difference among these variants is that they use different penalty loss functions for the misclassified points. l_2-SVM, as the name implied, uses the l_2 penalty loss. C-SVM and ν-SVM are two well-known SVM variants using l_1-loss. C-SVM uses the l_1-loss with penalty coefficient $C \in [0, \infty)$ [43]. On the other hand, ν-SVM reformulates C-SVM through taking a new regularization parameter $\nu \in (0, 1]$ [35]. However, given a C-SVM formulation, it is not easy to compute the regularization parameter ν and obtain an equivalent ν-SVM. Because the equivalence is based on some hard-to-compute constant. Compared to C-SVM, the parameter ν in ν-SVM has a more clear geometric interpretation: the objective is to minimize the distance between two reduced polytopes defined based on ν [10]. However, the best known algorithm for ν-SVM is much worse than that for C-SVM in practice (see below).

In general, SVMs can be formulated as convex quadratic programs and solved by quadratic programs in $O(n^2 d)$ time [21, 34]. However, better algorithms exists for some SVM variants, which we briefly discuss below.

For hard-margin SVM, [15] showed that Gilbert algorithm [16] achieves a $(1 - \epsilon)$-approximation with $O(nd/\epsilon\beta^2)$ running time where β is the ratio of the minimum distance to the maximum one among the points. l_2-SVM and C-SVM have been studied extensively and current best algorithms runs in time linear in the number n of data points [2, 6, 11, 13, 36]. Moreover, if the parameter C is sufficiently small, e.g., $C = \Theta(1/n)$, then C-SVM can be solved in $\tilde{O}(d/\epsilon)$ time [36], which is independent of n. However, these techniques cannot be extended to ν-SVM directly, mainly because ν-SVM cannot be transformed to single-objective unconstrained optimization problems. Except the traditional quadratic programming approach, there is no better algorithm known with provable guarantee for ν-SVM. Whether ν-SVM can be solved in nearly linear time is still open.

Distributed SVM has also attracted significant attention in recent years. A number of distributed algorithms for SVM have been obtained in the past [12, 17, 28, 30, 41]. Typically, the communication complexity is one of the key performance measurements for distributed algorithms, and has been studied extensively (see [25, 32, 40]). For hard-margin SVM, Liu et al. [26] proposed a distributed algorithm with $O(kd/\epsilon)$ communication cost, where k is the number of the clients. Hence, it is a natural question to ask whether the communication cost of their algorithm can be improved.

1.1 Our Contributions

We summarize our main contributions as follows.

1. **Hard-Margin SVM:** We provide a new $(1 - \epsilon)$-approximation algorithm with running time $\tilde{O}(nd + n\sqrt{d}/\sqrt{\epsilon\beta})$, where β is the ratio of the minimum distance to the maximum one among the points (see Theorem 9).[1] Compared to Gilbert algorithm [15], our algorithm

[1] $\tilde{O}$ notation hides logarithm factors such as $\log(n)$, $\log(\beta)$ and $\log(1/\epsilon)$.

improves the running time by a factor of $\sqrt{d}/\sqrt{\epsilon}$. First, we regard hard-margin SVM as computing the polytope distance between two classes of points. Then we translate the problem to a saddle point optimization problem using the properties of the geometric structures (Lemma 2), and provide an algorithm to solve the saddle point optimization.

2. **ν-SVM:** Then, we extend our algorithm to ν-SVM and design an $\tilde{O}(nd+n\sqrt{d}/\sqrt{\epsilon\beta})$ time algorithm, which is the most important technical contribution of this paper. To the best of our knowledge, it is the first nearly linear time algorithm for ν-SVM. It is known that ν-SVM is equivalent to computing the distance between two reduced polytopes [10, 5]. The obstacle for providing an efficient algorithm based on the reduced polytopes is that the number of vertices in the reduced polytopes may be exponentially large. However, in our framework, we only need to implicitly represent the reduced polytopes. We show that using the similar saddle point optimization framework, together with a new nontrivial projection method, ν-SVM can be solved efficiently in the same time complexity as in the hard-margin case. Compared with the QP-based algorithms in previous work [21, 34], our algorithm significantly improves the running time, by a factor of n.
3. **Distributed SVM:** Finally, we extend our algorithms for both hard-margin SVM and ν-SVM to the distributed setting. We prove that the communication cost of our algorithm is $\tilde{O}(k(d+\sqrt{d/\epsilon}))$, which is almost optimal according to the lower bound provided in [26]. For the hard-margin SVM, compared with the current best algorithm [26] with $O(kd/\epsilon)$ communication cost, our algorithm is more suitable when ϵ is small and d is large. For ν-SVM, our algorithm is the first practical distributed algorithm.

Besides, the numerical experiments support our theoretical bounds. We compare our algorithms with Gilbert Algorithm [15] and NuSVC in scikit-learn [33]. The experiments show that our algorithms converge faster on high dimensional, large and dense data sets. See the full version for the details.

1.2 Other Related Work

For the hard-margin SVM, there is an alternative to Gilbert's method, called the MDM algorithm, originally proposed by [29]. Recently, López and Dorronsoro proved that the rate of convergence of MDM algorithm is $O(n^2 d\log(1/\epsilon))$ [27] which is a linear convergence w.r.t. ϵ, but worse than Gilbert Algorithm w.r.t. n.

Both C-SVM and l_2-SVM have been studied extensively in the literature. Basically, there are three main algorithmic approaches: the primal gradient-based methods [24, 36, 13, 11, 2], dual quadratic programming methods [22, 37, 20] and dual geometry methods [39, 38]. Recently, [2] provided the current best algorithms which achieve $O(nd/\sqrt{\epsilon})$ time for l_2-SVM and $O(nd/\epsilon)$ time for C-SVM.

Sublinear time algorithms for hard-margin SVM and l_2-SVM have been proposed [8, 19]. These algorithms are sublinear w.r.t. nd, (i.e., the size of the input), but have worse dependency on $1/\epsilon$.

The algorithmic framework for saddle point optimization was first developed by Nesterov for structured nonsmooth optimization problem [31]. He only considered the full gradient in the algorithm. Recently, some studies have extended it to the stochastic gradient setting [42, 3]. The most related work is [3], in which the author obtained an $\tilde{O}(nd+n\sqrt{d}/\sqrt{\epsilon})$ algorithm for the minimum enclosing ball problem (MinEB) in Euclidean space, using the saddle point optimization. This result also implies an algorithm for l_2-SVM, by the connection between MinEB and l_2-SVM (see [39, 38, 18]). However, the implied algorithm is not as efficient. Based on [39, 38], the dual of l_2-SVM is equivalent to MinEB by a specific feature mapping.

It maps a d-dimensional point to the $(d+n)$-dimensional space. Thus, after the mapping, it takes quadratic time to solve l_2-SVM. To avoid this mapping, they designed an algorithm called Core Vector Machine (CVM), in which they can solve l_2-SVM by solving $O(1/\epsilon)$ MinEB problems sequentially.

2 Formulate SVM as Saddle Point Optimization

In this section, we formulate both hard-margin SVM and ν-SVM, and show that they can be reduced to saddle point optimizations. All vectors in the paper are all column vectors by default.

▶ **Definition 1** (Hard-margin SVM)**.** Given n points $x_i \in \mathbb{R}^d$ for $1 \le i \le n$, each x_i has a label $y_i \in \{\pm 1\}$. The hard-margin SVM can be formalized as the following quadratic programming [9].

$$\begin{aligned} \min_{w,b} \quad & \tfrac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w^{\mathrm{T}}x_i - b) \ge 1, \quad \forall i \end{aligned} \tag{1}$$

The dual problem of (1) is defined as follows, which is equivalent to finding the minimum distance between the two convex hulls of two classes of points when they are linearly separable [5]. We call the problem the C-Hull problem.

$$\begin{aligned} \min_{\eta,\xi} \quad & \tfrac{1}{2}\|A\eta - B\xi\|^2 \\ \text{s.t.} \quad & \|\eta\|_1 = 1, \|\xi\|_1 = 1. \quad \eta \ge 0, \xi \ge 0. \end{aligned} \tag{2}$$

where A and B are the matrices in which each column represents a vector of a point with label $+1$ or -1 respectively.

Denote the set of points with label $+1$ by $\mathcal{P}$ and the set with label -1 by $\mathcal{Q}$. Let $n_1 = |\mathcal{P}|$ and $n_2 = |\mathcal{Q}|$. Since $\sum_i \eta_i = 1$, we can regard it as a probability distribution among points in P (similarly for Q). We denote Δ_{n_1} to be the set of n_1-dimensional probability vectors over $\mathcal{P}$ and Δ_{n_2} to be that over $\mathcal{Q}$. Then, we prove that the C-Hull problem (2) is equivalent to the following saddle point optimization in Lemma 2.

▶ **Lemma 2.** *Problem C-Hull* (2) *is equivalent to the saddle point optimization* (3)*.*

$$\mathsf{OPT} = \max_{w} \min_{\eta \in \Delta_{n_1}, \xi \in \Delta_{n_2}} w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi - \frac{1}{2}\|w\|^2 \tag{3}$$

Proof. Consider the saddle point optimization (3). First, note that

$$w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi - \frac{1}{2}\|w\|^2 = w^{\mathrm{T}}(A\eta - B\xi) - \frac{1}{2}\|w\|^2$$

The range of the term $(A\eta - B\xi)$ for $\eta \in \Delta_{n_1}, \xi \in \Delta_{n_2}$ is a convex set, denoted by $\mathcal{S}$. Since the convex hulls of $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable, we have $0 \notin \mathcal{S}$. Denote $\phi(w,z) = w^{\mathrm{T}}z - \frac{1}{2}\|w\|^2$ for any $w \in \mathbb{R}^d, z \in \mathcal{S}$. Then (3) is equivalent to $\max_w \min_{z\in\mathcal{S}} \phi(w,z)$. Note that

$$\max_{w} \min_{z\in\mathcal{S}} \phi(w,z) \ge \min_{z\in\mathcal{S}} \phi(\mathbf{0}^d, z) = 0.$$

Thus, we only need to consider those directions $w \in \mathbb{R}^d$ such that there exists a point $z \in \mathcal{S}$ with $w^T z \ge 0$. We use $\mathcal{W}$ to denote the collection of such directions.

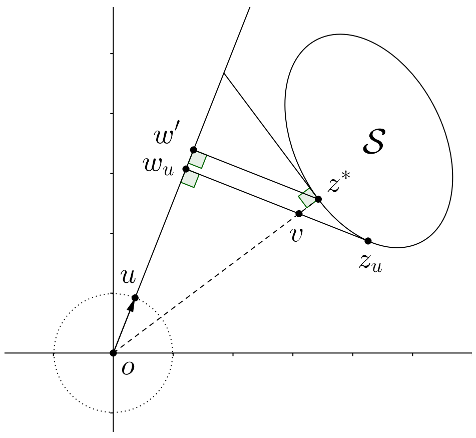

Figure 1 The equivalence between C-Hull and saddle point optimization (3).

Let u be a unit vector in $\mathcal{W}$. Denote

$$z_u := \arg\min_{z\in\mathcal{S}} \phi(u,z) = \arg\min_{z\in\mathcal{S}} u^T z.$$

By this definition, z_u is the point with smallest projection distance to u among $\mathcal{S}$ (see Figure 1). Observe that if a direction $w = c\cdot u$ ($c > 0$), then we have $\arg\min_z \phi(w,z) = \arg\min_z \phi(u,z)$. Also note that

$$\max_{w=c\cdot u:c>0} w^{\mathrm{T}} z_u - \frac{1}{2}\|w\|^2 = \max_{w=c\cdot u:c>0} \frac{1}{2}(-\|w-z_u\|^2 + \|z_u\|^2).$$

Let $w_u := \arg\max_{w=c\cdot u:c>0} \phi(w,z_u) = \arg\min_{w=c\cdot u:c>0} \|w-z_u\|^2$ be the projection point of z_u to the line ou, where o is the origin. See Figure 1 for an example. Overall, we have

$$\begin{aligned}\max_{w} \min_{\eta\in\Delta_{n_1},\xi\in\Delta_{n_2}} w^{\mathrm{T}}(A\eta - B\xi) - \frac{1}{2}\|w\|^2 &= \max_{u\in\mathcal{W}:\|u\|=1} \frac{1}{2}(-\|w_u - z_u\|^2 + \|z_u\|^2)\\ &= \max_{u\in\mathcal{W}:\|u\|=1} \frac{1}{2}\|w_u\|^2.\end{aligned}$$

The last equality is by the Pythagorean theorem. Let z^* be the closest point in $\mathcal{S}$ to the origin point. Next, we show that $\max_{u\in\mathcal{W}:\|u\|=1} \|w_u\|^2 = \|z^*\|^2$. Given a unit vector $u \in \mathcal{W}$, define w' to be the projection point of z^* to the line ou. By the definition of z_u and w_u, we have that $\max_u \|w_u\|^2 \le \|w'\|^2 \le \|z^*\|^2$. Moreover, let $u = z^*/\|z^*\|$. In this case, we have $\|w_u\|^2 = \|z^*\|^2$. Thus, we conclude that $\max_u \|w_u\|^2 = \|z^*\|^2$.

Overall, we prove that

$$\max_{u\in\mathcal{W}:\|u\|=1} \frac{1}{2}\|w_u\|^2 = \frac{1}{2}\|z^*\|^2 \min_{z\in\mathcal{S}} \frac{1}{2}\|z\|^2 = \min_{\eta\in\Delta_{n_1},\xi\in\Delta_{n_2}} \frac{1}{2}\|A\eta - B\xi\|^2.$$

Thus, C-Hull (2) is equivalent to the saddle point optimization (3). ◀

Let $\phi(w,\eta,\xi) = w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi - \|w\|^2/2$. Note that $\phi(w,\eta,\xi)$ is only linear w.r.t. η and ξ. However, in order to obtain an algorithm which converges faster, we hope that the objective function is strongly convex with respect to η and ξ. For this purpose, we can add a small regularization term which ensures that the objective function is strongly convex. This is a commonly used approach in optimization (see [3] for an example). Here, we use the entropy function $H(u) := \sum_i u_i \log u_i$ as the regularization term. The new saddle point optimization problem is as follows.

$$\max_{w} \min_{\eta\in\Delta_{n_1},\xi\in\Delta_{n_2}} \quad w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi + \gamma H(\eta) + \gamma H(\xi) - \tfrac{1}{2}\|w\|^2, \tag{4}$$

where $\gamma = \epsilon\beta/2\log n$. The following lemma describes the efficiency of the above saddle point optimization (4). We defer the proof to the full version.

► **Lemma 3.** *Let* (w^*, η^*, ξ^*) *and* $(w^\circ, \eta^\circ, \xi^\circ)$ *be the optimal solution of saddle point optimizations* (3) *and* (4) *respectively. Define* OPT *as in* (3). *Define*

$$g(w) := \min_{\eta\in\Delta_{n_1},\xi\in\Delta_{n_2}} w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi - \frac{1}{2}\|w\|^2.$$

Then $g(w^*) - g(w^\circ) \le \epsilon\mathsf{OPT}$ *(note that* $g(w^*) = \mathsf{OPT}$*).*

We call the saddle point optimization (4) the Hard-Margin Saddle problem, abbreviated as HM-Saddle. Next, we discuss ν-SVM (see [35, 10]) and again provide an equivalent saddle point optimization formulation.

► **Definition 4** (ν-SVM)**.** Given n points $x_i \in \mathbb{R}^d$ for $1 \le i \le n$, each x_i has a label $y_i \in \{+1, -1\}$. ν-SVM is the quadratic programming as follows.

$$\begin{aligned} \min_{w,b,\rho,\delta} \quad & \tfrac{1}{2}\|w\|^2 - \rho + \tfrac{\nu}{2}\textstyle\sum_i \delta_i \\ \text{s.t.} \quad & y_i(w^{\mathrm{T}}x_i - b) \ge \rho - \delta_i, \delta_i \ge 0, \quad \forall i \end{aligned} \tag{5}$$

Crisp and Burges [10] presented a geometry interpretation for ν-SVM. They proved ν-SVM is equivalent to the following problem of finding the closest distance between two reduced convex hulls.

$$\begin{aligned} \min_{\eta,\xi} \quad & \tfrac{1}{2}\|A\eta - B\xi\|^2 \\ \text{s.t.} \quad & \|\eta\|_1 = 1,\ \|\xi\|_1 = 1,\ 0 \le \eta_i \le \nu,\ 0 \le \xi_j \le \nu,\ \forall i, j \end{aligned} \tag{6}$$

We call the above problem the Reduced Convex Hull problem, abbreviated as RC-Hull. The difference between C-Hull (2) and RC-Hull (6) is that in the latter one, each entry of η and ξ has an upper bound ν. Geometrically, it means to compress the convex hull of $\mathcal{P}$ and $\mathcal{Q}$ such that the two reduced convex hulls are linearly separable. We define $\mathcal{D}_{n_1}$ to be the domain of η in RC-Hull, i.e., $\{\eta \mid \|\eta\|_1 = 1, 0 \le \eta_i \le \nu, \forall i\}$ and $\mathcal{D}_{n_2}$ to be the domain of ξ, i.e., $\{\xi \mid \|\xi\|_1 = 1, 0 \le \xi_j \le \nu, \forall j\}$. Similar to Lemma 2, we have the following lemma.

► **Lemma 5.** *RC-Hull* (6) *is equivalent to the following saddle point optimization.*

$$\mathsf{OPT} = \max_{w} \min_{\eta\in\mathcal{D}_{n_1},\, \xi\in\mathcal{D}_{n_2}} w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi - \frac{1}{2}\|w\|^2. \tag{7}$$

Proof. The proof is almost the same to the proof of Lemma 2. The only difference is that the range of the term $(A\eta - B\xi)$ is another convex set defined by $\eta \in \mathcal{D}_{n_1}, \xi \in \mathcal{D}_{n_2}$. ◄

Again, we add two entropy terms to make the objective function strongly convex with respective to η and ξ.

$$\max_{w} \min_{\eta\in\mathcal{D}_{n_1},\xi\in\mathcal{D}_{n_2}} \quad w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi + \gamma H(\eta) + \gamma H(\xi) - \tfrac{1}{2}\|w\|^2. \tag{8}$$

where $\gamma = \epsilon\beta/(2\log n)$. We call this problem a ν-Saddle problem. Similar to Lemma 3, we have the following lemma which states that ν-Saddle (8) is a $(1-\epsilon)$-approximation of the saddle point optimization (7). The proof can be found in the full version.

▶ **Lemma 6.** *Let* (w^*, η^*, ξ^*) *and* $(w^\circ, \eta^\circ, \xi^\circ)$ *be the optimal solution of saddle point optimizations* (7) *and* (8) *respectively. Define* OPT *as in* (7)*. Define*

$$g(w) := \min_{\eta \in \mathcal{D}_{n_1}, \xi \in \mathcal{D}_{n_2}} w^{\mathrm{T}} A\eta - w^{\mathrm{T}} B\xi - \frac{1}{2}\|w\|^2.$$

Then $g(w^*) - g(w^\circ) \le \epsilon\mathsf{OPT}$.

Overall, we formulate hard-margin SVM and ν-SVM as saddle point problems and prove that through solving HM-Saddle and ν-Saddle, we can solve hard-margin SVM and ν-SVM.[2]

3 Saddle Point Optimization Algorithms for SVM

In this section, we propose efficient algorithms to solve the two saddle point optimizations: HM-Saddle (4) and ν-Saddle (8). The framework is inspired by the prior work by [3]. However, their algorithm does not imply an effective SVM algorithm directly as discussed in Section 1.2. We modify the *update rules* and introduce new *projection methods* to adjust the framework to the HM-Saddle and ν-Saddle problems. We highlight that both the new update rules and projection methods are non-trivial.

First, we introduce a preprocess step to make the data vectors more homogeneous in each coordinate. Then, we explain the update rules and projection methods of our algorithm: Saddle-SVC.

For convenience, we assume that in the hard margin case $\|x_i\|^2 \le 1$ for $1 \le i \le n$. [3] Let W be the $d \times d$ Walsh-Hadamard matrix and D be a $d \times d$ diagonal matrix whose entries are i.i.d. chosen from ± 1 with equal probability. We transform the data by left-producting the matrix WD. Then with high probability, for any point x_i satisfied that [1]

$$\forall j \in [d],\ |(WDx_i)_j| \le O(\sqrt{\log n/d}).$$

Let $X^+ = WDA$ and $X^- = WDB$. It means that after transformation, with high probability, the value of each entry in X^+ or X^- is at most $O(\sqrt{\log n/d})$. This transformation can be completed in $O(nd\log d)$ time by FFT. Note that WD is an invertible matrix which represents a rotation and mirroring operation. Hence, it does not affect the optima of the problem. In fact, the "Hadamard transform trick" has been used in the numerical analysis literature explicitly or implicitly (see e.g., [14, 23, 3]). Roughly speaking, the main purpose of the transform is to make all coordinates of X more uniform, such that the uniform sampling (line 1 in Algorithm 2) is more efficient (otherwise, the large coordinates would have a disproportionate effect on uniform sampling).

After the data transformation, we define some necessary parameters. See line 4 of Algorithm 1 for details.[4] We use "$\alpha[t]$" to represent the value of variable "α" at iteration t.

[2] Some readers may wonder why the formulations of HM-Saddle and ν-Saddle only depends on (w, η, ξ) but not the offset b. In fact, according to the fact that the hyperplane bisects the closest points in the (reduced) convex hulls, it is not difficult to show that $b^* = w^{*\mathrm{T}}(A\eta^* + B\xi^*)/2$.

[3] It can be achieved by scaling all data by factor $1/\max\|x_i\|^2$ in $O(nd)$ time.

[4] Careful readers may notice that $\gamma = \epsilon\beta/(2\log n)$. But β is an unknown parameter, which is the ratio of the minimum distance to the maximum one among the points. The same issue also appears in the previous work [3]. The role of β is similar to the step size in the stochastic gradient descent algorithm. In practice, we could try several $\beta = 10^{-k}$ for $k \in \mathbb{Z}$ and choose the best one.

```
Algorithm 1 Pre-processing
Input: $\mathcal{P}$: $n_1$ points $x_i^+$ with label $+1$ and $\mathcal{Q}$: $n_2$ points $x_i^-$ with label $-1$
1: $W \leftarrow d$-dimensional Walsh-Hadamard Matrix
2: $D \leftarrow d \times d$ diagonal matrix whose entries are i.i.d. chosen from $\pm 1$
3: $X^+ \leftarrow WD \cdot [x_1^+, x_2^+, \ldots, x_{n_1}^+]$, $X^- \leftarrow WD \cdot [x_1^-, x_2^-, \ldots, x_{n_2}^-]$
4: $\gamma \leftarrow \frac{\epsilon\beta}{2\log n}$, $q \leftarrow O(\sqrt{\log n})$, $\tau \leftarrow \frac{1}{2q}\sqrt{\frac{d}{\gamma}}$, $\sigma \leftarrow \frac{1}{2q}\sqrt{d\gamma}$, $\theta \leftarrow 1 - \frac{1}{d+q\sqrt{d}/\sqrt{\gamma}}$
5: $w[0] = \mathbf{0}^{\mathrm{T}}$, $\eta[-1] = \eta[0] = \mathbf{1}^{\mathrm{T}}/n_1$, $\xi[-1] = \xi[0] = \mathbf{1}^{\mathrm{T}}/n_2$
```

```
Algorithm 2 Update Rules of Saddle-SVC
1: Pick an index $i^*$ in $[d]$ uniformly at random
2: $\delta_{i^*}^+ \leftarrow \langle X_{i^*}^+, \eta[t] + \theta(\eta[t] - \eta[t-1])\rangle$, $\delta_{i^*}^- \leftarrow \langle X_{i^*}^-, \xi[t] + \theta(\xi[t] - \xi[t-1])\rangle$
3: $\forall i \in [d]$, $w_i[t+1] \leftarrow \begin{cases} (w_i[t] + \sigma(\delta_i^+ - \delta_i^-))/(\sigma+1), & \text{if } i = i^* \\ w_i[t], & \text{if } i \neq i^* \end{cases}$
4: $\eta[t+1] \leftarrow \arg\min_{\eta \in \mathcal{S}_1} \{\frac{1}{d}(w[t] + d(w[t+1] - w[t]))^{\mathrm{T}} X^+ \eta + \frac{\gamma}{d} H(\eta) + \frac{1}{\tau} V_{\eta[t]}(\eta)\}$
5: $\xi[t+1] \leftarrow \arg\min_{\xi \in \mathcal{S}_2} \{-\frac{1}{d}(w[t] + d(w[t+1] - w[t]))^{\mathrm{T}} X^- \xi + \frac{\gamma}{d} H(\xi) + \frac{1}{\tau} V_{\xi[t]}(\xi)\}$
```

For example, $w[0]$, $\eta[0], \xi[0]$ are the initial value of w, η, ξ and are defined in line 5 of Algorithm 1.

Update Rules: In order to unify HM-Saddle and ν-Saddle in the same framework, we use $(\mathcal{S}_1, \mathcal{S}_2)$ to represent the domains $(\Delta_{n_1}, \Delta_{n_2})$ in HM-Saddle (see formula (3)) or $(\mathcal{D}_{n_1}, \mathcal{D}_{n_2})$ in ν-Saddle (see formula (7)).

Generally speaking, the update rules alternatively maximize the objective with respect to w and minimize with respect to η and ξ. See the details in Algorithm 2.

Firstly, we update w according to line 3 in Algorithm 2. It is equivalent to a variant of the proximal coordinate gradient method with l_2-norm regularization as follows.

$$w_{i^*}[t+1] = \arg\max_{w_{i^*}} -\big\{ -(\delta_{i^*}^+ - \delta_{i^*}^-)w_{i^*} + w_{i^*}^2/2 + (w_{i^*} - w_{i^*}[t])^2/2\sigma \big\} \tag{9}$$

We briefly explain the intuition of (9). Note that the term $(\delta_{i^*}^+ - \delta_{i^*}^-)$ in (9) can be considered as the term $\langle X_{i^*}^+, \eta[t]\rangle - \langle X_{i^*}^-, \xi[t]\rangle$ adding extra momentum terms $\langle X_{i^*}^+, \theta(\eta[t] - \eta[t-1])\rangle$ and $-\langle X_{i^*}^-, \theta(\xi[t] - \xi[t-1])\rangle$ for dual variable $\eta[t]$ and $\xi[t]$ respectively (see line 2 in Algorithm 2). Further, $(\langle X_{i^*}^+, \eta[t]\rangle - \langle X_{i^*}^-, \xi[t]\rangle)w_{i^*} - w_{i^*}^2/2$ is the term in the objective function (4) and (8) which are related to w. The $(w_{i^*} - w_{i^*}[t])^2/2)$ is the l_2-norm regularization term.

Moreover, rather than update the whole w vector, randomly selecting one dimension $i^* \in [d]$ and updating the corresponding w_{i^*} in each iteration can reduce the runtime per round.

The update rules for η and ξ are listed in line 4 and 5 in Algorithm 2, which are the proximal gradient method with a Bergman divergence regularization $V_x(y) = H(y) - \langle \nabla H(x), y - x\rangle - H(x)$. Similar to $(\delta_{i^*}^+ - \delta_{i^*}^-)$ in (9), we also add a momentum term $d(w[t+1] - w[t])$ for primal variable w when updating η and ξ.

Projection Methods: However, the update rules for η and ξ are implicit update rules. We need to show that we can solve the corresponding optimization problems in line 4 and 5 of Algorithm 2 efficiently. In fact, for both HM-Saddle and ν-Saddle, we can obtain explicit expressions of these two optimization problems using the method of Lagrange multipliers.

First, we can solve the optimization problem for HM-Saddle (line 4 and 5 of Algorithm 2) directly, and the explicit expressions for η and ξ are as follows. The proof can be found in the full version.

▶ **Lemma 7** (Update Rules of HM-Saddle). *For linearly separable cases, the update rules in line 4 and 5 of Algorithms 2 is equivalent to*

$$\begin{aligned} \eta_i[t+1] &\leftarrow \Phi(\eta_i[t], X^+)/Z^+, \quad \forall i \in [n_1], \\ \xi_j[t+1] &\leftarrow \Phi(\xi_j[t], X^-)/Z^-, \quad \forall j \in [n_2], \end{aligned} \tag{10}$$

where Z^+ and Z^- are normalizers that ensures $\sum_i \eta_i[t+1] = 1$ and $\sum_j \xi_j[t+1] = 1$, and

$$\Phi(\lambda_i, X) = \exp\left\{(\gamma + d\tau^{-1})^{-1}(d\tau^{-1}\log\lambda_i - y_i \cdot \langle w[t] + d(w[t+1] - w[t], X_{\cdot i})\rangle)\right\} \tag{11}$$

Note that the factors Z^+ and Z^- are used to project the value $\Phi(\eta_i[t], X^+)$ and $\Phi(\xi_j[t], X^-)$ to the domains Δ_{n_1} and Δ_{n_2}. The above update rules of η and ξ can be also considered as the multiplicative weight update method (see [4]).

Next, we consider ν-Saddle. Compared to HM-Saddle, ν-Saddle has extra constraints that $\eta_i, \xi_j \le \nu$. Thus, we need another projection process to ensure that $\eta[t+1]$ and $\xi[t+1]$ locate in domain $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$ respectively. For convenience, we only present the projection for η by the following Lemma 8. The projection for ξ is similar. Due to the space limit, we defer the proof of Lemma 8 to the full version.

▶ **Lemma 8** (Update Rules of ν-Saddle). *The following three update rules are equivalent.*

Rule 1:

$$\eta[t+1] := \arg\min_{\eta \in \mathcal{D}_{n_1}} \left\{\frac{1}{d}(w[t] + d(w[t+1] - w[t]))^{\mathrm{T}} X\eta + \frac{\gamma}{d}H(\eta) + \frac{1}{\tau}V_{\eta[t]}(\eta)\right\}$$

Rule 2:

- *Step 1:*

$$\eta_i := Z^{-1}\exp\{(\gamma + d\tau^{-1})^{-1}(d\tau^{-1}\log\eta_i[t] - \langle w[t] + d(w[t+1] - w[t]), X_{\cdot i}\rangle)\}$$

for each $i \in [n_1]$, where Z ensures $\sum_i \eta_i = 1$.
- *Step 2:*

```
while  $\varsigma := \sum_{\eta_i > \nu}(\eta_i - \nu) \neq 0$ :
    $\Omega = \sum_{\eta_i < \nu} \eta_i$
    $\forall i$,  if $\eta_i \ge \nu$,  then $\eta_i[t+1] = \nu$
    $\forall i$,  if $\eta_i < \nu$,  then $\eta_i[t+1] = \eta_i(1 + \varsigma/\Omega)$
```
(12)

Rule 3:

- *Step 1:*

$$\eta_i = Z^{-1}\exp\left\{(\gamma + d\tau^{-1})^{-1}(d\tau^{-1}\log\eta_i[t] - \langle w[t] + d(w[t+1] - w[t]), X_{\cdot i}\rangle)\right\}$$

for each $i \in [n_1]$, where Z ensures $\sum_i \eta_i = 1$.
- *Step 2: Sort η_i by the increasing order. W.l.o.g., assume that $\eta_1, \ldots, \eta_{n_1}$ is in increasing order. Define $\varsigma_i = \sum_{j \ge i}(\eta_j - \nu)$ and $\Omega_i = \sum_{j<i} \eta_j$. Find the largest index $i^* \in [n]$ such that $\varsigma_{i^*} \ge 0$ and $\eta_{i^*-1}(1 + \varsigma_{i^*}/\Omega_{i^*}) < \nu$ by binary search.*

- *Step 3:*

$$\forall i, \eta_i[t+1] = \begin{cases} \eta_i(1+\varsigma_{i^*}/\Omega_{i^*}), & \textit{if } i < i^* \\ \nu, & \textit{if } i \geq i^* \end{cases}$$

We use Rule 2 when $1/\nu$ is constant. Note that there are at most $1/\nu$ (a constant) entries η_i of value ν during the whole projection process. In each iteration, there must be at least 1 more entry $\eta_i = \nu$ since we make all entries $\eta_j > \nu$ equal to ν after the iteration. Thus, the number of iterations in (12) is at most $1/\nu$. By (12), we project η and ξ to the domains $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$ respectively. Thus, we need $O(n/\nu)$ time to compute $\eta[t+1]$. Since we assume ν is a constant, it only costs linear time.

When ν is extremely small, we use Rule 3 to project η and ξ to the domains $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$ respectively. It takes $O(n \log n)$ time because of sorting. Finally, we give our main theorem for our algorithm as follows. See the proof in the full version.

▶ **Theorem 9.** *Algorithm 2 computes $(1-\epsilon)$-approximate solutions for HM-Saddle and ν-Saddle by $\tilde{O}(d+\sqrt{d/\epsilon\beta})$ iterations. Moreover, it takes $O(n)$ time for each iteration.*

Combining with Lemmas 2, 3 and 5, we obtain $(1-\epsilon)$-approximate solutions for C-Hull and RC-Hull problems. Hence by strong duality, we obtain $(1-\epsilon)$-approximations for hard-margin SVM and ν-SVM in $\tilde{O}(n(d+\sqrt{d/\epsilon\beta}))$ time.

▶ **Theorem 10.** *A $(1-\epsilon)$-approximation for either hard-margin SVM or ν-SVM can be computed in $\tilde{O}(n(d+\sqrt{d/\epsilon\beta}))$ time.*

4 Distributed SVM

Server and Clients Model: We extend Saddle-SVC to the distributed setting and call it Saddle-DSVC. We consider the popular distributed setting: the *server* and *clients* model. Denote the server by S. Let $\mathcal{C}$ be the set of clients and $|\mathcal{C}| = k$. We use the notation $C.\alpha$ to represent any variable α saved in client C and use $S.\alpha$ to represent a variable α saved in the server.

First, we initialize some parameters in each client as the pre-processing step in Section 3. Each client maintains the same random diagonal matrix $D_{d\times d}$ and the total number of points in each type (i.e, $|\mathcal{P}| = n_1$ and $|\mathcal{Q}| = n_2$).[5] Moreover, each client C applies a Hadamard transformation to its own data and initialize the partial probability vectors $C.\eta$ and $C.\xi$ for its own points.

Formally speaking, assume there are m_1 points x_1^+, x_2^+, ..., $x_{m_1}^+$ and m_2 points x_1^-, x_2^-, ..., $x_{m_2}^-$ maintained in C. We use $\mathbf{1}^m$ to denote a vector with all components being 1. The initialization is as follows.

$$C.X^+ = WD\cdot[x_1^+, x_2^+, \ldots, x_{m_1}^+],\ C.\eta[-1] = C.\eta[0] = n_1^{-1}\mathbf{1}^{m_1}$$
$$C.X^- = WD\cdot[x_1^-, x_2^-, \ldots, x_{m_2}^-],\ C.\xi[-1] = C.\xi[0] = n_2^{-1}\mathbf{1}^{m_2}$$

We first consider HM-Saddle. The interaction between clients and the server can be divided into three rounds in each iteration.

1. In the first round, the server randomly chooses a number $i^* \in [d]$ and broadcasts i^* to all clients. Each client computes $C.\delta_{i^*}^+$ and $C.\delta_{i^*}^-$ and sends them back to the server.

[5] It can be realized using $O(k)$ communication bits.

2. In the second round, the server sums up all $C.\delta_{i^*}^+$ and $C.\delta_{i^*}^-$ and computes $S.\delta_{i^*}^+$ and $S.\delta_{i^*}^-$. We can see that $S.\delta_{i^*}^+$ (resp. $S.\delta_{i^*}^-$) is exactly $\delta_{i^*}^+$ (resp. $\delta_{i^*}^-$) in Algorithm 2. The server broadcasts $S.\delta_{i^*}^+$ and $S.\delta_{i^*}^-$ to all clients. By $S.\delta_{i^*}^+$ and $S.\delta_{i^*}^-$, each client updates w individually. Moreover, each client $C \in \mathcal{C}$ updates its own $C.\eta$ and $C.\xi$ according to the new directional vector w. In order to normalize the probability vectors η and ξ, each client sends the summation $C.Z^+$ and $C.Z^-$ to the server.
3. In the third round, the server computes $(S.Z^+, S.Z^-) \leftarrow \sum_{C\in\mathcal{C}}(C.Z^+, C.Z^-)$ and broadcasts to all clients the normalization factors $S.Z^+$ and $S.Z^-$. Finally, each client updates its partial probability vector $C.\eta$ and $C.\xi$ based on the normalization factors.

As we discuss in Section 3, for ν-Saddle, we need another $O(1/\nu)$ rounds to project η and ξ to the domains $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$.

4. Each client computes $C.\varsigma^+, C.\varsigma^-$ and $C.\Omega^+$, $C.\Omega^-$ according to (12) and sends them to the server. The server sums up all $C.\varsigma^+, C.\varsigma^-, C.\Omega^+, C.\Omega^-$ respectively and gets $S.\varsigma^+, S.\varsigma^-$, $S.\Omega^+, S.\Omega^-$. If both $S.\varsigma^+$ and $S.\varsigma^-$ are zeros, the server stops this iteration. Otherwise, the server broadcasts to all clients the factors $S.\varsigma^+, S.\varsigma^-, S.\Omega^+, S.\Omega^-$. All clients update their $C.\eta$ and $C.\xi$ according to (12) and repeat Step 4 again.

We give the pseudocode in the full version. Note that all clients in Saddle-DSVC get the same $w[t]$ in each iteration as the $w[t]$ in Saddle-SVC. Hence Saddle-DSVC has the same rate of convergence as Saddle-SVC. Finally, after $T = \tilde{O}(d + \sqrt{d/\epsilon})$ iterations (see Theorem 9) , all clients compute the same $(1-\epsilon)$-approximate solution $w = w[T]$ for SVM. W.l.o.g, let the first client send w to the server. Based on the w (at most $O(n)$ more communication cost), the server can compute the offset b, the margin for hard-margin SVM and the objective value for the ν-SVM.

Communication Complexity of Saddle-DSVC: Note that in each iteration, the server and clients interact three times for hard-margin SVM and $O(1/\nu)$ times for ν-SVM. Thus, the communication cost of each iteration is $O(k)$. By Theorem 9, it takes $\tilde{O}(d + \sqrt{d/\epsilon})$ iterations. Thus, we summarize the following theorem.

▶ **Theorem 11.** *The communication cost of Saddle-DSVC is* $\tilde{O}(k(d + \sqrt{d/\epsilon}))$.

Liu et al. [26] prove that the lower bound of the communication cost for distributed SVM is $\Omega(k \min\{d, 1/\epsilon\})$.

▶ **Theorem 12** (Theorem 6 in [26]). *Consider a set of d-dimension points distributed at k clients. The communication cost to achieve a $(1-\epsilon)$-approximation of the distributed SVM problem is at least $\Omega(k \min\{d, 1/\epsilon\})$ for any $\epsilon > 0$.*

If $d = \Theta(1/\epsilon)$, the communication lower bound is $\Omega(k(d + \sqrt{d/\epsilon}))$ which matches the communication cost of Saddle-DSVC.

References

1 Nir Ailon and Bernard Chazelle. Faster dimension reduction. *Communications of the ACM*, 53(2):97–104, 2010.

2 Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *STOC 16*, 2016.

3 Zeyuan Allen-Zhu, Zhenyu Liao, and Yang Yuan. Optimization algorithms for faster computational geometry. In *LIPIcs*, volume 55, 2016.

4 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

5 Kristin P Bennett and Erin J Bredensteiner. Duality and geometry in svm classifiers. In *ICML*, pages 57–64, 2000.

6 Marshall W. Bern and David Eppstein. Optimization over zonotopes and training support vector machines. In *WADS*, pages 111–121, 2001.

7 Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

8 Kenneth L Clarkson, Elad Hazan, and David P Woodruff. Sublinear optimization for machine learning. *Journal of the ACM (JACM)*, 59(5):23, 2012.

9 Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

10 DJ Crisp and CJC Burges. A geometry interpretation of μ-svm classifiers. *NIPS*, pages 244–251, 2000.

11 John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *JMLR*, 10(Dec):2899–2934, 2009.

12 Pedro A Forero, Alfonso Cano, and Georgios B Giannakis. Consensus-based distributed support vector machines. *JMLR*, 11(May):1663–1707, 2010.

13 Vojtěch Franc and Soeren Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *ICML 08*, pages 320–327. ACM, 2008.

14 Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

15 Bernd Gärtner and Martin Jaggi. Coresets for polytope distance. In *SOCG 09*, pages 33–42. ACM, 2009.

16 Elmer G Gilbert. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal on Control*, 4(1):61–80, 1966.

17 Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *NIPS*, volume 17, 2004.

18 Sariel Har-Peled, Dan Roth, and Dav Zimak. Maximum margin coresets for active and noise tolerant learning. In *IJCAI*, pages 836–841, 2007.

19 Elad Hazan, Tomer Koren, and Nati Srebro. Beating sgd: Learning svms in sublinear time. In *Advances in Neural Information Processing Systems*, pages 1233–1241, 2011.

20 Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *ICML 08*, pages 408–415. ACM, 2008.

21 Thorsten Joachims. Making large-scale svm learning practical. Technical report, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, 1998.

22 Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

23 Anatoli Juditsky, Fatma Kılınç Karzan, and Arkadi Nemirovski. Randomized first order algorithms with applications to ℓ_1-minimization. *Mathematical Programming*, 142(1-2):269–310, 2013.

24 Jyrki Kivinen, Alexander J Smola, and Robert C Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.

25 Eyal Kushilevitz. Communication complexity. *Advances in Computers*, 44:331–360, 1997.

26 Yangwei Liu, Hu Ding, Ziyun Huang, and Jinhui Xu. Distributed and robust support vector machine. In *LIPIcs*, volume 64, 2016.

27 Jorge López and José R Dorronsoro. Linear convergence rate for the mdm algorithm for the nearest point problem. *Pattern Recognition*, 48(4):1510–1522, 2015.

28 Yumao Lu, Vwani Roychowdhury, and Lieven Vandenberghe. Distributed parallel support vector machines in strongly connected networks. *IEEE Transactions on Neural Networks*, 19(7):1167–1178, 2008.

29 BF Mitchell, Vladimir Fedorovich Dem'yanov, and VN Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM Journal on Control*, 12(1):19–26, 1974.

30 A Navia-Vazquez, D Gutierrez-Gonzalez, Emilio Parrado-Hernández, and JJ Navarro-Abellan. Distributed support vector machines. *IEEE Trans. Neural Networks*, 17(4):1091–1097, 2006.

31 Yu Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal on Optimization*, 16(1):235–249, 2005.

32 Alon Orlitsky and Abbas El Gamal. Average and randomized communication complexity. *IEEE Transactions on Information Theory*, 36(1):3–16, 1990.

33 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12(Oct):2825–2830, 2011.

34 John C Platt. 12 fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods*, pages 185–208, 1999.

35 Bernhard Schölkopf, Alex J Smola, Robert C Williamson, and Peter L Bartlett. New support vector algorithms. *Neural computation*, 12(5):1207–1245, 2000.

36 Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Math. Program.*, 127(1):3–30, 2011.

37 Alexander J Smola, SVN Vishwanathan, Quoc V Le, et al. Bundle methods for machine learning. In *NIPS*, volume 20, pages 1377–1384, 2007.

38 Ivor W Tsang, Andras Kocsor, and James T Kwok. Simpler core vector machines with enclosing balls. In *ICML 07*, pages 911–918. ACM, 2007.

39 Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *JMLR*, 6(Apr):363–392, 2005.

40 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC 79*, pages 209–213. ACM, 1979.

41 Caoxie Zhang, Honglak Lee, and Kang G Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *Artificial Intelligence and Statistics*, pages 1398–1406, 2012.

42 Yuchen Zhang and Xiao Lin. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In *ICML*, pages 353–361, 2015.

43 Ji Zhu, Saharon Rosset, Robert Tibshirani, and Trevor J Hastie. 1-norm support vector machines. In *NIPS*, pages 49–56, 2004.